Appeal No. 2021-1858

United States Court of Appeals for the Federal Circuit

IN RE: PERSONALWEB TECHNOLOGIES, LLC,

AMAZON.COM, INC., AMAZON WEB SERVICES, INC.
Plaintiffs – Appellees

v.

PERSONALWEB TECHNOLOGIES, LLC
Defendant – Appellant

LEVEL 3 COMMUNICATIONS, LLC
Defendant

PERSONALWEB TECHNOLOGIES, LLC
Plaintiff – Appellant

LEVEL 3 COMMUNICATIONS, LLC
Plaintiff

v.

TWITCH INTERACTIVE, INC.,
Defendant – Appellee

2021-1858, 2021-1859, 2021-1860
IN RE: PERSONALWEB TECHNOLOGIES LLC

On Appeal from the United States District
Court for the Northern District of California
Honorable Beth Labson Freeman
Case Nos. 5:18-cv-00767-BLF; 5:18-cv-05619-BLF;
and 5:18-md-02834-BLF

PERSONALWEB TECHNOLOGIES, LLC'S CORRECTED PRINCIPAL APPEAL BRIEF

Michael A. Sherman
Jeffrey F. Gersh
Wesley W. Monroe
STUBBS ALDERTON & MARKILES, LLP
15620 Ventura Blvd., Floor 20
Sherman Oaks, CA 91403
Telephone: (818) 444-4528
Counsel for Plaintiff-Appellant

UNITED STATES COURT OF APPEALS FOR THE FEDERAL CIRCUIT

Amended

CERTIFICATE OF INTEREST

Case Number	2021-1858
Short Case Caption	In re: PersonalWeb Technologies, LLC
Filing Party/Entity	PersonalWeb Technologies, LLC

Instructions: Complete each section of the form. In answering items 2 and 3, be specific as to which represented entities the answers apply; lack of specificity may result in non-compliance. **Please enter only one item per box; attach additional pages as needed and check the relevant box**. Counsel must immediately file an amended Certificate of Interest if information changes. Fed. Cir. R. 47.4(b).

I certify the following information and any attached sheets are accurate and complete to the best of my knowledge.

Date: 08/27/2021

Signature: /s/ Michael A. Sherman

Name: Michael A. Sherman

1. Represented Entities. Fed. Cir. R. 47.4(a)(1).	**2. Real Party in Interest.** Fed. Cir. R. 47.4(a)(2).	**3. Parent Corporations and Stockholders.** Fed. Cir. R. 47.4(a)(3).
Provide the full names of all entities represented by undersigned counsel in this case.	Provide the full names of all real parties in interest for the entities. Do not list the real parties if they are the same as the entities. ☑ None/Not Applicable	Provide the full names of all parent corporations for the entities and all publicly held companies that own 10% or more stock in the entities. ☑ None/Not Applicable
PersonalWeb Technologies, LLC		

☐ Additional pages attached

4. Legal Representatives. List all law firms, partners, and associates that (a) appeared for the entities in the originating court or agency or (b) are expected to appear in this court for the entities. Do not include those who have already entered an appearance in this court. Fed. Cir. R. 47.4(a)(4).

☐ None/Not Applicable ☐ Additional pages attached

Maceiko IP c/o Theodore S. Maceiko	Kent, Beatty & Gordon LLP c/o Jack A. Gordan, Joshua B. Katz	
Stanley H. Thompson, Jr.	Farnan, LLP c/o Brian E. Farnan, Michael J. Farnan	
Sandeep Seth		

5. Related Cases. Provide the case titles and numbers of any case known to be pending in this court or any other court or agency that will directly affect or be directly affected by this court's decision in the pending appeal. Do not include the originating case number(s) for this case. Fed. Cir. R. 47.4(a)(5). See also Fed. Cir. R. 47.5(b).

☐ None/Not Applicable ☑ Additional pages attached

See attached Exhibit A		

6. Organizational Victims and Bankruptcy Cases. Provide any information required under Fed. R. App. P. 26.1(b) (organizational victims in criminal cases) and 26.1(c) (bankruptcy case debtors and trustees). Fed. Cir. R. 47.4(a)(6).

☑ None/Not Applicable ☐ Additional pages attached

Exhibit A

The following cases pending in this or any other court or agency may directly affect, or be directly affected by, this court's decision in the pending appeal:

Federal Circuit Cases

1. *In re PersonalWeb Techs. LLC*, Case No. 19-1918, docketed on 5/24/2019, and decided on 6/17/2020; and
2. *In re PersonalWeb Techs. LLC*, Case No. 20-1566, docketed on 3/6/2020.

District Court Cases (all pending in the Northern District of California)

1. *Amazon.com, Inc. and Amazon Web Services, Inc. v. PersonalWeb Techs., LLC et. al.,* No. 5:18-cv-00767-BLF;
2. *PersonalWeb Techs., LLC v. Twitch Interactive, Inc.,* No. 3:18-cv-05619-BLF;
3. *In re PersonalWeb Techs., LLC* No. 5:18-md-02834-BLF;
4. *PersonalWeb Techs., LLC et al. v. Buzzfeed Inc.,* No. 5:18-cv-06046-BLF;
5. *PersonalWeb Techs., LLC et al. v. Dictionary.com*, *LLC*, No. 5:18-cv-05606-BLF;
6. *PersonalWeb Techs., LLC et al. v. Oath Inc*., No. 5:18-cv-06044-BLF;
7. *PersonalWeb Techs., LLC et al. v. Patreon, Inc*., No. 5:18-cv-05599-BLF;
8. *PersonalWeb Techs., LLC et al. v. Popsugar, Inc*., No. 5:18-cv-06612-BLF;
9. *PersonalWeb Techs., LLC et al. v. Vice Media, LLC*, No. 5:18-cv-05970-BLF;
10. *PersonalWeb Techs., LLC et al. v. Vox Media, Inc*., No. 5:18-cv-05969-BLF;
11. *PersonalWeb Techs., LLC et al. v. Ziff Davis, LLC*, No. 5:18-cv-07119-BLF;
12. *PersonalWeb Techs. LLC v. Airbnb, Inc*., No. 5:18-cv-00149-BLF;
13. *PersonalWeb Techs. LLC v. Amicus FTV, Inc.,* No. 5:18-cv-00150-BLF;
14. *PersonalWeb Techs. LLC v. Atlas Obscura, Inc.,* No. 5:18-cv-04037-BLF;
15. *PersonalWeb Techs. LLC v. Atlassian, Inc*., No. 5:18-cv-00154-BLF;
16. *PersonalWeb Techs. LLC v. BDG Media, Inc.,* No. 5:18-cv-03571-BLF;
17. *PersonalWeb Techs. LLC v. Bitly, Inc*., No. 5:18-cv-03 572-BLF;

18. *PersonalWeb Techs. LLC v. Blue Apron, LLC,* No. 5:18-cv-03573-BLF;

19. *PersonalWeb Techs. LLC v. Braze Inc.,* No. 5:18-cv-04624-BLF;

20. *PersonalWeb Techs. LLC v. Brooklyn Brewery Corp.,* No. 5:18-cv-05436-BLF;

21. *PersonalWeb Techs. LLC v. Capterra, Inc*., No. 5:18-cv-03458-BLF;

22. *PersonalWeb Techs. LLC v. Cars.com, Inc*., No. 5:18-cv-05195-BLF;

23. *PersonalWeb Techs. LLC v. Centaur Media USA, Inc.,* No. 5:18-cv-03577-BLF;

24. *PersonalWeb Techs. LLC v. Cloud Warmer, Inc.,* No. 5:18-cv-03998-BLF;

25. *PersonalWeb Techs. LLC v. Cloud 66, Inc.,* No.5:18-cv-00155-BLF;

26. *PersonalWeb Techs. LLC v. Curebit, Inc.,* No. 5:18-cv-00156-BLF;

27. *PersonalWeb Techs. LLC v. Curious.com, Inc.,* No. 5:18-cv-05198-BLF;

28. *PersonalWeb Techs. LLC v. Dollar Shave Club, Inc.,* No. 5:18-cv-05373-BLF;

29. *PersonalWeb Techs. LLC v. Doximity, Inc*., No. 5:18-cv-00157-BLF;

30. *PersonalWeb Techs. LLC v. Fab Commerce & Design, Inc*., No: 5:18-cv-03578-BLF;

31. *PersonalWeb Techs. LLC v. FanDuel, Inc. and FanDuel Limited*, No. 5:18.cv-03582-BLF;

32. *PersonalWeb Techs. LLC v. Fiverr International Ltd.,* No. 5:18-cv- 03455-BLF;

33. *PersonalWeb Techs. LLC v. Food52, Inc.,* No. 5:18-cv-03579-BLF;

34. *PersonalWeb Techs. LLC v. Goldbely, Inc.,* No. 5:18-cv-00160-BLF;

35. *PersonalWeb Techs. LLC v. Goodreads, LLC*, No. 5:18-cv-05595-BLF;

36. *PersonalWeb Techs. LLC v. Gopro, Inc.,* No. 5:18-cv-00161-BLF;

37. *PersonalWeb Techs. LLC v. Heroku, Inc*., 5:18-cv-00162-BLF;

38. *PersonalWeb Techs. LLC v. Hootsuite, Inc.,* No. 5:18-cv-06615-TSH;

39. *PersonalWeb Techs. LLC v. Imgur, Inc.,* No. 5:18-cv-05596-BLF;

40. *PersonalWeb Techs. LLC v. Intuit, Inc.,* No. 5:18-cv-05611-BLF;

41. *PersonalWeb Techs. LLC v. Karma Mobility, Inc*., No. 5:18-cv-03459-BLF;

42. *PersonalWeb Techs. LLC v. Kickstarter, PBC*, No. 5:18-cv-03997-BLF;

43. *PersonalWeb Techs. LLC v. Kongregrate, Inc.,* No. 5:18-cv-04625-BLF;

44. *PersonalWeb Techs. LLC v. Leap Motion, Inc.,* No. 5:18-cv-00163-BLF;

45. *PersonalWeb Techs. LLC v. Lesson Nine GmbH,* No. 5:18-cv-03453-BLF;

46. *PersonalWeb Techs. LLC v. Match Group, LLC*, No. 5:18-cv-03462-BLF;

47. *PersonalWeb Techs. LLC v. Mavenlink, Inc.,* No. 5:18-cv-05200-BLF;

48. *PersonalWeb Techs. LI.C v. Melian Labs, Inc*., No. 5:18-cv-00165-BLF;

49. *PersonalWeb Techs. LLC v. Merkle, Inc.,* No. 5:18-cv-00409-BLF;

50. *PersonalWeb Techs. LLC v. NRT LLC and NRT New York LLC d/b/a Citi Habitats,* No. 5:18-cv-05201-BLF;

51. *PersonalWeb Techs. LLC v. Optimizely, Inc.,* No. 3:18-cv-06614-BLF;

52. *PersonalWeb Techs. LLC v. Our Film Festival, Inc. d/b/a Fandor, Inc.,* No. 5:18-cv-00159-BLF;

53. *PersonalWeb Techs. LLC v. Panjiva, Inc.,* No, 5:18-cv-03580-BLF;

54. *PersonalWeb Techs. LLC v. PayPal, Inc. formerly sued as Venmo Inc.,* No. 5:18-cv-00177-BLF;

55. *PersonalWeb Techs. LLC v. Peek Travel, Inc.,* No. 5:18-cv-04628-BLF;

56. *PersonalWeb Techs. LLC v. Quotient Technology Inc*., No. 5:18-cv-00169-BLF;

57. *PersonalWeb Techs. LLC v. Reddit, Inc*., No. 5:18-cv-00170-BLF;

58. *PersonalWeb Techs. LLC v. RetailMeNot, Inc.,* No. 5:18-cv-05966-BLF;

59. *PersonalWeb Techs. LLC v. Roblox Corporation,* No. 5:18-cv-00171-BLF;

60. *PersonalWeb Techs. LLC v. Rockethub, Inc and ELEQT Group Ltd.,* No. 5:18-cv-03583-BLF;

61. *PersonalWeb Techs. LLC v. ShareFile, LLC,* No. 5:18-cv-05202-BLF;

62. *PersonalWeb Techs. LLC v. Shopify (USA) Inc*., No. 5:18-cv-04626-BLF;

63. *PersonalWeb Techs. LLC v. Slack Techs., Inc.,* No. 5:18-cv-05600-BLF;

64. *PersonalWeb Techs. LLC v. Spokeo, Inc.,* 5:18-cv-02140-BLF;

65. *PersonalWeb Techs. LLC v. Spongecell, Inc.,* No. 5:18-cv-03564-BLF;

66. *PersonalWeb Techs. LLC v. Square, Inc.,* 5:18-cv-00183-BLF;

67. *PersonalWeb Techs. LLC v. Stack Exchange, Inc*., No. 5:18-cv-06045-BLF;

68. *PersonalWeb Techs. LLC v. StartDate Labs, Inc.*, No. 5:18-cv-05203-BLF;

69. *PersonalWeb Techs. LLC v. Stitch Fix, Inc*., No. 5:18-cv-00173-BLF;

70. *PersonalWeb Techs. LLC v. Strava, Inc*., No. 5:18-cv-04627-BLF;

71. *PersonalWeb Techs. LLC v. TastyTrade, Inc.,* No, 5:18-cv-05204-BLF;

72. *PersonalWeb Techs. LLC v. Teespring, Inc*., No. 5:18-cv-00175-BLF;

73. *PersonalWeb Techs. LLC v. Tophatter, Inc.,* No. 5:18-cv-00176-BLF;

74. *PersonalWeb Techs. LLC v. Treehouse Island, Inc*., No. 5:18-cv-05205-BLF;

75. *PersonalWeb Techs. LLC v. Trello, Inc.,* No. 5:18-cv-6045-BLF;

76. *PersonalWeb Techs. LLC v. TripAdvisor LLC,* No. 5:18-cv-05967-BLF;

77. *PersonalWeb Techs. LLC v. Under Armour, Inc.*, No.5:18-cv-00166-BLF;

78. *PersonalWeb Techs. LLC v. Upwork Global, Inc.,* No. 5:18-cv-05624-BLF;

79. *PersonalWeb Techs. LLC v. Urban Dictionary, LLC*, No. 5:18-cv-05968-BLF;

80. *PersonalWeb Techs. LLC v. Valassis Communications, Inc.,* No. 5:18-cv-05206-BLF;

81. *PersonalWeb Techs. LLC v. Vimeo, Inc.,* No. 5:18-cv-06042-BLF;

82. *PersonalWeb Techs. LLC v. Webflow, Inc.,* No. 5:18-cv-00178-BLF;

83. *PersonalWeb Techs. LLC v. WeddingWire, Inc.,* No. 5:18-cv-03463-BLF;

84. *PersonalWeb Techs. LLC v. Wework Companies, Inc.,* No. 5:18-cv-05272-BLF;

85. *PersonalWeb Techs. LLC v. Yotpo Ltd.,* No. 5:18-cv-03412-BLF;

86. *PersonalWeb Techs. LLC v. Zoom Video Communications, Inc.,* No.5:18-cv-05625-BLF.

TABLE OF CONTENTS

TABLE OF AUTHORITIES iv

STATEMENT OF RELATED CASES vi

JURISDICTIONAL STATEMENT xiii

STATEMENT OF THE ISSUES 1

STATEMENT OF THE CASE 1

SUMMARY OF THE ARGUMENT 26

ARGUMENT 30

I. STANDARD OF REVIEW 29

II. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FIRST FINDING 32

A. The District Court's Determination that PersonalWeb's Infringement Positions were Meritorious 32

B. The District Court Failed to Address PersonalWeb's Meritorious Position on Privity 34

C. The Erroneous Determination that PersonalWeb's Position on *Kessler* was Objectively Baseless 36

III. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE SECOND FINDING 38

A. PersonalWeb Did Not Change its Infringement Positions 39

B. The District Court Contradicted its Finding that PersonalWeb Changed its Infringement Positions 40

C. The District Court Erroneously Concluded that PersonalWeb Frequently Changed its Infringement Positions 41

IV. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE THIRD FINDING 42

A. The District Court Contradicted its Finding that PersonalWeb Prolonged the Case Post Claim Construction 42

B. The District Court Erroneously Concluded that PersonalWeb's Motion to Clarify was Prompted by the Threat of Sanctions.....44

C. The District Court Erroneously Supported its Third Finding Based on PersonalWeb's Failure to Proceed with One of Two Reasonable Courses of Action 45

D. The District Court's Third Finding is Contradicted by the District Court's Determination that PersonalWeb Efficiently Handled the Case Post-Claim Construction.....46

V. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FOURTH FINDING.....46

A. The Fourth Finding is Contradicted by the District Court's Requirement of Proceeding with a Single Customer Representative for all Categories 47

B. The District Court Erroneously Determined that Twitch's Removal as a Category 2 Representative Would Unravel the MDL.....48

VI. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FIFTH FINDING.....50

VII. THE DISTRICT COURT ABUSED ITS DISCRETION IN AWARDING FEES NOT ATTRIBUTABLE TO EXCEPTIONAL CONDUCT.....52

A. Erroneous Fees for Non-Exceptional Case Management Conduct....53

B. Erroneous Fees for Non-Exceptional MDL Conduct.....57

C. Erroneous Fees for Investigating and Responding to PersonalWeb's Claims 57

D. Erroneous Fees for Non-Exceptional Motions to Stay.....58

E. Erroneous Fees for Non-Exceptional Injunction Motion 59

F. Erroneous Fees for Non-Exceptional Infringement Contentions.....59

G. Erroneous Fees for Non-Exceptional Claim Construction Conduct ..60

VIII. CONCLUSION 61

ADDENDUM.....62

TABLE OF AUTHORITIES

Cases Page(s)

ATEN International Co., Ltd. v. Uniclass Technology Co., Ltd.,
932 F.3d 1371 (Fed. Cir. 2019) 30
Fox v. Vice,
563 U.S. 826 (2011).....31
Garity v. APWU National Labor Organization,
828 F.3d 848 (9th Cir. 2016) 35
Goodyear Tire & Rubber Co. v. Haeger,
137 S. Ct. 1178 (2017)..... passim
Highmark Inc. v. Allcare Health Mgmt. Sys., Inc.,
572 U.S. 559 (2014).....30
In re PersonalWeb Technologies LLC,
961 F.3d 1365(Fed. Cir. 2020) 19, 21, 27, 38
J*ohnson v. Washington Metro. Transit Auth.*,
993 F.2d 125 (D.C. Cir. 1989).....51
Lumen View Technology LLC v. Findthebest.com, Inc.,
811 F.3d 479 (Fed. Cir. 2016) 30
Molinaro v. Am. Tel. & Co.,
460 F. Supp. 673 (E.D. Pa. 1978).....16
Octane Fitness, LLC v. ICON Health & Fitness, Inc.,
572 U.S. 545 (2014)..... 31, 32
SFA Sys., LLC v. Newegg Inc.,
793 F.3d 1344 (Fed. Cir. 2015) 32
Speedtrack, Inc. v. Office Depot, Inc.,
791 F.3d 1317 (Fed. Cir. 2015) 16, 17
SunEarth, Inc. v. Sun Earth Solar Power Co., Ltd.,
839 F.3d 1179 (9th Cir. 2016) 30
Taurus IP, LLC v. DaimlerChrysler Corp.,
726 F.3d 1306 (Fed. Cir. 2013) 32
Transclean Corp. v. Jiffy Lube Int'l, Inc.,
474 F.3d 1298 (Fed. Cir. 2007) 35
Villiarimo v. Aloha Island Air, Inc.,
281 F.3d 1054 (9th Cir. 2002) 51, 52

Statutes

28 U.S.C. § 1295(a)(1)..... vi

28 U.S.C. §§ 1331 and 1338(a)..... vi
35 U.S.C. § 285..... passim
35 U.S.C. §§ 1..... vi

Other Authorities

6,928,442 (the "422 patent").....5
7,802,310 (the "310 patent").....5
8,099,420 (the "420 patent").....5
18A Fed. Prac. & Proc. Juris. §4460 36
U.S. Patent Numbers 5,978,791 (the "791 patent").....5

STATEMENT OF RELATED CASES

Pursuant to Federal Circuit Rule 47.5, Plaintiff-Appellant PersonalWeb Technologies, LLC (PersonalWeb") states that the following appeals from the same civil action were previously before, or are currently before, this Court:

1. *In re PersonalWeb Technologies LLC,* CAFC Case No. 19-1918, docketed on 5/24/2019, and decided on 6/17/2020.

2. In re PersonalWeb Technologies LLC, CAFC Case No. 20-1566, docketed on 3/17/2020, and decided on 8/12/2021.

PersonalWeb further states that the following cases pending in this court or in other courts could directly affect, or be directly affected by, this Court's decision:

District Cases

The following district court cases are all pending in the US District Court for the Northern District of California, before the Hon. Beth Labson Freeman:

1. *Amazon.com, Inc. and Amazon Web Services, Inc. v. PersonalWeb Techs., LLC et. al.,* No. 5:18-cv-00767-BLF;

2. *PersonalWeb Techs., LLC v. Twitch Interactive, Inc.,* No. 3:18-cv-05619-BLF;

3. *In re PersonalWeb Techs., LLC* No. 5:18-md-02834-BLF;

4. *PersonalWeb Techs., LLC et al. v. Buzzfeed Inc.,* No. 5:18-cv-06046-BLF;

5. *PersonalWeb Techs., LLC et al. v. Dictionary.com*, *LLC*, No. 5:18-cv-05606-BLF;

6. *PersonalWeb Techs., LLC et al. v. Oath Inc*., No. 5:18-cv-06044-BLF;

7. *PersonalWeb Techs., LLC et al. v. Patreon, Inc*., No. 5:18-cv-05599-BLF;

8. *PersonalWeb Techs., LLC et al. v. Popsugar, Inc*., No. 5:18-cv-06612-BLF;

9. *PersonalWeb Techs., LLC et al. v. Vice Media, LLC*, No. 5:18-cv-05970-BLF;

10. *PersonalWeb Techs., LLC et al. v. Vox Media, Inc*., No. 5:18-cv-05969-BLF;

11. *PersonalWeb Techs., LLC et al. v. Ziff Davis, LLC*, No. 5:18-cv-07119-BLF;

12. *PersonalWeb Techs. LLC v. Airbnb, Inc*., No. 5:18-cv-00149-BLF;

13. *PersonalWeb Techs. LLC v. Amicus FTV, Inc.,* No. 5:18-cv-00150-BLF;

14. *PersonalWeb Techs. LLC v. Atlas Obscura, Inc.,* No. 5:18-cv-04037-BLF;

15. *PersonalWeb Techs. LLC v. Atlassian, Inc*., No. 5:18-cv-00154-BLF;

16. *PersonalWeb Techs. LLC v. BDG Media, Inc.,* No. 5:18-cv-03571-BLF;

17. *PersonalWeb Techs. LLC v. Bitly, Inc*., No. 5:18-cv-03 572-BLF;

18. *PersonalWeb Techs. LLC v. Blue Apron, LLC,* No. 5:18-cv-03573-BLF;

19. *PersonalWeb Techs. LLC v. Braze Inc.,* No. 5:18-cv-04624-BLF;

20. *PersonalWeb Techs. LLC v. Brooklyn Brewery Corp.,* No. 5:18-cv-05436-BLF;

21. *PersonalWeb Techs. LLC v. Capterra, Inc*., No. 5:18-cv-03458-BLF;

22. *PersonalWeb Techs. LLC v. Cars.com, Inc*., No. 5:18-cv-05195-BLF;

23. *PersonalWeb Techs. LLC v. Centaur Media USA, Inc.,* No. 5:18-cv-03577-BLF;

24. *PersonalWeb Techs. LLC v. Cloud Warmer, Inc.,* No. 5:18-cv-03998-BLF;

25. *PersonalWeb Techs. LLC v. Cloud 66, Inc.,* No.5:18-cv-00155-BLF;

26. *PersonalWeb Techs. LLC v. Curebit, Inc.,* No. 5:18-cv-00156-BLF;

27. *PersonalWeb Techs. LLC v. Curious.com, Inc.,* No. 5:18-cv-05198-BLF;

28. *PersonalWeb Techs. LLC v. Dollar Shave Club, Inc.,* No. 5:18-cv-05373-BLF;

29. *PersonalWeb Techs. LLC v. Doximity, Inc*., No. 5:18-cv-00157-BLF;

30. *PersonalWeb Techs. LLC v. Fab Commerce & Design, Inc*., No: 5:18-cv-03578-BLF;

31. *PersonalWeb Techs. LLC v. FanDuel, Inc. and FanDuel Limited*, No. 5:18.cv-03582-BLF;

32. *PersonalWeb Techs. LLC v. Fiverr International Ltd.,* No. 5:18-cv- 03455-BLF;

33. *PersonalWeb Techs. LLC v. Food52, Inc.,* No. 5:18-cv-03579-BLF;

34. *PersonalWeb Techs. LLC v. Goldbely, Inc.,* No. 5:18-cv-00160-BLF;

35. *PersonalWeb Techs. LLC v. Goodreads, LLC*, No. 5:18-cv-05595-BLF;

36. *PersonalWeb Techs. LLC v. Gopro, Inc.,* No. 5:18-cv-00161-BLF;

37. *PersonalWeb Techs. LLC v. Heroku, Inc*., 5:18-cv-00162-BLF;

38. *PersonalWeb Techs. LLC v. Hootsuite, Inc.,* No. 5:18-cv-06615-TSH;

39. *PersonalWeb Techs. LLC v. Imgur, Inc.,* No. 5:18-cv-05596-BLF;

40. *PersonalWeb Techs. LLC v. Intuit, Inc.,* No. 5:18-cv-05611-BLF;

41. *PersonalWeb Techs. LLC v. Karma Mobility, Inc*., No. 5:18-cv-03459-BLF;

42. *PersonalWeb Techs. LLC v. Kickstarter, PBC*, No. 5:18-cv-03997-BLF;

43. *PersonalWeb Techs. LLC v. Kongregrate, Inc.,* No. 5:18-cv-04625-BLF;

44. *PersonalWeb Techs. LLC v. Leap Motion, Inc.,* No. 5:18-cv-00163-BLF;

45. *PersonalWeb Techs. LLC v. Lesson Nine GmbH,* No. 5:18-cv-03453-BLF;

46. *PersonalWeb Techs. LLC v. Match Group, LLC*, No. 5:18-cv-03462-BLF;

47. *PersonalWeb Techs. LLC v. Mavenlink, Inc.,* No. 5:18-cv-05200-BLF;

48. *PersonalWeb Techs. LI.C v. Melian Labs, Inc*., No. 5:18-cv-00165-BLF;

49. *PersonalWeb Techs. LLC v. Merkle, Inc.,* No. 5:18-cv-00409-BLF;

50. *PersonalWeb Techs. LLC v. NRT LLC and NRT New York LLC d/b/a Citi Habitats,* No. 5:18-cv-05201-BLF;

51. *PersonalWeb Techs. LLC v. Optimizely, Inc.,* No. 3:18-cv-06614-BLF;

52. *PersonalWeb Techs. LLC v. Our Film Festival, Inc. d/b/a Fandor, Inc.,* No. 5:18-cv-00159-BLF;

53. *PersonalWeb Techs. LLC v. Panjiva, Inc.,* No, 5:18-cv-03580-BLF;

54. *PersonalWeb Techs. LLC v. PayPal, Inc. formerly sued as Venmo Inc.,* No. 5:18-cv-00177-BLF;

55. *PersonalWeb Techs. LLC v. Peek Travel, Inc.,* No. 5:18-cv-04628-BLF;

56. *PersonalWeb Techs. LLC v. Quotient Technology Inc*., No. 5:18-cv-00169-BLF;

57. *PersonalWeb Techs. LLC v. Reddit, Inc*., No. 5:18-cv-00170-BLF;

58. *PersonalWeb Techs. LLC v. RetailMeNot, Inc.,* No. 5:18-cv-05966-BLF;

59. *PersonalWeb Techs. LLC v. Roblox Corporation,* No. 5:18-cv-00171-BLF;

60. *PersonalWeb Techs. LLC v. Rockethub, Inc and ELEQT Group Ltd.,* No. 5:18-cv-03583-BLF;

61. *PersonalWeb Techs. LLC v. ShareFile, LLC,* No. 5:18-cv-05202-BLF;

62. *PersonalWeb Techs. LLC v. Shopify (USA) Inc*., No. 5:18-cv-04626-BLF;

63. *PersonalWeb Techs. LLC v. Slack Techs., Inc.,* No. 5:18-cv-05600-BLF;

64. *PersonalWeb Techs. LLC v. Spokeo, Inc.,* 5:18-cv-02140-BLF;

65. *PersonalWeb Techs. LLC v. Spongecell, Inc.,* No. 5:18-cv-03564-BLF;

66. *PersonalWeb Techs. LLC v. Square, Inc.,* 5:18-cv-00183-BLF;

67. *PersonalWeb Techs. LLC v. Stack Exchange, Inc*., No. 5:18-cv-06045-BLF;

68. *PersonalWeb Techs. LLC v. StartDate Labs, Inc.*, No. 5:18-cv-05203-BLF;

69. *PersonalWeb Techs. LLC v. Stitch Fix, Inc*., No. 5:18-cv-00173-BLF;

70. *PersonalWeb Techs. LLC v. Strava, Inc*., No. 5:18-cv-04627-BLF;

71. *PersonalWeb Techs. LLC v. TastyTrade, Inc.,* No, 5:18-cv-05204-BLF;

72. *PersonalWeb Techs. LLC v. Teespring, Inc*., No. 5:18-cv-00175-BLF;

73. *PersonalWeb Techs. LLC v. Tophatter, Inc.,* No. 5:18-cv-00176-BLF;

74. *PersonalWeb Techs. LLC v. Treehouse Island, Inc*., No. 5:18-cv-05205-BLF;

75. *PersonalWeb Techs. LLC v. Trello, Inc.,* No. 5:18-cv-6045-BLF;

76. *PersonalWeb Techs. LLC v. TripAdvisor LLC,* No. 5:18-cv-05967-BLF;

77. *PersonalWeb Techs. LLC v. Under Armour, Inc.*, No.5:18-cv-00166-BLF;

78. *PersonalWeb Techs. LLC v. Upwork Global, Inc.,* No. 5:18-cv-05624-BLF;

79. *PersonalWeb Techs. LLC v. Urban Dictionary, LLC*, No. 5:18-cv-05968-

BLF;

80. *PersonalWeb Techs. LLC v. Valassis Communications, Inc.,* No. 5:18-cv-

05206-

BLF;

81. *PersonalWeb Techs. LLC v. Vimeo, Inc.,* No. 5:18-cv-06042-BLF;

82. *PersonalWeb Techs. LLC v. Webflow, Inc.,* No. 5:18-cv-00178-BLF;

83. *PersonalWeb Techs. LLC v. WeddingWire, Inc.,* No. 5:18-cv-03463-BLF;

84. *PersonalWeb Techs. LLC v. Wework Companies, Inc.,* No. 5:18-cv-05272-

BLF;

85. *PersonalWeb Techs. LLC v. Yotpo Ltd.,* No. 5:18-cv-03412-BLF;

86. *PersonalWeb Techs. LLC v. Zoom Video Communications, Inc.,* No.5:18-cv-05625-BLF.

JURISDICTIONAL STATEMENT

The District Court had subject matter jurisdiction of the action pursuant to 28 U.S.C. §§ 1331 and 1338(a). This Court has jurisdiction pursuant to 28 U.S.C. § 1295(a)(1) because the Orders on appeal constitute "a final decision of a district court of the United States…in any civil action arising under, or in any civil action in which a party has asserted a compulsory counterclaim arising under, any Act of Congress relating to patents…"—here, the patent laws of the United States, 35 U.S.C. §§ 1, *et seq*.

STATEMENT OF THE ISSUES

1. Whether the District Court abused its discretion in finding that the case was exceptional pursuant to 35 U.S.C. § 285.

2. Whether the District Court abused its discretion in awarding attorneys' fees and costs to Amazon.com, Inc., Amazon Web Services, Inc. (collectively, "Amazon"), and Twitch Interactive, Inc. ("Twitch") (collectively "Amazon" or "defendants", unless otherwise necessary to delineate the parties) on the basis that the case was exceptional pursuant to 35 U.S.C. § 285, resulting in a fee award that was punitive and not compensatory.

3. Whether the District Court abused its discretion in awarding a portion of attorneys' fees and costs to Amazon that were attributable to, and incurred as a result of, non-exceptional conduct.

STATEMENT OF THE CASE

Background on PersonalWeb and the Patents

The underlying action concerned PersonalWeb Technologies, LLC's ("PersonalWeb") efforts to enforce a valuable family of "True Name" patents (collectively, "patents") that PersonalWeb owns. Appx0723, ¶23. The patents were invented and designed to solve inefficiencies in the way computer networks in the 990s identified data in their systems. Appx0721-722, ¶¶15-16. Then, conventional computer networks could not link a data name given to the data item and the contents

of the data item. "The same file name in two different folders could refer to different data items, or two different file names could refer to the same data item." Appx7853. This limitation often congested computer networks with duplicate data and impaired the efficiency of data processing systems. *Id*.

The patents were invented in the mid-1990s to solve this problem. The patented True Name system creates a substantially unique identifier for each data item, based upon the contents of the data. *See*, *e.g*., Appx3335. That way, computers on a network can more efficiently store, manage, and access data. There were ten issued patents, with hundreds of claims; the summaries of invention in the patents described wide, potential usage and applicability, covering many aspects of distributive computing and networks, data file control, file access, cache control, *etc*. Appx0232-0542.

PersonalWeb was formed in 2010 and has continuously owned the patents since July 2011. Appx1430, ¶3. PersonalWeb's business plan called for, *inter alia*, developing products using its patents as the foundation for educational software, as well as licensing its products and patents. Appx7912, ¶2. *Id*. In furtherance of its business plan, PersonalWeb, as well as its parent company/predecessor-in-interest, Kinetech, Inc. ("Kinetech"), had utilized the patents to develop new technologies. Appx6404-6408. They have also licensed the patents to various companies, allowing these companies to expand on their products and services. Before the District Court

were multiple declarations of PersonalWeb's Non-Executive Chairman, Kevin Bermeister, detailing PersonalWeb and Kinetech's collective "approximate fifteen years of licensing activity associated with the 'True Name' patent family." Appx7912-7916, Appx9651-9654. As some examples, (1) in 2009, Kinetech licensed the patents to Skype Technologies SA in exchange for substantial cash consideration, and (2) beginning in 2011, PersonalWeb utilized the patents in connection with its efforts to develop an educational software product, in collaboration with the University of Texas. *Id*., Appx9652-9653, ¶¶8-9.

PersonalWeb's Prior Efforts to Enforce its Patents in the Texas Action

On December 8, 2011, PersonalWeb sued Amazon for patent infringement in the Eastern District of Texas, in the case entitled *PersonalWeb Technologies, LLC and Level 3 Communications LLC v. Amazon.com, Inc.; Amazon Web Services LLC; and Dropbox, Inc.*, Case No. 6:11-cv-00658 (the "Texas Action"). Appx1154. PersonalWeb recognizes that the legal significance and consequence of what PersonalWeb actually alleged and sought to recover in the Texas Action has been the subject of prior orders of this Court and the District Court. Adhering to those orders and without revisiting what PersonalWeb representatives should have known at times in the past, its Non-Executive Chairman, Kevin Bermeister, presented in his declaration before the District Court that "[i]n initiating the Texas Action in late 2011 and throughout its prosecution, I understood that it only accused the Multipart

Upload functionality of Amazon Web Service's Simple Storage Service ('S3') system." Appx1430, ¶4.

During the course of the Texas Action, PersonalWeb determined that any victory would be pyrrhic (*i.e.*, the cost of litigation would exceed the amount of potentially recoverable damages) resulting in PersonalWeb voluntarily dismissing its claims, with final judgment thereupon entered. Appx1430-1431, Appx1565-1566, Appx1152. No dispositive motions were filed in the Texas Action, nor did the court make a specific finding of non-infringement. *See* Appx1408, Appx2843.

PersonalWeb's Analysis and Diligence Before Filing this Case

Beginning in January 2018, PersonalWeb filed 85 single-defendant patent infringement actions against website operators in six judicial districts across the United States, necessitated by venue rules applicable to patent infringement disputes. Appx0037. *See*, *e.g.*, Appx2970. In its complaints, PersonalWeb alleged the defendants infringed five of the patents, *i.e.*, U.S. Patent Numbers 5,978,791 (the "791 patent"), 7,802,310 (the "310 patent"), 8,099,420 (the "420 patent"), 6,928,442 (the "422 patent"), and 7,945,544 (the "544 patent"). *See*, *e.g.*, Appx2990. The first patent issued on November 2, 1999. Appx2059. The subject patents had all expired as of the filing of the first lawsuits, with PersonalWeb seeking damages for the time periods before said expirations. *See* Appx2201, ¶18, Appx2970, ¶17.

In its initial complaints, PersonalWeb made substantially the same

infringement allegations against each defendant, centering around the defendants' (1) use of content-based ETags and use of them in serving their web site content, and (2) embedding file content fingerprints into filenames for web site content. Appx7941-7943. In the Counts for most of the patents—where PersonalWeb applied specific language to the accused methods/systems—PersonalWeb also alleged that the defendants (1) caused the content-based ETags to be generated by uploading files to Amazon S3 servers, which then served the files with the ETags to web users, and (2) embedded the fingerprints into filenames using open-source software for creating and serving web sites, called Ruby on Rails ("RoR"). The allegations in the Counts for the '791 and '442 patents, though, include no reference to Amazon S3 or RoR. *See*, *e.g.*, Appx2976-2977. Further, in the Count for the '310 patent, PersonalWeb alleged the use of content-based ETags in serving files for webpages, but did not allege embedding fingerprints in filename. Appx2981-2982.

Prior to filing, PersonalWeb, through multiple, experienced patent counsel and technical experts, conducted an extensive and diligent review of the patents, the claims, the prior art, the potential defendants, *inter partes* reviews, other post-grant proceedings with the US Patent & Trademark Office, other True Name litigation, and the Texas Action (including claim constructions issued by U.S. District Judge Rodney Gilstrap and claim charts for Preliminary Infringement Contentions). Appx1433-1434, ¶3, Appx8039-8046, Appx8098-8103, Appx8107-8109, ¶¶2-7.

The District Court rejected Amazon's characterizations of PersonalWeb's pre-filing conduct as deficient, and specifically commended PersonalWeb's "extensive efforts to investigate the infringement theories prior to bringing these cases." Appx0014. Based on this extensive investigation and analysis, PersonalWeb commenced these lawsuits. Appx1433-1434, ¶3, Appx8039-8046, Appx8098-8103, Appx8107-8109, ¶¶2-7.

In reviewing and evaluating its claims prior to filing its lawsuits, PersonalWeb's counsel was significantly assisted by Dr. Samuel Russ—an independent technical expert and a professor of electrical and computer engineering. Dr. Russ was retained to (1) evaluate and investigate whether various website operators infringed PersonalWeb's patent portfolio, and (2) analyze and verify PersonalWeb and its counsel's infringement findings. Appx8039-8046. Prior to filing, Dr. Russ provided PersonalWeb's counsel with his written expert opinions of validity and infringement. *Id*., Appx8041, ¶10. Dr. Russ based his expert opinion in part on Judge Gilstrap's prior claim construction order in the Texas Action. Appx5298, ¶64. Further, prior to and during the lawsuit, PersonalWeb was also supported by an outside technical team of engineers and computer technologists that assessed and determined the viability of infringement claims against website

operators believed to infringe the patents.[1] Appx7937, ¶4.

Amazon's Early Participation in the Lawsuits

On February 5, 2018, Amazon undertook the defense on behalf of the alleged infringers. Appx0037. Separately, Amazon filed a complaint against PersonalWeb, seeking a declaratory judgment that PersonalWeb's 2014 voluntary dismissal in the Texas Action precluded the claims in this case ("Amazon's declaratory judgment action"). Appx9399-9418. PersonalWeb counterclaimed in Amazon's declaratory judgment action for infringement against Amazon. Appx0718-738.

On February 20, 2018—two weeks after Amazon filed its complaint for declaratory judgment—Amazon filed a Motion for a Preliminary Injunction ("Injunction Motion") in Amazon's declaratory judgment action. In its Injunction Motion, Amazon asserted that "PersonalWeb's complaints are identical, alleging infringement of the same patents and accusing the same Amazon technology at issue in this case." *See* Appx7737.

PersonalWeb Moves to Transfer the Related Cases into an MDL

On February 27, 2018, PersonalWeb filed a Motion for Transfer and

[1] Notably, the District Court rejected Amazon's argument of exceptionality in PersonalWeb "su[ing] scores of disparate companies in a sprawling MDL over technology—like Amazon S3—that those defendants did not create or supply' in order to collect 'nuisance-value settlements.'" Appx0030. The District Court determined that "the record does not support a finding that PersonalWeb brought these lawsuits to collect 'nuisance-value' settlements." Appx0031.

Consolidation with the Judicial Panel ("Panel") on Multidistrict Litigation ("MDL Motion"). Appx7725-7746. PersonalWeb argued that—based upon the significant commonality of factual and legal issues between the actions—consolidation would prevent inconsistent rulings, conserve judicial and party resources, and provide convenience for the parties and witnesses. Appx7739-7745. Amazon opposed consolidation, even though Amazon asserted in its above-referenced Injunction Motion that "PersonalWeb's complaints are identical…" Appx7737. The Panel agreed with PersonalWeb, and granted the MDL Motion on June 6, 2018 ("MDL Order"). Appx0543-548.

In its MDL Motion, PersonalWeb made clear that its infringement theory involved RoR *and* S3. Appx7734-7735. PersonalWeb also emphasized that:

> … the existing and future PersonalWeb patent infringement actions present the same infringement issue: whether the accused websites that operate in the same manner, being based on the same Ruby on Rails architecture and using the same aspects of the internet protocol to communicate their authorizations and re-authorizations to control what web pages files a browser uses to render their webpages, infringe the Patents-in-Suit…Because all the PersonalWeb actions involve the same type of website operation, the resolution of those cases will necessarily involve overwhelmingly common questions of fact. Appx.7740.

In granting PersonalWeb's MDL Motion over Amazon's opposition, the Panel transferred the customer cases and Amazon's declaratory judgment action into the MDL proceeding, and assigned the cases to the District Court. The Panel made

the following findings:

> On the basis of the papers filed and hearing session held, we find that these 48 actions involve common questions of fact, and that centralization in the Northern District of California will serve the convenience of the parties and witnesses and promote the just and efficient conduct of the litigation. The actions contain similar allegations regarding the infringement, validity and enforceability of five now-expired PersonalWeb patents that, broadly, concern cloud computing concepts that ensure that internet browsers do not use outdated and no longer authorized content when rendering the defendants' business webpages for viewing. Appx0543.

The Panel ordered the transfer, despite "[r]ecent developments such as PersonalWeb's amendments to several customer action complaints, combined with some customers' alleged use of cloud services other than S3…" Appx0544. The Panel's decision did not distinguish between or specify RoR or S3—which makes sense, as some of PersonalWeb's infringement claims involved one, but not the other. Plainly stated, amending the complaints by adding or removing defendants or allegations—including by removing allegations that the infringing activity was based on the use of either S3 or RoR— would not, presumptively, change the appropriateness of consolidation in the cases, as long as there was commonality among the general nature of the alleged infringing activity.

During the Course of District Court Case Management, PersonalWeb Files Additional Lawsuits, and Amends its Complaints with Court Permission

On February 23, 2018—as PersonalWeb was preparing its MDL Motion—PersonalWeb's counsel received a letter from one of the defendants' counsel,

contending that the defendant's website "was not developed with and has not used RoR architecture from 2012 to 2016." Appx7938, Appx8112, Appx8133-8134. Based on this newly disclosed information, PersonalWeb immediately performed additional technical analysis of all defendants' websites that was aided by a computer program written specifically for the investigative effort by PersonalWeb's counsel. Appx7939-7940, ¶¶10-11, Appx8112-8113, ¶¶15-18. As a result, over the next several months, PersonalWeb discovered that (1) some of the defendants' websites did not use RoR architecture during the infringement period, (2) fingerprints in Uniform Resource Identifiers ("URIs") may be created by programs other than RoR, (3) some defendants generated content-based ETags and served website files with these ETags without using Amazon's S3, and (4) some defendants used content-based ETags, but not fingerprints in URIs. Appx7939-7942, ¶¶10-23. PersonalWeb's additional analysis *did not* result in any *new* allegations or theories of infringement. Rather, it resulted in PersonalWeb's discovery that (1) some defendants did not infringe in *all* the ways alleged in the original complaints, (2) some of the defendants performed the same infringement alleged in the original complaints, but by using servers/software other than S3 and RoR, and (3) there was insufficient evidence that five of the defendants used either content-based ETags or fingerprints in filenames. *Id*.

Consequently, beginning in late May 2018, prior to the hearing before the

Panel, PersonalWeb began amending its complaints (either because no responsive pleading had been filed in certain instances, or as specifically allowed by the District Court). Appx7940, ¶15, Appx0755, Appx0687. PersonalWeb completed amending complaints on October 3, 2018. *See*, *e.g.*, Appx0692-0717. PersonalWeb promptly dismissed the complaints against the above-referenced five defendants for which there was insufficient evidence of infringement. Appx7938, ¶8.

PersonalWeb's Infringement Contentions and Case Management Efforts

After PersonalWeb had found that the infringement by the defendants were subsets of all the original January 8, 2018 allegations and varied between defendants, PersonalWeb sought to make it easier for the District Court and the defendants to understand the amended infringement allegations. In its portion of the Preliminary Joint Management Conference Statement submitted to the District Court on September 13, 2018, PersonalWeb introduced the concept of four categories of infringing activity. Appx0605-607, Appx0795-797. For each defendant, PersonalWeb identified the categories of infringing activity that were alleged in the complaint against that defendant. *Id*. Categories 1-3 all included generating content-based ETags for files served as part of webpages. In category 1, the file was a webpage index file. In category 2, the content-based ETags was not generated by S3. In category 3, the content-based ETags was generated by S3. Category 4 was for defendants that embedded fingerprints into filenames of webpage files. Appx0561.

While not denominated as such at the time of filing of the original complaints, all four categories of conduct had been included in the original January 8, 2018 complaints. *See*, *e.g.*, Appx9655-9679.

On September 20, 2018, the District Court held a Case Management Conference ("CMC"). During the Conference, counsel for one of the website operator defendants advised that he believed that PersonalWeb's complaints were devoid of specific facts, violating Federal Rule of Civil Procedure. Appx0622-649. In response, on October 3 and 4, 2018, in conjunction with the 81 amended complaints that PersonalWeb filed pursuant to Court order, each amended complaint identified, for each defendant, the specific applicable categories, the particular file that fell within that category, the date on which the file had been served over the web, and either the content-based ETag served with the file (categories 1-3) or the fingerprint that was embedded in the filename (category 4). Appx0692-0717, Appx0756.

For some defendants, the information on the specific files identified in the amended complaints, as revealed by the specially developed computer program, showed that the files were not served by Amazon S3, but by another Amazon service, Amazon CloudFront. Unlike S3, Amazon's CloudFront is a content delivery network that can cache and serve files that a website operator hosts on S3 or a completely unrelated host system. Since CloudFront and S3 have different software

engineers, written in different languages, are priced different, and have different customers, PersonalWeb's infringement contentions (served on Amazon on October 29, 2018) included the scenario where the webpage files were served by Amazon S3, and the scenario where the webpage files were served by CloudFront. Appx1188-1200. In response, Amazon filed a Motion for Judgment on the Pleadings, asserting that PersonalWeb lacked standing to bring infringement claims against CloudFront. Appx9680-9689. The District Court denied said Motion. Appx6759.

The District Court addressed these four categories at a November 2, 2018 CMC. During the CMC, the District Court indicated that it was inclined to stay all cases, except for Amazon's declaratory judgment action and a representative case against one of Amazon's customers. The District Court explained that it would proceed with a representative case due to the concern that a verdict against Amazon in Amazon's declaratory judgment action may leave unresolved issues as to the liability of the other defendants in the customer cases. *See* Appx0836. The District Court remarked that, while no one had a "crystal ball", designating a single customer to represent the four categories of infringement was the "most efficient way to move to a resolution of the case." Appx0804-805, 7:17-8:8. The District Court rejected PersonalWeb's request to use more than one website operator as a representative case, commenting that: "So I'm not comfortable with a separate defendant from each

of the four categories, because I don't find that manageable. I'm willing to do this with a single customer." Appx0806, 9:21-24.

At the time (*i.e.*, November 2, 2018), and based on its continuing investigation bereft of any defendant discovery, PersonalWeb believed that the Twitch case likely involved infringement of all four categories. There had been no meaningful discovery concerning the allegedly infringing systems (as just the day before the November 1, 2018 CMC, PersonalWeb received Amazon's *initial* responses to the first set of discovery). Appx1157-1164. At the CMC, Amazon's counsel agreed that Twitch would be an appropriate representative. Amazon's counsel indicated that "I have actually discussed this with Twitch. And if a representative case is required, they would be amenable." Appx0802, 5:12-14. The District Court agreed, noting that "[a]s agreed to by the parties, Twitch ***seems to be*** an ideal exemplar." Appx0838 (emphasis added). The District Court acknowledged that there was a possibility that proceeding with Amazon's declaratory judgment action and the representative case against Twitch may not resolve all of the cases—however, the District Court did not "want to kick the can down the road, try the Amazon case, and find that all it did was conclude the Amazon case, and that would be really unfortunate." Appx0807, 10:5-9. Consequently, on November 27, 2018, the District Court stayed all cases, except for Amazon's declaratory judgment action and PersonalWeb's single representative customer case against Twitch. Appx0836-838.

Amazon Moves for Summary Judgment on Preclusion

On November 28, 2018, Amazon filed its first Motion for Summary Judgment. Appx0847-868. Amazon argued that, since PersonalWeb voluntarily dismissed the Texas Action, PersonalWeb's infringement action was barred by claim preclusion and the *Kessler* doctrine. *Id*.

One of the disputed issues in Amazon's first Motion for Summary Judgment was whether Amazon's customers were in privity with Amazon. *See* Appx0865. This was significant because privity is a necessary element of claim preclusion. Amazon sought to establish privity on the basis that Amazon agreed to indemnify its customers pursuant to an indemnification provision in its Customer Agreement. Appx0858. Since Amazon raised the issue of indemnification for the first time after PersonalWeb filed its lawsuit, PersonalWeb did not know about, and could not have known about, any alleged indemnification, or its scope, until post-filing. *See* Appx1416. PersonalWeb contended that there was no privity between Amazon and its website operators, who were not parties to the prior Texas Action. In challenging privity, PersonalWeb emphasized that the indemnification provisions (1) stated that Amazon had no contractual obligation to indemnify for the infringement claims, and (2) had become effective on June 28, 2017—which was six months after the period of infringement had ended. Appx1417. The District Court determined that, while there "appears to be privity…Even if the Court found no privity between Amazon

and its customers with respect to claim preclusion, the *Kessler* doctrine would fill the gap and thus preclude PersonalWeb's claims." Appx3244-3245.

Another disputed issue in the first Motion for Summary Judgment was whether the *Kessler* doctrine precluded PersonalWeb's claims. PersonalWeb argued that its claims were not barred by *Kessler*, on the basis that *Kessler* is rooted in issue preclusion, and, like issue preclusion, would apply only if the issue of infringement was actually litigated and decided in the Texas Action. Appx3256. Consequently, PersonalWeb contended that the voluntary dismissal of the Texas Action did not trigger *Kessler*. *Id*.

The District Court rejected PersonalWeb's position. The District Court relied on *Molinaro v. Am. Tel. & Co*., 460 F. Supp. 673, 676 (E.D. Pa. 1978), *aff'd sub nom. Molinaro, Appeal of*, 620 F.2d 288 (3rd Cir. 1980), which was "the only case that the parties have identified that squarely addresses the applicability of *Kessler* based on a prior dismissal with prejudice." Appx3257. *Molinaro*, a non-controlling 50-year-old district court case, had been cited by the Federal Circuit in the 2015 case of *Speedtrack, Inc. v. Office Depot, Inc*., 791 F.3d 1317, 1326 (Fed. Cir. 2015). While the *Speedtrack* decision cited *Molinaro* "for the specific proposition that a customer could invoke the *Kessler* doctrine…" (Appx3257), the *Speedtrack* decision had not addressed whether *Kessler* requires an actual adjudication of non-infringement. The District Court acknowledged as such, but determined that

"nothing in *Speedtrack* suggested that the Federal Circuit disagreed with *Molinaro* as to the preclusive effect of a dismissal with prejudice." Appx3257. PersonalWeb's counsel had carefully reviewed *Kessler* preclusion issues prior to filing suit and had articulated reasons, both pro and con, for a conclusion that *Kessler* might be found to apply and bar PersonalWeb's claims. Appx7943, ¶¶25-26

Another disputed issue in Amazon's first Motion for Summary Judgment was, for purposes of claim preclusion, whether the claims in the Texas Action were the same as the claims in this case. PersonalWeb challenged Amazon's position that the claims were the same. As part of its opposition papers, PersonalWeb submitted six declarations, including the Declaration of Lawrence Hadley (PersonalWeb's attorney who had litigated the Texas Action), and the Declaration of Kevin Bermeister. Appx1563-1566, Appx1429-1431. PersonalWeb submitted these declarations to establish that PersonalWeb had not been seeking findings in the Texas Action that the functionality at issue in the Texas Action was the same as the functionality in this case. *Id*. The declarations attested to the material differences between the infringement contentions in this case and in the Texas Action, and to the different purposes of the two actions, and different reasons for origination. *Id*. Messrs. Bermeister and Hadley's respective declarations provided that, at the time of the Texas Action, they were not aware that there were S3 website operator customers who were using ETags for an entirely different purpose of controlling

browser by directing browsers to use conditional HTTP GET requests containing ETags. *Id*.

The District Court determined that "Mr. Hadley's declaration does not create a genuine dispute of material fact because the declaration is 'uncorroborated and self-serving.'" Appx3250. In weighing the evidentiary value of Mr. Hadley's Declaration, the District Court determined that "Mr. Hadley's posthoc recollection of his litigation strategy is simply insufficient to place this issue in dispute." *Id*. The District Court commented at the February 28, 2019 hearing on the first Motion for Summary Judgment, as follows: "So here's the situation I'm in. Mr. Hadley gives me evidence of what the case was about, and I have other evidence also really emanating from Mr. Hadley of his discovery circumstances in his Complaint. I think I can ignore his Declaration, to the extent that it contradicts the record." Appx3122, 74:7-12.

Ultimately, on April 2, 2019, the District Court granted Amazon's first Motion for Summary Judgment ("first Summary Judgment Ruling"). Appx3233-3259. PersonalWeb appealed the first Summary Judgment Ruling. The Federal Circuit affirmed said Ruling, as set forth in its opinion in *In re PersonalWeb Technologies LLC* 961 F.3d 1365(Fed. Cir. 2020). PersonalWeb has filed a Writ of *Certiorari* to the United States Supreme Court, which is pending.

PersonalWeb Discovers New Information Bearing upon Twitch's Designation as a Category 2 Representative

In July 2019, PersonalWeb discovered new information bearing upon whether Twitch should remain as a category 2 representative. On July 26, 2019, PersonalWeb deposed Twitch's person most knowledgeable, James Richard. Appx5798. Based on Mr. Richard's testimony, PersonalWeb discovered, for the first time, that Twitch's web service platform is conducted in such a manner that Twitch was not representative of category 2 infringement. *Id*. Mr. Richard affirmed that there was no way for the public to have known of this information, *i.e*., PersonalWeb could not have uncovered this information with its pre-litigation investigation. Appx6766. The District Court never determined that PersonalWeb should have known that Twitch was not representative of category 2 at any earlier time.

Meanwhile, the District Court Issues its Order Construing Claims ("Construction Order") and PersonalWeb Moves for Judgment of Noninfringement Against It

The *Markman* hearing date of May 24, 2019 had been set months in advance by the District Court, in September, 2018. Appx0688. As the *Markman* hearing was upcoming, on March 1, 2019, the District Court set August 23, 2019 as the expert report submission deadline. Appx3047. To meet the expert submission deadline, PersonalWeb promptly began working with its expert, Erik de la Iglesia.

On August 16, 2019, one week before the August 23, 2019 expert submission deadline, the District Court issued its 34-page Construction Order. Appx5170-5203. When PersonalWeb received that Order, Mr. de la Iglesia's 64-page technical expert

report was practically complete. As of August 16, 2019, Mr. de la Iglesia had interpreted the term "valid license" to mean "valid rights to content", in accordance with Judge Gilstrap's claim construction order in the Texas Action, and in a circumstance where the District Court specifically found such interpretational efforts as of August 16, 2019 to be "reasonable." Appx3449, Appx5298, ¶64. Specifically, the District Court noted in its Construction Order that "PersonalWeb made reasonable, albeit unsuccessful, arguments that the terms authorized/unauthorized or licensed/unlicensed should be given their plain and ordinary meaning." Appx0014.

In its Construction Order, the District Court construed the claim term "unauthorized or unlicensed" as "not compliant with a valid license", and construed the claim term "authorization" as "a valid license." Appx5202. Immediately, the parties found themselves disagreeing over the meaning of the "licensed" versus "unlicensed"—which was not expressly construed/defined in the Construction Order. Appx5207. PersonalWeb contended that a "license" only required "valid rights to content" (consistent with Judge Gilstrap's order), while Amazon contended that a "license" required a written instrument or oral agreement. *Id*., Appx5342.

Within hours of issuance of the Construction Order, Amazon's counsel sent a letter to PersonalWeb's counsel. Appx5218. The letter stated that, based on the Construction Order, PersonalWeb had no viable patent infringement claims. The letter further advised that, if PersonalWeb failed to immediately dismiss all its claims

against Amazon and Twitch, then Amazon and Twitch would seek sanctions against PersonalWeb. *Id*. Despite the parties' prior zealous advocacy of infringement/non-infringement, the letter marked the first time that Amazon raised the specter of Rule 11 in PersonalWeb pursuing invalid infringement theories. *See* Appx8115, ¶24.

The next business day, on August 19, 2019, PersonalWeb's counsel responded. PersonalWeb proposed to stipulate to judgment of non-infringement against Amazon, as well as a portion of the claims against Twitch that were no longer viable as a result of the Construction Order. Appx5220. Amazon, however, *rejected* PersonalWeb's proposal—Amazon would not even agree to stipulate to PersonalWeb's proposal to dismiss all claims against Amazon. Appx5223. Thus, despite approximately six written communications between counsel between August 16 and 23, 2019 and phone calls, the parties could not bridge their different interpretations of the Construction Order. Appx5218-5224, Appx6371, ¶5.

On August 23, 2019, PersonalWeb's expert submitted his expert infringement opinion, acknowledging the above ambiguity with the Construction Order and incorporating Judge Gilstrap's construction from the Texas Action of "license"/"unlicensed" to resolve what was believed to be a persisting ambiguity. Appx5298, ¶64.

The next business day, on August 26, 2019, PersonalWeb filed a Motion to Clarify, seeking clarification on "whether the word 'license' in the Court's

construction 'meant something different than 'valid rights to content' (i.e., a narrower/license instrument-type of meaning)", as found by Judge Gilstrap. Appx5207. The purpose of the Motion to Clarify was to obtain clarification "before the parties and the Court spend more time on this case." *Id*. Thus, as set forth in the Motion to Clarify, if the District Court disagreed with PersonalWeb's interpretation, then "PersonalWeb will withdraw the report of the technical expert and dismiss, in order to preserve appellate rights." *Id*. Consistent with the Motion to Clarify, Mr. de la Iglesia noted in his expert report that "I may also supplement or amend my report or opinions in response to claim construction decisions…" Appx5276-5277, ¶3. On October 1, 2019, the District Court determined that PersonalWeb's Motion to Clarify was "a generally proper request" (Appx0021) and "properly submitted" (Appx5433), yet declined to clarify or supplement its Construction Order. Appx5428-5437.

The following day, October 2, 2019, PersonalWeb filed its Motion for Entry of Judgment ("Motion for Entry"). Appx5438-5444. In its Motion for Entry, PersonalWeb conceded that, in light of the Construction Order, "PersonalWeb cannot meet its burden of proving infringement. Entering final judgment of non-infringement in favor of Amazon will allow the parties to forego further litigation and conserve judicial resources in this case while preserving PersonalWeb's right to appeal…" Appx5440. Amazon opposed the Motion for Entry. Notably, in opposing

the Motion for Entry, "Amazon and Twitch sought findings of non-infringement based on additional grounds independent of the District Court's Claim Construction Order." Appx0005.

The District Court Comments on PersonalWeb's Efforts to Economically Handle the Case

On October 3, 2019, the District Court held a CMC, and commended PersonalWeb for its efficiency in handling the post-claim construction stage of the case, as follows:

> And to the extent that there's ever an issue of attorney's fees, clearly Mr. Sherman, you've made your record on the conduct of PersonalWeb to mitigate any obligation, if there is any ever. I mean, there may never been, I'm not suggesting that. But I know you are looking at the efficiency and the cost issues as well. Appx6060, 38:4-7.

Amazon Files its Second Motion for Summary Judgment

In October 2019, Amazon filed its second Motion for Summary Judgment. Appx6716-6732. Amazon "sought summary judgment of non-infringement not only based on the Court's claim construction, but also on additional grounds unrelated to the terms the Court had construed." Appx0020. Those "additional grounds" included that PersonalWeb failed to put forth any evidence to meet its burden of proof of infringement on other patents, and that PersonalWeb lacked standing to assert any claim against CloudFront. Appx6716-6732.

Since "Amazon/Twitch were no longer interested in a claim construction-only

judgment and sought a finding of non-infringement as to all the grounds raised in their motions", the District Court found that "the prolongation of the case at that stage did not rest solely on PersonalWeb's shoulders." Appx0022. "In other words, if Amazon/Twitch wanted to end the case, they could have asked the Court to hold PersonalWeb to its promise to dismiss all claims against Twitch based on the Court's claim construction. They did not. Instead, they pushed ahead with summary judgment on additional grounds." *Id*.

On February 3, 2020, the District Court granted the second Motion for Summary Judgment. Appx6734-6760. Two weeks later, in response to the District Court's request for the parties to advise the District Court as to how the granting of the second Motion for Summary Judgment should be applied to the stayed customer cases (Appx6761), PersonalWeb informed the District Court that the deposition testimony of Twitch's engineer on July 26, 2019 had revealed that Twitch did not perform the actions in category 2 and the engineer's testimony that there was no way that PersonalWeb could have discovered this before July 26, 2019. Appx6766. For this reason, the Court's ruling on the second Motion for Summary Judgment did not address the alleged category 2 infringement. *Id*. Thirteen of the stayed customer cases contained allegation of category 2 infringement and PersonalWeb requested that judgment not be entered as to these defendants for alleged category 2 infringement. *Id*.

PersonalWeb also requested that it be allowed to pursue its allegations of infringement based on the "terms of service" used by the defendants in the stayed customers cases. *Id*. at 3-4. While the Magistrate Judge had denied PersonalWeb's Motion to Amend Infringement Contentions to explicitly add this contention, no infringement contentions had been served regarding the stayed customer cases and neither the Court nor the Magistrate Judge had given any indication whether the infringement contentions in the Twitch case would automatically be applied to the stayed customer cases. Appx5157-5169.

The District Court Finds the Case Exceptional and Awards Fees and Costs

On March 20, 2020, Amazon filed its Motion for Attorney Fees and Costs ("Fee Motion"), on the basis that the case was exceptional pursuant to 35 U.S.C. § 285. Appx6776-6795. Amazon sought $6.1 million in attorneys' fees and $323,668.06 in expenses. Appx6794. The District Court heard the Fee Motion on August 6, 2020. Appx8362-8364. On October 6, 2020, the District Court determined that the case was exceptional for the following reasons:

> This case is exceptional because (1) PersonalWeb's infringement claims related to Amazon S3 were objectively baseless and not reasonable when brought because they were barred due to a final judgment entered in the Texas Action; (2) PersonalWeb frequently changed its infringement positions to overcome the hurdle of the day; (3) PersonalWeb unnecessarily prolonged this litigation after claim construction foreclosed its infringement theories; (4) PersonalWeb's conduct and positions regarding the customer cases were unreasonable; and (5) PersonalWeb submitted declarations that it should have known were not accurate. Appx0033.

Thereafter, on March 2, 2021, the District Court awarded Amazon $4,615,242.28 in fees and $203,300.10 in non-taxable costs. Appx0065. On March 11, 2021, Amazon and Twitch sought a supplemental $694,147.86 in attorneys' fees and $$11,120.97 in non-taxable costs. Appx0066. On April 19, 2021, the District Court awarded Amazon an additional $571,961.71 in attorneys' fees and $11,120.97 in non-taxable costs. Appx0069. The Orders dated October 6, 2020 ("First Order"), March 2, 2021 ("Second Order"), and April 19, 2021 ("Third Order") are collectively referred to as the "Orders".

Even assuming that exceptional conduct was properly found, the District Court awarded fees for various categories of legal services that are not attributable to any exceptional conduct. For example, the District Court erroneously awarded attorneys' fees for legal services arising from PersonalWeb's successful efforts in amending its complaint to add CloudFront as a defendant, even though the District Court denied Amazon's Motion for Judgment on that very issue and "easily resolved this motion in favor of PersonalWeb." Appx0051.

SUMMARY OF THE ARGUMENT

The record below demonstrated a hard-fought patent infringement dispute; one in which the District Court specifically commended the patent holder for its extensive pre-filing investigative efforts. In filing suit against many alleged infringers, the patent holder strived to produce efficiency and economy for all

involved by promptly seeking consolidation in an MDL proceeding (achieved over the objection of intervenor Amazon) and fully cooperating with the District Court's active case management. The patent holder has a robust history of commercializing and licensing the patents-in-suit, and foreswore any hint of "nuisance-value" settlements. Efforts to impugn it with that tactic failed, the District Court rejecting such categorization. As demonstrated below, the District Court contradicted itself in numerous, material ways on its findings of PersonalWeb's exceptional litigation conduct.

That the District Court determined applicability of both claim preclusion and of the *Kessler* doctrine—findings upheld on appeal by this Circuit, demonstrates nothing more than the patent holder's lack of success on issues that this Court itself (as respects *Kessler*) concluded in its precedential opinion did not require an actual litigation resulting in a prior finding of non-infringement. *In re PersonalWeb Technologies LLC,* 961 F.3d 1365 (Fed. Cir. 2020). Indeed, perhaps in light of what might be viewed as the imprecise contours of the *Kessler* doctrine, the US Supreme Court has requested Amazon to respond to PersonalWeb's Petition for Writ of *Certiorari*.

Regarding PersonalWeb's infringement theories, the District Court made multiple findings in its *Octane* order confirming the reasonableness of PersonalWeb's infringement position. The District Court expressly found

PersonalWeb's efforts at clarification of its Claim Construction Order to be procedurally proper, specifically commended PersonalWeb's counsel on their efforts in that regard, and claimed to equally charge both PersonalWeb and Amazon with prolonging disposition of this case post Claim Construction Order. Yet, inconsistently, the District Court's *Octane* Order concludes that PersonalWeb's conduct in that regard was exceptional.

One of the District Court's key findings undergirding its conclusion that PersonalWeb's positions regarding the customer cases was unreasonable was PersonalWeb's discovery towards the close of fact discovery that the customer "test case" case involving Twitch would not resolve every category of claimed infringement. The District Court criticized PersonalWeb's calling attention to that development, concluding that PersonalWeb was in some fashion advocating "unravel[ing] the MDL" when PersonalWeb never advanced unraveling the MDL, and when nothing in the record supports such a drastic result attributable to PersonalWeb. There is no finding that Twitch not falling within every category of alleged infringement, or even the potential necessity of considering terms of service to determine infringement, as leading to any MDL unraveling.

The final finding of PersonalWeb's "exceptional" conduct relates to two declarations it submitted in opposition to the first summary judgment motion. The District Court found one of the declarations to be insufficient to create a genuine

dispute of material fact to preclude the grant of summary judgment on claim preclusion/*Kessler*, and expressly rejected Amazon's characterization of the declarations as "sham". Neither of the declarations figured in claim construction or infringement positions, and there is *zero* linkage anywhere located, showing that these declarations figured at all (in any other way) in the prosecution or defense of this case, or to any increase in legal fees to Amazon.

In addition to the District Court abusing its discretion in making a finding of "exceptionality," the District Court further conflated the *Octane* mandated direction that the "totality of the circumstances" provides the basis of such a decision, with its recognition that there be a "causal connection" (a but-for test) between the exceptional conduct and the legal fees imposed. The Orders are bereft of any findings of Amazon legal fees incurred because of any of the five instances of PersonalWeb exceptional conduct and the Orders run afoul of the requirement that a § 285 award must be compensatory rather than punitive. Even if the "exceptionality" finding were to stand, the District Court abused its discretion in awarding $1,948,987.01 in attorneys' fees to Amazon that are attributable to non-exceptional conduct.

ARGUMENT

I. STANDARD OF REVIEW

In cases of patent infringement, "the court in exceptional cases may award reasonable attorney fees to the prevailing party." 35 U.S.C. § 285. This Court reviews a finding of exceptionality or an award of attorney fees pursuant to 35 U.S.C. § 285 under the law of the regional circuit—here, the Ninth Circuit. The Ninth Circuit reviews determinations under 35 U.S.C. § 285 for abuse of discretion. *SunEarth, Inc. v. Sun Earth Solar Power Co., Ltd*., 839 F.3d 1179, 1181 (9th Cir. 2016).

"A district court would necessarily abuse its discretion if it based its ruling on an erroneous view of the law or on a clearly erroneous assessment of the evidence." *ATEN International Co., Ltd. v. Uniclass Technology Co., Ltd*., 932 F.3d 1371, 1373 (Fed. Cir. 2019) (citing *Highmark Inc. v. Allcare Health Mgmt. Sys., Inc*., 572 U.S. 559, 561 (2014). A district court also abuses its discretion if it awards attorneys fees that are not attributable to conduct that renders the case exceptional. *See Lumen View Technology LLC v. Findthebest.com, Inc*., 811 F.3d 479, 483 (Fed. Cir. 2016).

If a district court finds that a case is exceptional, an award of any reasonable attorney fees is not automatic, but rather is within the district court's discretion. *Octane Fitness, LLC v. ICON Health & Fitness, Inc*., 572 U.S. 545, 550 (2014). Further, even if a district court exercises its discretion to award reasonable attorney fees under 35 U.S.C. § 285, the court can award only those attorney fees that were incurred as a result of the misconduct that renders the case exceptional. An award

that "extends further than that—to fees that would have been incurred without the misconduct—then it crosses the boundary from compensation to punishment." *Goodyear Tire & Rubber Co. v. Haeger*, 137 S. Ct. 1178, 1186 (2017). There must be a "causal connection" between the misconduct and the award of attorney fees, and the prevailing party "may recover 'only the portion of his fees that he would not have paid but for' the misconduct.'" *Id*., at 1186-87 (quoting *Fox v. Vice*, 563 U.S. 826, 836 (2011). A movant must establish its entitlement to attorneys' fees under 35 U.S.C. § 285 by a preponderance of the evidence. *Octane Fitness, LLC,* 572 U.S. at 557.

An exceptional case "is simply one that stands out from others with respect to the substantive strength of a party's litigating position (considering both the governing law and the facts of the case) or the unreasonable manner in which the case was litigated." *Id.* at 545. In considering the totality of the circumstances, the Supreme Court suggested that "district courts could consider 'nonexclusive' factors it previously set forth concerning a similar provision in the Copyright Act, including 'frivolousness, motivation, objective unreasonableness (both in the factual and legal components of the case) and the need in particular circumstances to advance considerations of compensation and deterrence.'" *Id.*, at 554 n.6. The basic premise of the objective test is that absent misconduct, an alleged infringer must show that the suit is objectively baseless in order for the case to be exceptional. The relevant

question in the objective test is whether any reasonable litigant could have reasonably expected success on the merits. *Taurus IP, LLC v. DaimlerChrysler Corp.*, 726 F.3d 1306, 1327 (Fed. Cir. 2013).

Fee awards are not to be used "as a penalty for failure to win a patent infringement suit." *Octane Fitness, LLC,* 572 U.S. at 548. Further, "it is the 'substantive strength of the party's litigating position' that is relevant to an exceptional case determination, not the correctness or eventual success of that position." *SFA Sys., LLC v. Newegg Inc.*, 793 F.3d 1344, 1348 (Fed. Cir. 2015).

II. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FIRST FINDING

The First Finding was that "PersonalWeb's infringement claims related to the use of Amazon S3 were objectively baseless and clearly untenable based on established Federal Circuit precedent and in light of the with-prejudice dismissal of the Texas Action." Appx0013. The District Court found that such infringement claims "were objectively baseless and not reasonable ***when brought***." Appx0033 (emphasis added). However, the District Court's own conclusions and findings in the Orders show the District Court also found the *exact opposite*.

A. The District Court's Determination that PersonalWeb's Infringement Positions were Meritorious

The District Court repeatedly explained that—even though the District Court may have determined that PersonalWeb's infringement claims were barred by claim

preclusion and the *Kessler* doctrine—that determination *did not* render PersonalWeb's claims objectively baseless or unreasonable. Notably, Amazon's position in *Octane* briefing was that PersonalWeb's "infringement theory was refuted by even a cursory review of the publicly available HTTP specification" and that on the substantive merits of the patents themselves, PersonalWeb had "no viable infringement theory." Appx7786.

At the time that PersonalWeb commenced its lawsuit in January 2018, PersonalWeb did not have the benefit of an applicable claim construction order in this case. As a result, PersonalWeb constructed its infringement claims consistent with Judge Gilstrap's claim construction order in the Texas Action. Appx5298, ¶64. The District Court determined that "PersonalWeb made reasonable, albeit unsuccessful, arguments that the terms authorized/unauthorized or licensed/unlicensed should be given their plain and ordinary meaning." Appx0014.

In endorsing PersonalWeb's position, the District Court concluded: "Regarding Judge Gilstrap's claim construction, PersonalWeb correctly notes this Court's conclusion that Judge Gilstrap's claim construction order was not 'directly on point with the instant dispute' and ***therefore, the merit of PersonalWeb's infringement positions were not clear until after this Court issued its Claim Construction Order***." Appx0014 (emphasis added). This determination contradicts the District Court's First Finding that PersonalWeb's infringement claims were

objectively baseless or unreasonable *when brought*.

The District Court observed: "Regarding PersonalWeb's infringement theories, while Amazon has pointed out several meaningful deficiencies in PersonalWeb's contentions, as evidenced by Amazon's complete success at summary judgment, ***those theories were not baseless or unreasonable from the start***." Appx0014 (emphasis added). The District Court elaborated: "The declarations submitted by PersonalWeb demonstrate its extensive efforts to investigate the infringement theories prior to bringing these cases and the Court is persuaded that, ***while unsuccessful, PersonalWeb presented reasonable infringement arguments throughout the case, including at summary judgment***." Appx0014-15 (emphasis added). The Order therefore provides that "***PersonalWeb's infringement positions were not objectively baseless and thus, do not support a finding of an exceptional case***." Appx0015 (emphasis added).

B. The District Court Failed to Address PersonalWeb's Meritorious Position on Privity

Independently, the First Finding is premised on a misapplication of the law. That the District Court found the claims baseless when brought logically means *only* that some aspect of claim preclusion operated in this case. To establish claim preclusion, the defendant has the burden of proving each of the three elements, *i.e.*, the prior lawsuit (1) reached a final judgment on the merits, (2) involved identical parties or privies, and (3) involved the same claim or cause of action. *Garity v.*

APWU National Labor Organization, 828 F.3d 848, 855 (9th Cir. 2016).

If the defendant proves only two of the three elements, then claim preclusion *does not* apply. Thus, before reaching any conclusion in its determination of exceptional conduct justifying a fee award that PersonalWeb's infringement claims were objectively baseless and barred by claim preclusion, the District Court would have needed to determine that PersonalWeb had no reasonable or meritorious position on *any* of the elements of claim preclusion.

The District Court's First Order found that "PersonalWeb's arguments against the application of claim preclusion were not well founded." Appx0009. At the risk of understatement, that sums up why PersonalWeb lost on claim preclusion. However, nowhere in the context of exceptional litigation conduct did the District Court identify or explain why PersonalWeb's arguments before the District Court citing to and arguing applicability of both (1) *dicta* in this Court's decision in *Transclean Corp. v. Jiffy Lube Int'l, Inc*., 474 F.3d 1298, 1306 (Fed. Cir. 2007), that "a manufacturer or seller of a product who is sued for patent infringement typically is not in privity with a party…ordinarily such parties are not so closely related and their litigation interests are not so nearly identical that a patentee's suit against one would bar a second action against the other under the doctrine of claim preclusion" or (2) a leading federal practice treatise in synthesizing privity relationships (18A Fed. Prac. & Proc. Juris. §4460 Commercial Relationships (2d ed.)), concluding that

"parties on opposite sides of a contract ordinarily do not have authority to bind each other by litigation with third parties," were each objectively baseless or unreasonable when applied to this fact pattern.

Aside from the conclusion of "not well founded," in the Orders, there is nothing more. The Orders contain zero references to the word "privity" or "same party" (or any functional equivalent thereof), and no references to why citing to this Court's *dicta* or a leading federal practice treatise on manufacturer/customer privity was exceptional. This fatal omission necessarily nullifies the First Finding.

PersonalWeb did not know, and could not have known, about the indemnification provision until after the lawsuit, since Amazon raised the indemnification provision after PersonalWeb filed its lawsuit (and was not meaningfully developed until Amazon's first Motion for Summary Judgment). *See* Appx0858. Moreover, the indemnification provision stated that Amazon had no contractual obligation to indemnify for the infringement claims; and the indemnification agreement became effective on June 28, 2017, which was six months after the patents expired. Appx2408, 49:5-21. PersonalWeb had reasonable, meritorious reasons for pursuing the testing of Amazon's claim of indemnification up through the time of the ruling on the first Summary Judgment Ruling.

C. The Erroneous Determination that PersonalWeb's Position on *Kessler* was Objectively Baseless

As support for the District Court's conclusion that "PersonalWeb's claims

related to S3 were clearly barred based on existing Federal Circuit precedent on the Kessler doctrine and thus, were objectively unreasonable when brought," the District Court relied on this Court affirming the District Court's earlier determination that *Kessler* precluded some of PersonalWeb's claims. In particular, the District Court emphasized that "the Federal Circuit rejected PersonalWeb's assertion that the *Kessler* doctrine was essentially equivalent to issue preclusion – and did so not by setting new precedent, but by reiterating its existing caselaw and explaining that the Kessler doctrine precludes relitigation of claims that were brought or could have been brought in the prior action." Appx0009 (emphasis omitted).

The reasonable inference from this excerpt is that the District Court determined that PersonalWeb's position on *Kessler* was "objectively unreasonable when brought" because this Court affirmed the District Court's determination that *Kessler* precluded some of PersonalWeb's claims. Erroneously, in finding that the infringement claims were unreasonable "when brought", the District Court *could not have* relied on this Court's June 17, 2020 opinion, which occurred two-and-a-half years *after* PersonalWeb filed its lawsuit in January 2018. Further, this Court's precedential decision added significantly to the body of law surrounding the reach and scope of the hundred-year-old *Kessler* doctrine. *See* United States Court of Appeals for the Federal Circuit, Internal Operating Procedures, IOP#10.

Dispelling any notion of well-established *Kessler* jurisprudence, this Court

noted in its precedential decision that *Kessler* is a "separate and less frequently invoked doctrine" and cited to policy considerations—*i.e.*, this Court observed that "the policy that drove the Supreme Court's decision in Kessler would be ill-served by adopting the rule proposed by PersonalWeb." *In re PersonalWeb Technologies LLC*, 961 F.3d at 1378. The District Court acknowledged that, aside from a 50-year-old non-controlling district court case, no other case applied *Kessler* to situations in which the issue of non-infringement was not actually litigated. That the District Court disagreed with PersonalWeb's position on *Kessler* otherwise does not make PersonalWeb's position frivolous or without merit. Perhaps to that point, the District Court observed that "the baselessness of PersonalWeb's claims related to the use of S3 plays a limited role in the Court's 'totality of circumstances' analysis for finding this case exceptional." Appx0013. "Limited role" or not (and whatever that means), such considerations ought not have played *any* role. Ultimately the last word on *Kessler* will be had by the United States Supreme Court in ruling on PersonalWeb's pending Petition for Writ of *Certiorari*.

III. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE SECOND FINDING

The District Court's Second Finding was that "PersonalWeb frequently changed its infringement positions to overcome the hurdle of the day." Appx0019. The Second Finding reflects two erroneous conclusions—*i.e.*, first, that PersonalWeb "changed" its infringement positions, and second, that PersonalWeb

did so "frequently".

A. PersonalWeb Did Not Change its Infringement Positions

The Second Finding centered around a common, non-exceptional occurrence in lawsuits—in pursuit of ethically proper zealous advocacy, a party reasonably emphasizes what it believes to be meritorious facts at one stage in the proceeding, and subsequently emphasizes other meritorious facts later. The District Court focused on the arguments that PersonalWeb asserted before the MDL Order, and after the MDL Order. Before the MDL Order, PersonalWeb emphasized that the alleged infringement involved the use of RoR *and* S3. The District Court noted that, after the MDL Order, PersonalWeb focused on and emphasized that the alleged infringement involved the use of RoR, without a proportional emphasis on S3. Appx0016.

At all times, PersonalWeb's "infringement theory" was and remained the same. As PersonalWeb consistently alleged, the websites infringe when they (1) generate a fingerprint of the content of files when compiled for inclusion in a webpage, (2) have a unique content-based E-Tag value generated for files included in webpages, and (3) authorize new content and re-authorize what cached web page files may be used to render their web pages, and communicate those authorizations using HTTP 200 AND HTTP 304 messages. Appx0692-716, Appx2966-2990, Appx7941-7943, ¶¶19-24. The four categories that PersonalWeb identified in

September 2018 were not new infringement theories, but rather were all subsets of the same infringement theory that PersonalWeb pleaded in its original complaints. *Id*. The notion that emphasizing different parts of a consistent, overall theory at different times amounts to a "frequent shifting" in infringement theories is factually and legally specious, and cannot support a finding of exceptionality.

B. The District Court Contradicted its Finding that PersonalWeb Changed its Infringement Positions

The District Court contradicted the Second Finding. In describing PersonalWeb's emphasis of different facts at different times, the District Court noted that: "To the extent that PersonalWeb argues it did not hide the ball as to Ruby on Rails from the MDL Panel, the Court agrees. That said, PersonalWeb ***clearly emphasized each Defendant's use of Amazon S3*** as a common issue supporting its centralization efforts . . . But after the MDL was created . . . PersonalWeb distanced its infringement position from S3 and ***put its focus on Ruby on Rails***." Appx0016 (emphasis added). In elaborating on the above, the District Court further found that PersonalWeb "did not hide the ball as to Ruby on Rails from the MDL Panel…" *Id*. The District Court could not have found that PersonalWeb frequently shifted its legal infringement theories, as any such finding would have necessarily required something more than *not* hiding the ball, and something more than "emphasizing" S3 at an earlier stage, and "focusing on" RoR at a later stage—especially since S3 and RoR *are part of the same argument*. The language of the Order expressly

contradicts the Second Finding that PersonalWeb "frequently shifted" infringement positions.

C. The District Court Erroneously Concluded that PersonalWeb Frequently Changed its Infringement Positions

The only other instance of changed infringement positions the District Court pointed to was PersonalWeb's inclusion of CloudFront in its infringement contentions served on October 29, 2018, when all but the Amazon case had been stayed since very early in the cases. The suggestion that PersonalWeb's efforts to add a new Amazon product feature during the pendency of the lawsuit amounts to a shifting legal theory is a non-sequitur. In fact, in part of its Fee Motion, Amazon sought to recover its fees incurred in connection with its Motion for Judgment, on the basis that "PersonalWeb accessed CloudFront for the first time ten months into the case—one of the changes in infringement positions that the Court found made the case exceptional.'" Appx8975. The District Court rejected Amazon's position, on the basis that "[t]he Court easily resolved this motion in favor of PersonalWeb." Appx0051. Having found that Amazon was not entitled to recover its attorneys' fees in challenging PersonalWeb's allegations against CloudFront, the District Court could not have determined that the CloudFront allegations rendered the case exceptional.

The District Court also used "shifting positions" at one point to describe PersonalWeb's conduct regarding the stayed customer cases (Appx0025-26), yet the

District Court separated this conduct from the category of "frequently changed… infringement positions." Appx0033. Nevertheless, characterizing PersonalWeb's conduct regarding the stayed customer cases as "shifting positions" is incorrect. The District Court criticized PersonalWeb for not bringing its discovery of facts, known to Twitch since the beginning of the case, to the District Court's attention sooner. Also, despite all customer cases (other than Twitch) being stayed when the Magistrate Judge denied leave to amend the Twitch infringement contentions, the District Court criticized PersonalWeb for asking, *for the first time*, whether the denial should apply to the stayed customer cases. Appx0018-19.

IV. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE THIRD FINDING

The Third Finding was that "PersonalWeb unnecessarily prolonged this litigation after claim construction foreclosed its infringement theories." Appx0033. Yet, the District Court (1) expressly validated PersonalWeb's post-claim construction efforts to streamline (and thus avoid prolonging) the lawsuit, and (2) recognized that Amazon was responsible for unnecessarily prolonging the case in certain respects.

A. The District Court Contradicted its Finding that PersonalWeb Prolonged the Case Post Claim Construction

In opposing the Motion for Entry, Amazon prolonged the case because it "sought findings of non-infringement based on additional grounds independent of

the District Court's Claim Construction Order." Appx0005. Then in October 2019, Amazon filed its second Motion for Summary Judgment. Amazon "sought summary judgment of non-infringement not only based on the Court's claim construction, but also on additional grounds unrelated to the terms the Court had construed." Appx0020. Since "Amazon/Twitch were no longer interested in a claim construction-only judgment and sought a finding of non-infringement as to all the grounds raised in their motions", the District Court found that "the prolongation of the case at that stage did not rest solely on PersonalWeb's shoulders." Appx0022. "In other words, if Amazon/Twitch wanted to end the case, they could have asked the Court to hold PersonalWeb to its promise to dismiss all claims against Twitch based on the Court's claim construction. They did not." *Id*.

Post-Construction Order, Amazon prolonged the case by seeking summary judgment on grounds unrelated to said Order. The Third Finding is directly contradicted by the District Court's finding that "the prolongation of the case at that stage did not rest solely on PersonalWeb's shoulders." Appx0022. Yet, in abuse of its discretion, in finding that PersonalWeb unnecessarily prolonged the case, the District Court placed the burden entirely on PersonalWeb's shoulders. In doing so, the District Court violated the legal prohibition that (as quoted in the Second Order): "As such, 'the court can shift only those attorney's fees incurred because of the misconduct at issue.' An award that 'extends further than that — to fees that would

have been incurred without the misconduct—then…crosses the boundary from compensation to punishment. Hence, a 'causal connection' is required between the misbehavior and the legal fees imposed…" Appx0039.

B. The District Court Erroneously Concluded that PersonalWeb's Motion to Clarify was Prompted by the Threat of Sanctions

The Third Finding is further nullified by the District Court's conclusion that "[i]t was improper for Mr. de la Iglesia to 'interpret' the Court's construction and apply his 'interpretation' to the infringement analysis." Appx0021. The District Court elaborated that, although PersonalWeb's Motion to Clarify was "a generally properly request", PersonalWeb "did so *after* it was faced with Amazon/Twitch's threat of sanctions under Rule 11 (ECF 507 at 2) and *after* it instructed its expert to apply the improper interpretation." Appx0021 (emphasis in original). The reasonable inference, according to the District Court, is that PersonalWeb must have filed its Motion to Clarify in response to the letter threatening sanctions (as opposed to the Construction Order). In reaching that conclusion, the District Court overlooked the most important fact—*i.e.*, that the District Court issued its Construction Order just hours before Amazon's letter threatening sanctions. Given this gap of only hours, the District Court had no basis to find or even suggest that PersonalWeb filed its Motion to Clarify in a strategic effort to simply avoid the threat of sanctions—*especially* considering that the District Court determined that PersonalWeb's position was 'reasonable, albeit unsuccessful" and that its Motion to

Clarify was a "proper request." Appx0014, Appx0021.

C. The District Court Erroneously Supported its Third Finding Based on PersonalWeb's Failure to Proceed with One of Two Reasonable Courses of Action

The Third Finding is belied by another contradiction. In criticizing PersonalWeb's post construction efforts, the District Court explained that "PersonalWeb's reasonable course of action would have been to stipulate to non-infringement in both Amazon and Twitch cases immediately after claim construction when it received Amazon's Rule 11 letter and proceed to appeal." Appx0022. Yet, the District Court contradictorily found that this approach would not have been possible or accepted by Amazon because Amazon based the second Motion for Summary Judgment "on additional grounds independent of the Claim Construction Order.*"* Appx0023.

Alternatively, the District Court suggested that "PersonalWeb could have immediately sought clarification of the Court's Claim Construction Order before instructing its expert to 'interpret' the Court's construction." Appx0022. Through its Motion to Clarify, t*hat is exactly what PersonalWeb did*, although, ultimately, the District Court declined to adopt Judge Gilstrap's construction of the term "licensed/unlicensed." Appx5202. The District Court even endorsed the Motion to Clarify as "proper" (over Amazon's strident opposition of procedural irregularity) and noted that PersonalWeb's position in the Motion was "reasonable, albeit

unsuccessful…" and its position on infringement "reasonable … throughout the case" Appx0014-15.

In sum, having found that PersonalWeb's Motion to Clarify was proper and reasonable, the District Court could not have simultaneously found that the Motion to Clarify that was ruled on prior to Amazon filing its second summary judgment motion, unnecessarily prolonged the case.

D. The District Court's Third Finding is Contradicted by the District Court's Determination that PersonalWeb Efficiently Handled the Case Post-Claim Construction

The Third Finding is contradicted by the District Court's endorsement of PersonalWeb's post claim construction efforts at the October 3, 2019 CMC. There, the court commented: "And to the extent that there's ever an issue of attorney's fees, clearly Mr. Sherman, you've made your record on the conduct of PersonalWeb to mitigate any obligation…" which is consistent with the court's observation in the First Order that "setting aside the claim construction 'clarification' issue, PersonalWeb endeavored to avoid prolonging this case." Appx6060, 38:4-7, Appx0021.

V. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FOURTH FINDING

The District Court abused its discretion in finding that "PersonalWeb's conduct and positions regarding the customer cases were unreasonable". Appx0033. Specifically, the District Court took issue with PersonalWeb's discovery in July

2019 that Twitch was not an adequative category 2 representative. The District Court concluded that removing Twitch as the representative case for that category would "unravel" the MDL. Appx0025. The Fourth Finding is not based in law or fact.

A. The Fourth Finding is Contradicted by the District Court's Requirement of Proceeding with a Single Customer Representative for all Categories

The Fourth Finding is contradicted by the District Court's determination that, based on the information available as of November 2018, and as agreed to by the parties, Twitch "seems to be the ideal exemplar" as the single customer representative for all categories. Appx0838. Counsel and the District Court acknowledged that this was a belief, and could be impacted by information learned during discovery. Hence, the District Court stated that Twitch "***seems to be*** an ideal exemplar." *Id*. (emphasis added). Further, at the November 2, 2018 CMC, the District Court explained: "***None of us has a crystal ball…I'm not asking for a binding commitment***, I'm really — so I'm looking for the best, most efficient way to move to a resolution of the case." Appx0804-805, 7:17-8:8 (emphasis added). According to Twitch's designated representative, PersonalWeb could not have discovered this information earlier. *See* Appx6766. To that point, there was no adjudication, nor could there have been, that PersonalWeb should have known that Twitch was not representative of Category 2.

Thus, the District Court cannot base any finding of exceptionality on

PersonalWeb's subsequent discovery that Twitch's web service platform is conducted in such a manner that challenged the parties' earlier belief that Twitch was an adequate category 2 representative.

B. The District Court Erroneously Determined that Twitch's Removal as a Category 2 Representative Would Unravel the MDL

Further, contrary to the District Court's erroneous determination, the removal of Twitch as the category 2 representative would not have "unraveled" the MDL. Rather, the action would have proceeded with one of the thirteen stayed customer cases on the narrow undecided issues as to category 2 alleged infringement. Indeed, the District Court recognized and accepted this distinct possibility when it selected Twitch as the representative case, stating: "This gives me better hope that with a verdict, in an Amazon and customer case, that it leads you to a mediator who takes care of the case for us. I mean, that's all a bellwether ever does. A bell weather isn't binding on anybody else, I don't even think of this as a bell weather, actually, but you know, because that name means lots of different things." Appx0807, 10-15.

The District Court neither addressed nor explained how Twitch ceasing to be representative threatened to "unravel" the MDL, as opposed to illustrating how the MDL achieved its purpose by eliminating all but a few issues for the remaining thirteen cases. In fact, in the MDL Order, the Panel did not draw any distinction between S3 or RoR (as some infringement claims did not involve S3, and other infringement claims did not involve RoR) and would continue to apply to the thirteen

category 2 defendants. The four categories were, in no way, inconsistent with the MDL Order, or could have, in any way, "unraveled" the MDL.

The District Court's Fourth Finding appears to have been linked to the court's comment during an August 6, 2020 hearing that it is not Amazon's "job to prove" PersonalWeb's case. Appx8401. Yet, PersonalWeb was in no way reliant on Amazon to "prove" its case, as there were other defendants that could have been a representative case for category 2. Further, as an officer of the court, Twitch's counsel was obliged to not permit a framework of litigation to move forward where Twitch is the only party with the information necessary to show that it is not representative of the category 2 infringement. This is particularly so when, at the November 2, 2018 hearing, the District Court originally suggested using a different defendant, Centaur Media, as the representative case. Appx0802, 4:22-5:2. Indeed, it was Amazon's counsel who suggested using Twitch instead of Centaur Media: "There are a number of customers similarly [in all four categories]…So if you look on page 2, an example is Twitch Interactive, which I also represent…and I have actually discussed this with Twitch. and if a representative case is required, they would be amenable." Appx0802, 5:12-14 (brackets added). This statement that Twitch, a wholly owned Amazon subsidiary, was "amenable" to serve as the category 2 representative is perhaps the strongest evidence that it was Amazon that believed Twitch was the ideal single-customer representative. Nonetheless, the

District Court did not address why Twitch—the only party before the Court as of November 2, 2018 with the relevant information bearing upon the propriety of using Twitch as the category 2 representative because Twitch's computer architecture operated in a certain way—was not obligated to apprise the District Court that Twitch was not representative of the category 2 infringement allegations (instead of taking the position that Twitch was "amenable").

VI. THE DISTRICT COURT ABUSED ITS DISCRETION IN MAKING THE FIFTH FINDING

The District Court found that "PersonalWeb submitted declarations that it should have known were not accurate." Appx0033. The Fifth Finding is contradicted by the Orders and cannot support a finding of exceptionality.

The District Court characterized Mr. Hadley's Declaration as "uncorroborated and self-serving", and determined that "Messrs. Hadley and Bermeister should have been 'aware' that conditional GET was part of the Texas Action." Appx0029. The court did not determine that either declarant was *actually aware* that years later a court would conclude that some passing mentions of conditional GETs apart from the uploading feature meant that the operation of downloading was being pursued in the Texas Action. The court rejected Amazon's argument that they were "sham" declarations, determining that "there was no showing of blatant misrepresentation or bad faith", and found that Mr. Hadley's Declaration did not amount to a breach of duty of candor. Appx0027-29. While the declarations were admittedly unpersuasive,

these findings belie exceptionality. Nonetheless, in abuse of its discretion, the District Court proceeded to "consider these declarations as one factor in the 'totality of the circumstances' related to PersonalWeb's unreasonable conduct in this litigation." Appx0029. The District Court had no factual or legal basis to do so.

The District Court could not have premised its finding of exceptionality on its determination that Mr. Hadley's Declaration was "uncorroborated and self-serving." As addressed above, in characterizing Mr. Hadley's Declaration as such, the District Court misconstrued that it made that assessment *from an evidentiary perspective* when ruling on Amazon's first Motion for Summary Judgment. After all, the terms "uncorroborated and self-serving" did not originate from the District Court. Rather, the District Court was quoting from the following excerpt from *Villiarimo v. Aloha Island Air, Inc*. 281 F.3d 1054, 1061 (9th Cir. 2002): "However, this court has refused to find a 'genuine issue' where the only evidence presented is 'uncorroborated and self-serving' testimony." Appx0028. That excerpt from *Villiarimo* was followed by a citation to J*ohnson v. Washington Metro. Transit Auth*., 993 F.2d 125, 128 (D.C. Cir. 1989), along with the parenthetical explanation, "discussing cases in which self-serving testimony uncorroborated by other evidence did not create a genuine issue of material fact." *Villiarimo*, 281 F. 3d at 1061.

Based on *Villiarimo* and *Johnson*, it is not uncommon for courts to determine that declarations are "uncorroborated and self-serving" for purposes of analyzing a

genuine issue of material fact in connection with motions for summary judgment. Through what the District Court concluded was Mr. Hadley's "insufficient" "posthoc recollection" (Appx3250), there was nothing "exceptional" about Mr. Hadley's Declaration.

Finally, the court acknowledged that "self-serving" advocacy does not support a finding of exceptionality. In analyzing Amazon's conduct, the District Court determined that: "While Amazon/Twitch at times failed to provide the full context of the certain cited portions of the record – e.g., PersonalWeb's representation to the MDL panel (Motion at 3) and the colloquy between the Court and PersonalWeb as to whether the patents-in-suit covered the entire Internet (Motion at 10) – none raise to the level of 'falsehood' PersonalWeb suggests." Appx0032. The District Court acknowledged that (1) "self serving" zealous advocacy is expected and part of litigation, and (2) in the absence of any falsehood or misrepresentation, the fact that a pleading or position is self-serving does not make it exceptional. For these reasons, the declarations do not, and cannot, support any finding of exceptionality.

VII. THE DISTRICT COURT ABUSED ITS DISCRETION IN AWARDING FEES NOT ATTRIBUTABLE TO EXCEPTIONAL CONDUCT

In the Second and Third Orders, the District Court awarded Amazon a total of $5,187,203.99 in attorneys' fees and $214,421.07 in costs. Appx0065, Appx0069. In awarding as such, the District Court abused its discretion because, of the $5,187,203.99 in attorneys' fees awarded, $1,948,987.01 is not attributable to any

exceptional conduct. In other words, although the District Court stated that there must be a "causal connection" between the exceptional conduct and the amount of attorneys' fees awarded, the District Court did just the opposite—it awarded attorneys' fees that Amazon would have incurred, regardless of any exceptional conduct. *See* Appx0042.

Before turning to specific categories of erroneously awarded fees/costs, and issues of claim preclusion/*Kessler* bar to the side, the law is clear that the exceptional conduct found by the District Court may only result in fee shifting with a cause-and-effect relationship. *Goodyear Tire & Rubber Co. v. Haeger*, 137 S. Ct. 1178, 1187 (2017). With the MDL having been ordered over Amazon's objection (and never at any risk of "unraveling") and with the First Order replete with multiple findings of reasonable infringement positions being advanced by PersonalWeb through the disposition on the second Motion for Summary Judgment, with the Twitch facts surrounding category 2 becoming known in July 2019, and with the Bermeister and Hadley declarations having been disregarded by the Court in its rulings on the first Motion for Summary Judgment—the District Court rulings are only punitive and in no way compensatory.

A. Erroneous Fees for Non-Exceptional Case Management Conduct

Amazon sought $1,079,001.52 in attorneys' fees related to case management, and the District Court awarded $809,251.14, amounting to a 25% reduction.

Appx0046-47. The District Court stated that "PersonalWeb's frivolous conduct was a significant contributor to the need for case management." Appx0047. The District Court elaborated that "PersonalWeb's ever-shifting infringement theories—S3, Ruby on Rails, the late emergence of CloudFront and complete abandonment of Ruby on Rails, and finally, PersonalWeb's extraordinary declaration that Twitch was not a representative customer case only fifteen months after insisting just the opposition — support Amazon's need for significant case management efforts." *Id*. The District Court abused its discretion in awarding such fees.

First, the 25% reduction to $809,251 was completely arbitrary—it was not based on any determination that Amazon would have incurred $809,251 in case management fees but for any exceptional case management conduct. The District Court did not identify the basis of the 25% reduction, or attempt to explain how any exceptional case management conduct caused Amazon to incur $809,251.14 in fees.

Second, the District Court did not explain what it meant by "significant" or "frivolous conduct." Indeed, the District Court even acknowledged that "it is particularly difficult to attribute case management activities in this case…" *Id*. Any such difficulties in ascertaining the amount of case management fees tied to exceptional conduct must fall on Amazon, which had the burden of proof in establishing the amount of fees attributable to exceptional conduct. Yet, in abuse of its discretion, the District Court awarded $809,251 in attorneys' fees to Amazon,

even though Amazon did not satisfy its burden of proof.

Third, the District Court's decision to award fees for PersonalWeb's claims related to the use of S3 is contradicted by the District Court's determinations that (1) "PersonalWeb's infringement positions were not objectively baseless and thus, do not support a finding of an exceptional case"; and (2) "the baselessness of PersonalWeb's claims related to the use of S3 plays a limited role in the Court's 'totality of circumstances' analysis for finding this case exceptional." Appx0015, Appx0013. Based on the District Court's determination that PersonalWeb's S3 claims played only a "limited role", the District Court, by its very own standards, abused its discretion in awarding Amazon 75% of its requested attorneys' fees for this category.

Fourth, the District Court's decision to award fees for "the late emergence of CloudFront" was erroneous. The District Court rejected Amazon's Motion for Judgment on that very issue, and "easily resolved this motion in favor of PersonalWeb." Appx0051. The District Court even noted that "the Court cannot impute misconduct to PersonalWeb based on a failed litigation decision by Amazon" (*id.*) — yet, in awarding attorneys' fees related to CloudFront, that is exactly what the District Court did. In fact, "the Court did not find PersonalWeb's CloudFront infringement claims were baseless." Appx0053.

Fifth, as addressed above, the District Court's statement that there was a

"complete abandonment of Ruby on Rails" is contradicted by the District Court's determination that PersonalWeb "emphasized" S3 prior to the MDL Order, and "put its focus on Ruby on Rails" following the MDL Order. Appx0016.

Sixth, the District Court had no basis to award any fees concerning Twitch. In late 2018, the District Court *rejected* "PersonalWeb's proposal to have multiple customer cases go forward", and instead decided to proceed with a single customer representative. Appx0047. Indeed, as set forth below, the District Court awarded fees related to the MDL on the basis that "had the Court accepted PersonalWeb's proposal to have multiple customer cases go forward, the MDL proceeding would have been significantly more expensive." *Id*. In other words, Twitch was the category 2 representative only because the District Court ordered a representative for all categories and rejected PersonalWeb's preference for having a second customer case. Having determined that there could be only one representative, the District Court cannot now penalize PersonalWeb for choosing a representative that all parties had agreed "seems to be the ideal exemplar" at the time. Appx0838, Appx0802. The fact that PersonalWeb subsequently discovered that Twitch was not an adequate category 2 representative based upon the July 2019 deposition testimony of Twitch's person most knowledgeable has no bearing on the reasonableness of PersonalWeb's belief as of September 2018 that Twitch was an adequate representative—especially considering that Amazon and Twitch's counsel agreed

and did nothing to disabuse either the District Court or PersonalWeb from that perception.

B. Erroneous Fees for Non-Exceptional MDL Conduct

Amazon sought, and the District Court awarded, $145,613.05 in fees for work related to the MDL. Appx0047. The District Court agreed with Amazon's position that Amazon was entitled to such fees because "it was Amazon's…presence in the MDL that made it possible for the Court to resolve the case as efficiently as it did. And had the Court accepted PersonalWeb's proposal to have multiple customer cases go forward, the MDL proceeding would have been significantly more expensive." *Id*. The District Court concluded that "Amazon would have not incurred these fees but for PersonalWeb's frivolous conduct…" Appx0048.

The District Court's determination was erroneous—the District Court did not provide any analysis or support. Further, the suggestion that Amazon's MDL position "made it possible for the Court to resolve the case as efficiently as it did" is false—Amazon *objected* to the MDL. For the reasons addressed above, the District Court had no basis to determine that PersonalWeb shifted its positions, or that PersonalWeb did anything "to unravel the MDL after summary judgment and to keep some of the customer cases in play." Appx0026. To the contrary, the District Court determined that "setting aside the claim construction 'clarification' issue, PersonalWeb endeavored to avoid prolonging this case." Appx0021.

C. Erroneous Fees for Investigating and Responding to PersonalWeb's Claims

Amazon sought $212,047.88 in attorneys' fees related to investigating and responding to PersonalWeb's claims, and the District Court awarded $159,035.91 (amounting to a 25% reduction). Appx0048. In justifying said amount, the District Court stated that "PersonalWeb's abandoned infringement theories serially dominated the case, leading to unnecessary work across multiple categories of litigation." *Id*.

The reduction of 25% was entirely arbitrary and not based on any causal connection with exceptional conduct. Moreover, the award is nullified by the District Court's findings that "PersonalWeb's infringement positions were not objectively baseless and thus, do not support a finding of an exceptional case" and that "while unsuccessful, PersonalWeb presented reasonable infringement arguments throughout the case, including at summary judgment." Appx0015, Appx0014.

D. Erroneous Fees for Non-Exceptional Motions to Stay

Amazon sought, and the District Court awarded, $151,668.62 in attorneys' fees for work related to Amazon's motions to stay. Appx0050. The District Court found that "PersonalWeb's S3-related misconduct led Amazon to file its declaratory judgment suit, the motions to stay, and the motion for a preliminary injunction." *Id*. That determination was erroneous, as PersonalWeb's S3-related claims were only a small portion of the case, and (as addressed above) the District Court determined

that the remainder of PersonalWeb's claims were reasonable. Indeed, the case proceeded after the District Court granted the first Motion for Summary Judgment attacking PersonalWeb's S3 claims. This necessarily establishes that such fees would have been incurred even if PersonalWeb did not pursue any S3-related claims.

E. Erroneous Fees for Non-Exceptional Injunction Motion

Amazon sought, and the District Court awarded, $219,985.69, in attorneys' fees for work related to its Injunction Motion. Appx0051. The District Court relied on the same reasons that it awarded fees related to the motions to stay, as addressed above. Yet, as set forth above, but-for PersonalWeb's S3 infringement claims, the Injunction Motion still would have been filed, and would have been based on positions that the District Court determined were reasonable. Thus, Amazon would have necessarily incurred such fees but for any exceptional conduct.

F. Erroneous Fees for Non-Exceptional Infringement Contentions

Amazon sought, and the District Court awarded, $72,896.40 in attorneys' fees for work related to PersonalWeb's infringement contentions. Appx0051. The District Court determined that "the case schedule required Amazon to review and respond to PersonalWeb's contentions…Buttressing this conclusion is the fact that in June 2019 PersonalWeb moved to amend its infringement contentions in bad faith. *See* Order Awarding Fees at 24-26." Appx0052. In June 2019, no such finding was made, not by the District Court and not by the Magistrate Judge who ruled on

PersonalWeb's motion to amend. That the District Court objected in February 2020 to PersonalWeb asking whether the Magistrate Judge's August 7, 2019 order automatically applied to the stayed cases says nothing about attorneys' fees incurred in 2019.

G. Erroneous Fees for Non-Exceptional Claim Construction Conduct

Amazon requested $520,714.93 in attorneys' fees for work related to claim construction, and the District Court awarded Amazon $390,536.20 (amounting to a 25% reduction). Appx0053-54. Despite the District Court's conclusion that "PersonalWeb made reasonable, albeit unsuccessful, arguments that the terms authorized/unauthorized or licensed/unlicensed should be given their plain and ordinary meaning" (Appx0014), the District Court awarded such fees on the basis that "PersonalWeb's attempt to work around the Court's claim construction ruling was egregious and exceptional…" Appx0053-54.

There was no basis to award such fees. The use of the 25% reduction was arbitrary, and was not tied to any exceptional conduct. In fact, in analyzing this category of fees, the District Court even acknowledged that "the Court cannot trace all of the requested fees to PersonalWeb's misconduct." Appx0054. As addressed above, this can only mean that Amazon did not meet its burden in establishing its entitlement to such attorneys' fees.

Further, PersonalWeb's post claim construction conduct could not have

supported any award of fees, as the District Court determined that the Motion to Clarify was "generally proper" and contained "reasonable, albeit unsuccessful, arguments…" Appx0021, Appx14. Finally, even assuming, *arguendo*, that this conduct should have been deemed exceptional—it should not have been—the District Court determined that it became exceptional only *after* its Construction Order. Yet, it applied only a blanket 25% reduction, even though the attorneys' fees were necessarily incurred *before* the Construction Order.

CONCLUSION

In light of the foregoing, the Orders should be reversed, in their entirety.

August 27, 2021

/s/ *Michael A. Sherman*

Michael A. Sherman
Jeffrey F. Gersh
Wesley W. Monroe

STUBBS, ALDERTON & MARKILES, LLP
15260 Ventura Blvd. Floor 20
Sherman Oaks, CA 91403
Telephone: (818) 444-4528
masherman@stubbsalderton.com
jgersh@stubbsalderton.com
wmonroe@stubbsalderton.com

ADDENDUM

US005978791A

United States Patent [19]

Farber et al.

[11] Patent Number: 5,978,791

[45] Date of Patent: Nov. 2, 1999

[54] **DATA PROCESSING SYSTEM USING SUBSTANTIALLY UNIQUE IDENTIFIERS TO IDENTIFY DATA ITEMS, WHEREBY IDENTICAL DATA ITEMS HAVE THE SAME IDENTIFIERS**

[75] Inventors: **David A. Farber**, Ojai, Calif.; **Ronald D. Lachman**, Northbrook, Ill.

[73] Assignee: **Kinetech, Inc.**, Northbrook, Ill.

[21] Appl. No.: **08/960,079**

[22] Filed: **Oct. 24, 1997**

Related U.S. Application Data

[63] Continuation of application No. 08/425,160, Apr. 11, 1995, abandoned.

[51] **Int. Cl.**[6] **G06F 17/30**

[52] **U.S. Cl.** **707/2**; 707/1; 707/200

[58] **Field of Search** 707/2, 1, 200

[56] **References Cited**

U.S. PATENT DOCUMENTS

3,668,647	6/1972	Evangelisti et al.	340/172.5
4,215,402	7/1980	Mitchell et al.	364/200
4,290,105	9/1981	Cichelli et al.	364/200
4,376,299	3/1983	Rivest	364/900
4,405,829	9/1983	Rivest et al.	178/22.1
4,412,285	10/1983	Neches et al.	364/200
4,414,624	11/1983	Summer, Jr. et al.	364/200
4,441,155	4/1984	Fletcher et al.	364/200
4,464,713	8/1984	Benhase et al.	364/200
4,490,782	12/1984	Dixon et al.	364/200
4,571,700	2/1986	Emry, Jr. et al.	364/900
4,577,293	3/1986	Matick et al.	365/189
4,642,793	2/1987	Meaden	364/900
4,675,810	6/1987	Gruner et al.	364/200
4,691,299	9/1987	Rivest et al.	365/185
4,725,945	2/1988	Kronstadt et al.	364/200
4,773,039	9/1988	Zamora	364/900
4,887,235	12/1989	Holloway et al.	364/900
4,888,681	12/1989	Barnes et al.	364/200
4,922,414	5/1990	Holloway et al.	364/200
4,972,367	11/1990	Burke	364/900
5,007,658	4/1991	Bendert et al.	395/600
5,025,421	6/1991	Cho	365/230.05
5,050,074	9/1991	Marca	364/200
5,050,212	9/1991	Dyson	380/25
5,057,837	10/1991	Colwell et al.	341/55
5,129,081	7/1992	Kobayashi et al.	395/600
5,129,082	7/1992	Tirfing et al.	395/600
5,144,667	9/1992	Pogue, Jr. et al.	380/45
5,179,680	1/1993	Colwell et al.	395/425
5,202,982	4/1993	Gramlich et al.	395/600
5,208,858	5/1993	Vollert et al.	380/43
5,276,901	1/1994	Howell et al.	395/800
5,301,286	4/1994	Rajani	395/400
5,301,316	4/1994	Hamilton et al.	395/600
5,343,527	8/1994	Moore	380/4
5,357,623	10/1994	Megory-Cohen	395/425
5,384,565	1/1995	Cannon	340/825.44
5,404,508	4/1995	Konrad et al.	395/600

OTHER PUBLICATIONS

Witold Litwin et al, Linear Hashing for Distributed Files, ACM SIGMOD, May, 1993 pp. 327–336.

Ming–Ling Lo, et al, On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69–78, May 1993.

Thomas A. Berson, Differential Cryptanalysis Mod 2^{32} with Applications to MD5, pp. 69–81, 1992.

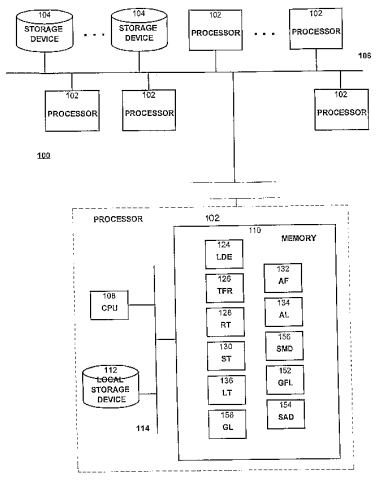
(List continued on next page.)

Primary Examiner—Paul V. Kulik
Assistant Examiner—Jean R. Homere
Attorney, Agent, or Firm—Pillsbury Madison & Sutro LLP

[57] **ABSTRACT**

In a data processing system, a mechanism identifies data items by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items. The system also determines whether a particular data item is present in the database by examining the identifiers of the plurality of data items.

48 Claims, 31 Drawing Sheets



OTHER PUBLICATIONS

William Perrizo, et al., Distributed Join Processing Performance Evaluation, 1994. Twenty–Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236–244.

A concurrency Control Mechanism based on Extendible Hashing for Main Memory Database Systems, Vijay Kumar, pp. 109–113, ACM, vol. 3, 1989.

Birgit Pfitzmann, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93 pp. 74–85.

Bert dem Boer, et al., Collisions for the compression function of MD_5 pp. 292–304, 1994.

Sakti Pramanik, et al., Multi–Directory Hashing, 1993, Info. Sys., vol. 18, No. 1, pp. 63–74.

Murlidhar Koushik, Dynamic Hashing With Distributed Overflow Space: A File Organization With Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299–317.

Witold Litwin, et al., LH*–Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL–93–21 Jun. 1993 pp. 1–22.

Yuliang Zheng, et al., HAVAL — A One–Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83–105, Advances in Cryptology, AUSCRIPT '92, 1992.

Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, pp. 156–164, 1993.

Zhiyu Tian, et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3–13, SIGIR Forum, 1993.

G. L. Friedman, Digital Camera With Apparatus For Authentication of Images Produced From an Image File, NASA Case No. NPO–19108–1–CU, Serial No. 08/159,980, Nov. 24, 1993.

H. Goodman, Feb. 9, 1994 Ada, Object–Oriented Techniques, and Concurrency in Teaching Data Sructures and File Management Report Documentation P. AD–A275 385 — 94–04277.

Advances in Cryptology–EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings.

Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.

Advances in Cryptology–AUSCRYPT '92 — Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia Dec. 13–16, 1992 Proceedings.

Search Report dated Jun. 24, 1996.

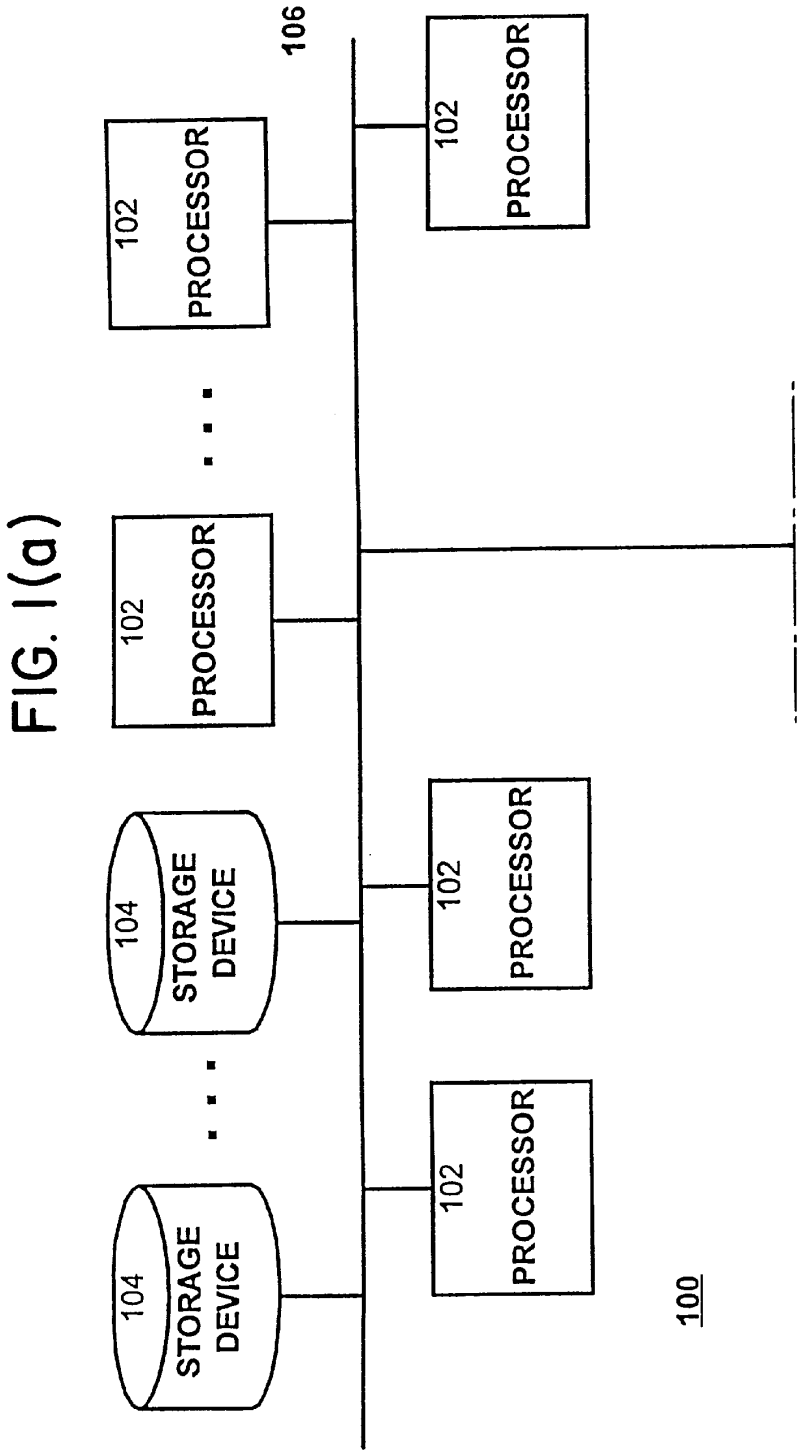

106
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
104
STORAGE DEVICE
104
STORAGE DEVICE
100

FIG. 1(a)

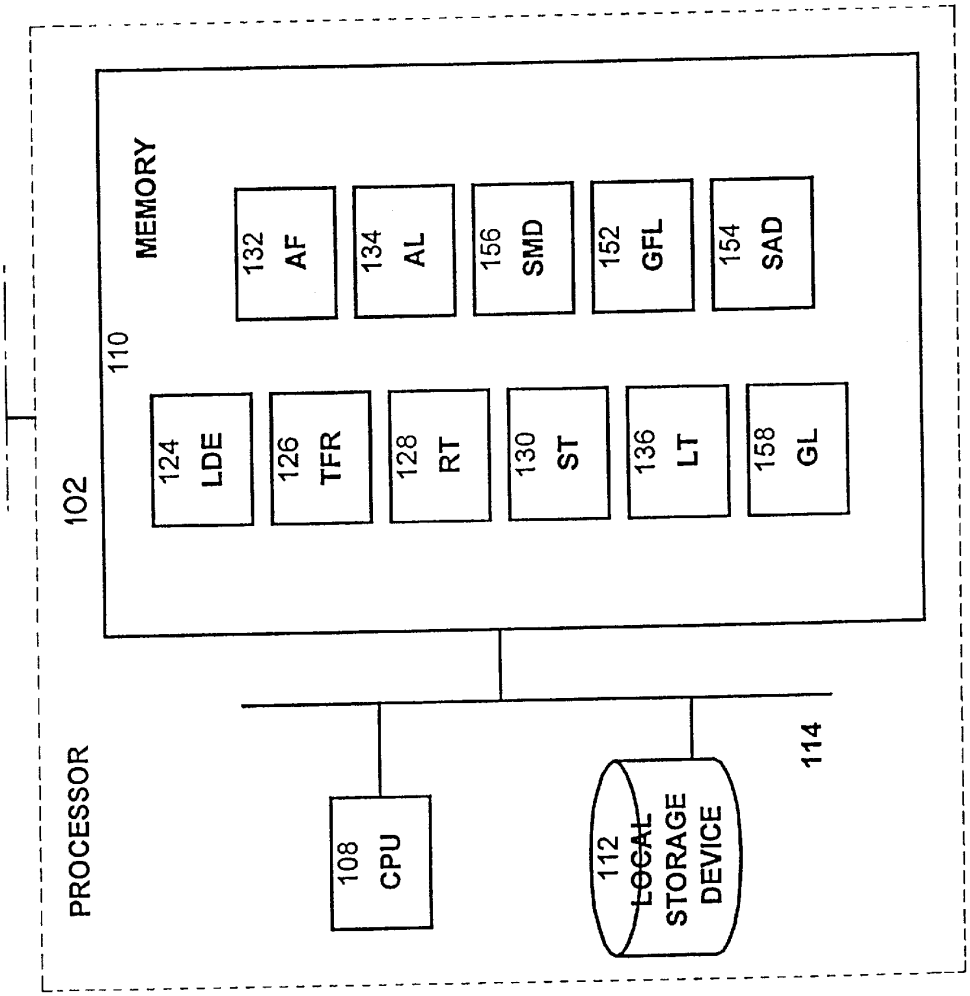

PROCESSOR
102
MEMORY
110
124 LDE
126 TFR
128 RT
130 ST
136 LT
158 GL
132 AF
134 AL
156 SMD
152 GFL
154 SAD
108 CPU
112 LOCAL STORAGE DEVICE
114

FIG. 1(b)

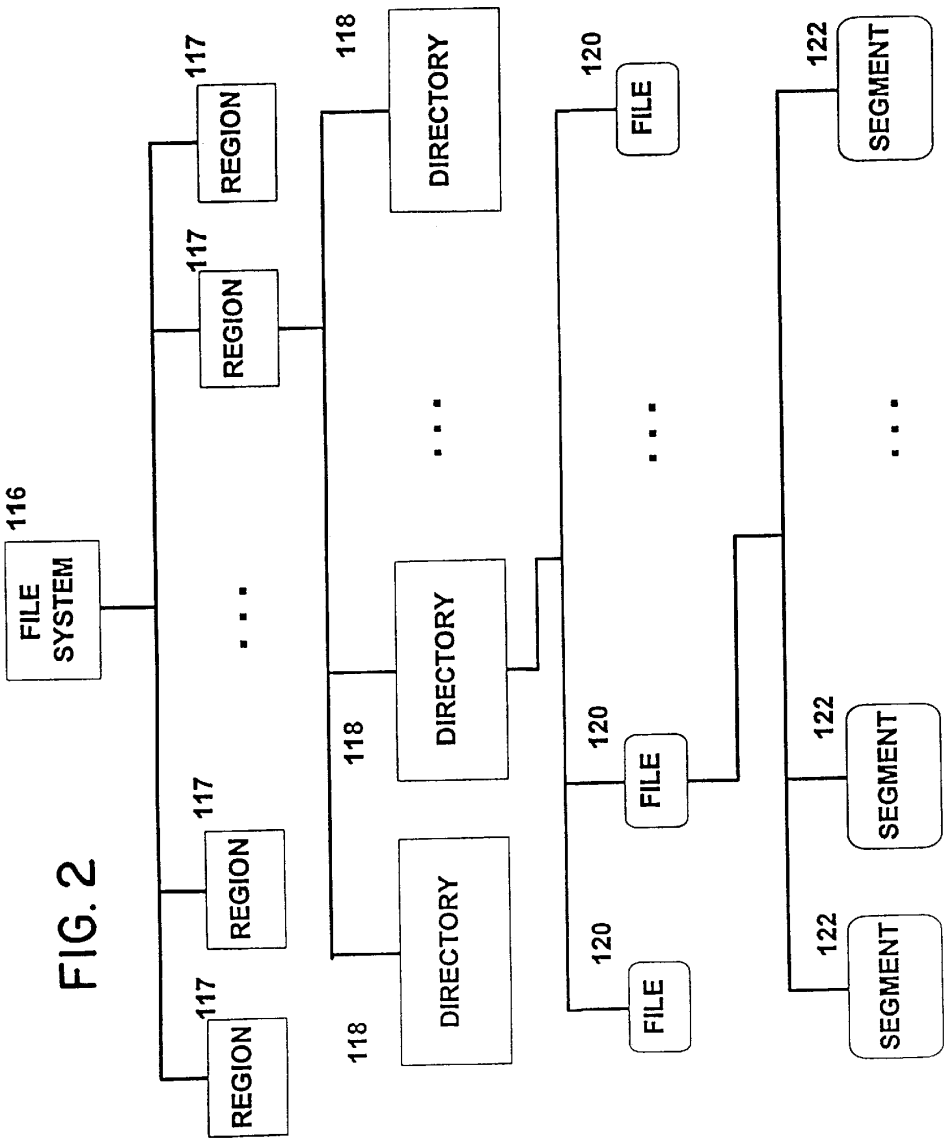

116
FILE SYSTEM
117
REGION
117
REGION
117
REGION
118
DIRECTORY
118
DIRECTORY
118
DIRECTORY
120
FILE
120
FILE
120
FILE
122
SEGMENT
122
SEGMENT
122
SEGMENT

FIG. 2

FIG. 3

138

Region ID
Pathname
True Name
Type
File ID
Time of last access
Time of last modification
Safe flag
Lock flag
Size
Owner

FIG. 4

140

True Name
File ID
Compressed File ID
Source IDs
Dependent processors
Use count
Time of last access
Expiration
Grooming delete count

142

Region ID
Region file system
Region pathname
Region status
Mirror processor(s)
Mirror duplication count
Policy

FIG. 5

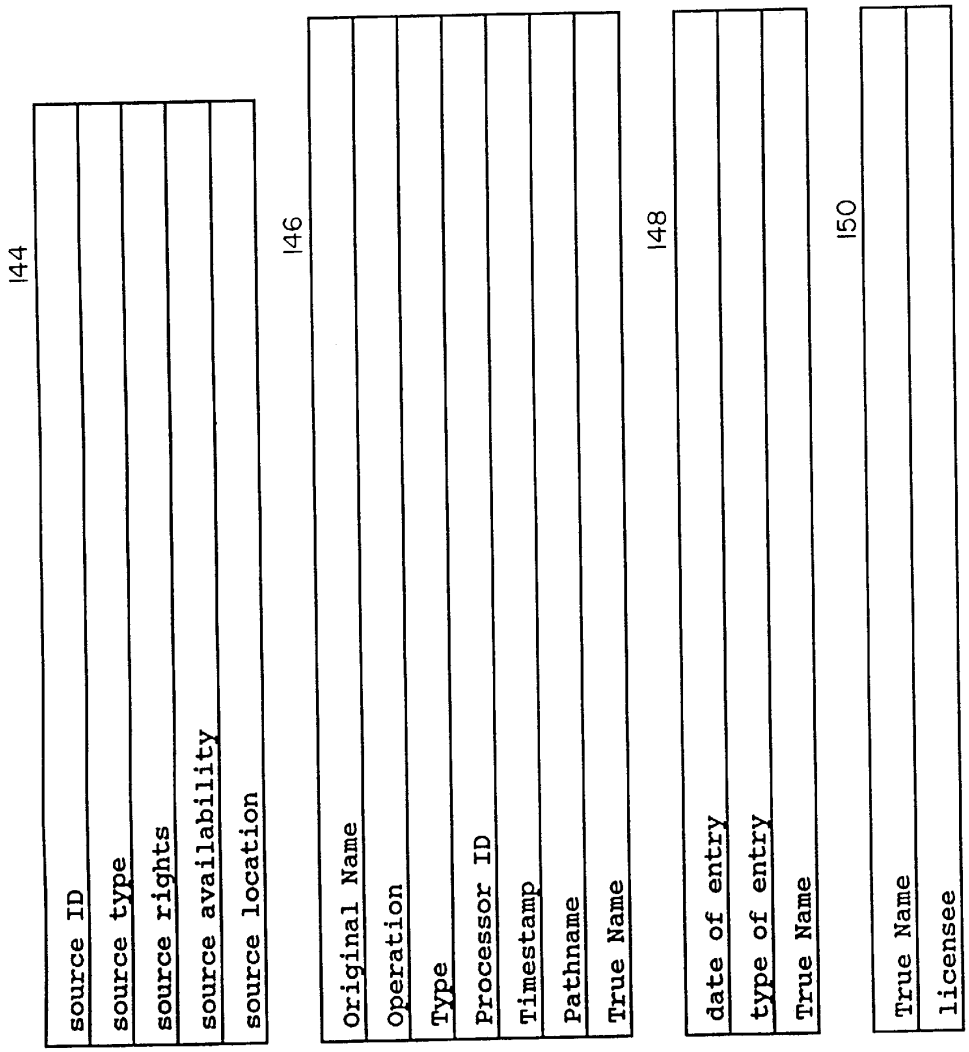


FIG. 6

FIG. 7

FIG. 8

FIG. 9

FIG. 10(a)

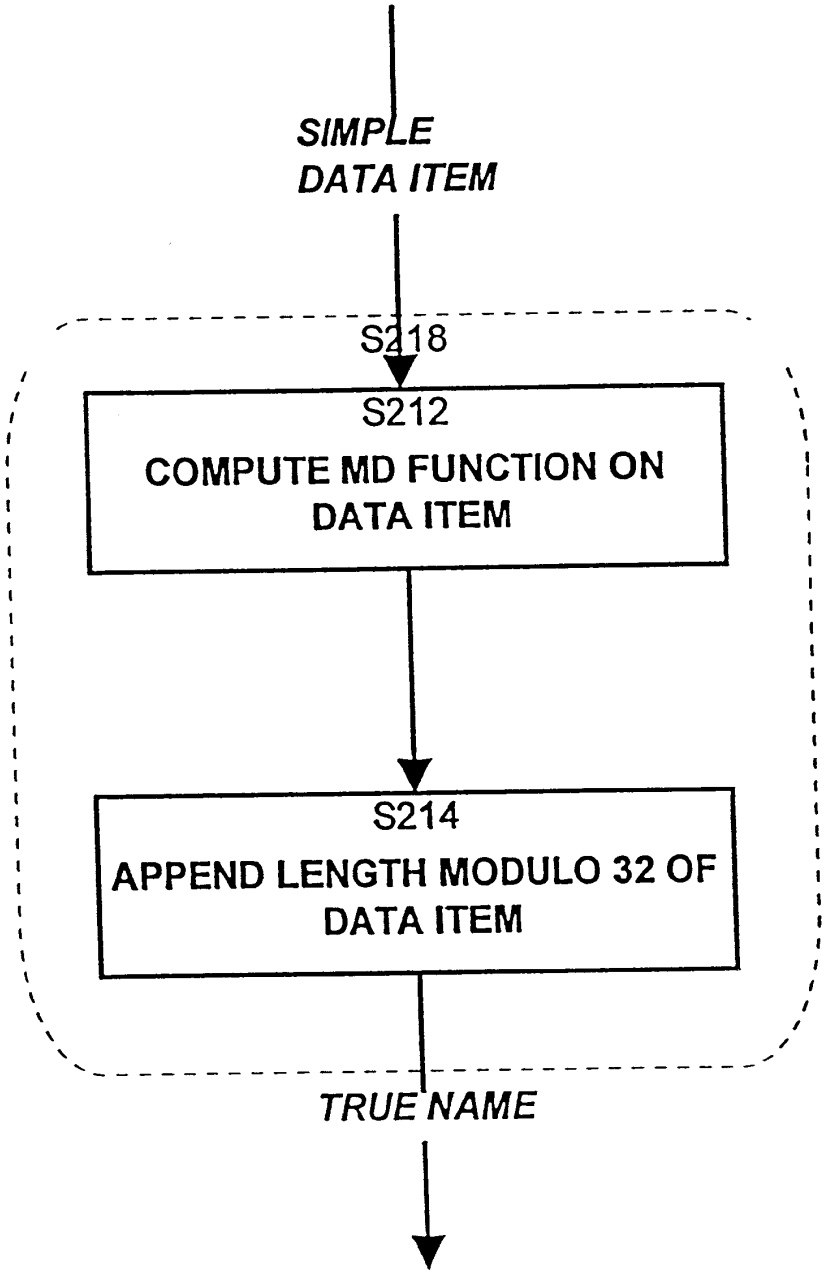
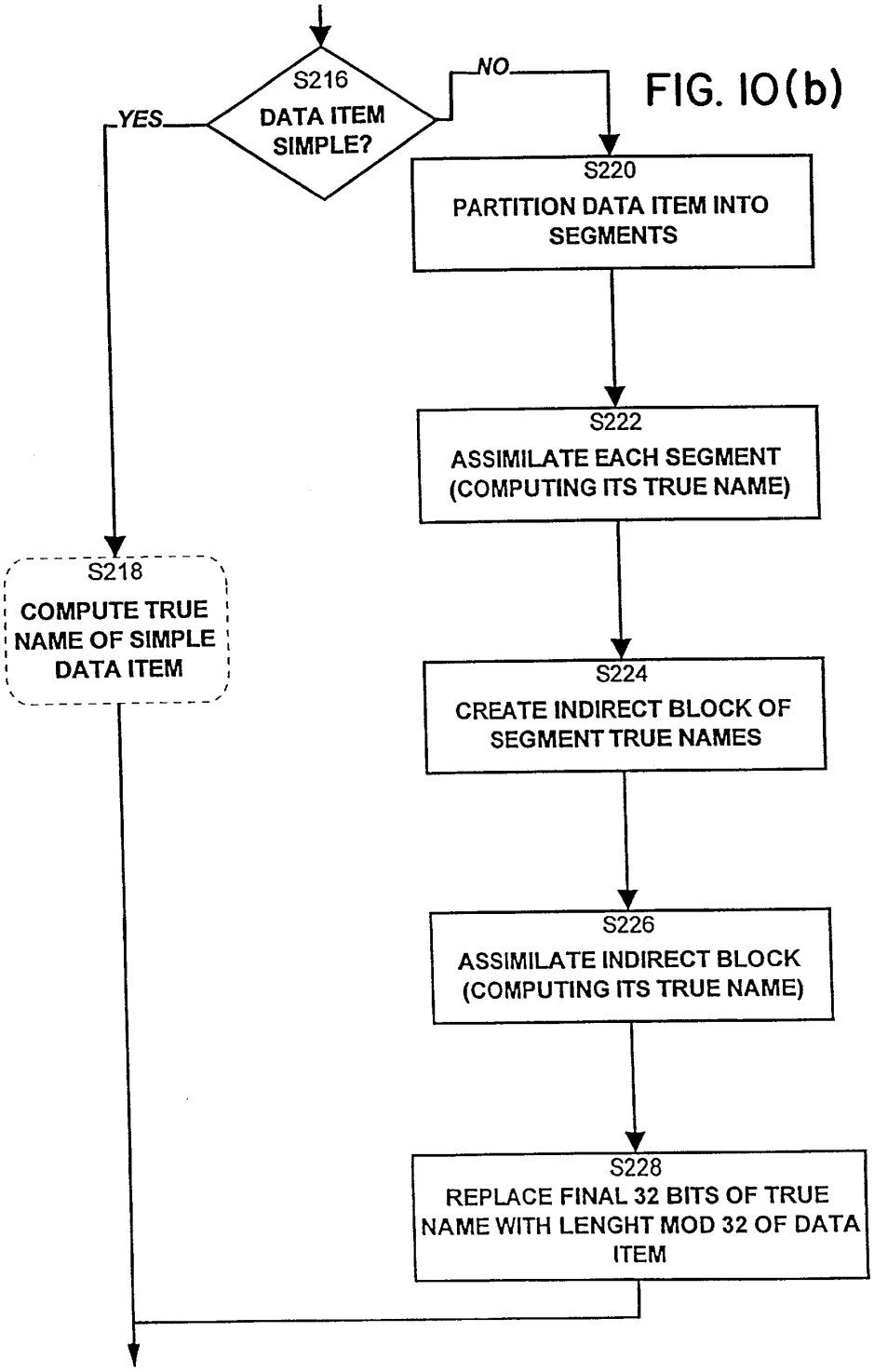

SIMPLE
DATA ITEM
S218
S212
COMPUTE MD FUNCTION ON
DATA ITEM
S214
APPEND LENGTH MODULO 32 OF
DATA ITEM
TRUE NAME

FIG. 10(b)


S216
DATA ITEM SIMPLE?
YES
NO
S218
COMPUTE TRUE NAME OF SIMPLE DATA ITEM
S220
PARTITION DATA ITEM INTO SEGMENTS
S222
ASSIMILATE EACH SEGMENT (COMPUTING ITS TRUE NAME)
S224
CREATE INDIRECT BLOCK OF SEGMENT TRUE NAMES
S226
ASSIMILATE INDIRECT BLOCK (COMPUTING ITS TRUE NAME)
S228
REPLACE FINAL 32 BITS OF TRUE NAME WITH LENGHT MOD 32 OF DATA ITEM

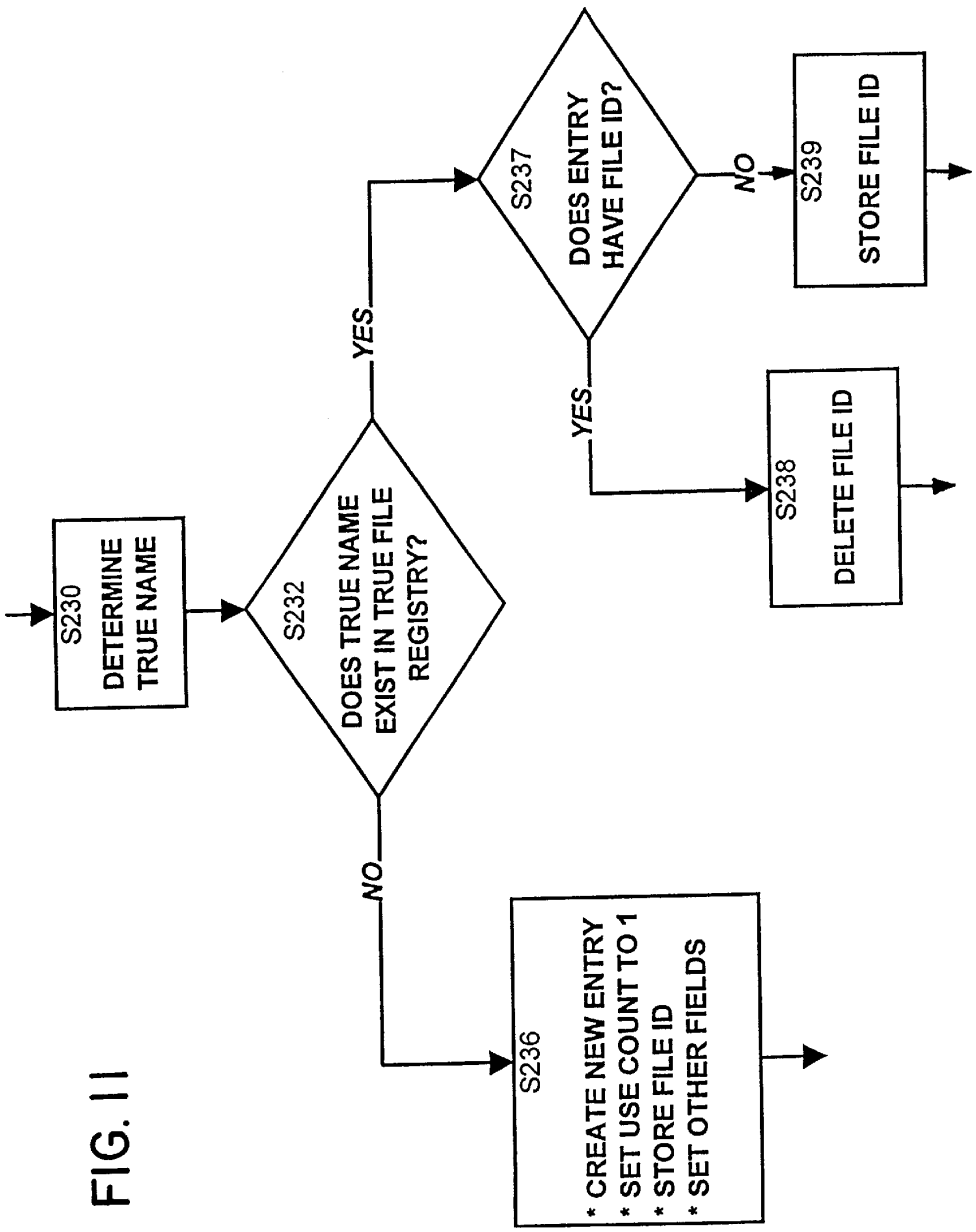

S230
DETERMINE TRUE NAME
S232
DOES TRUE NAME EXIST IN TRUE FILE REGISTRY?
YES
NO
S237
DOES ENTRY HAVE FILE ID?
YES
NO
S238
DELETE FILE ID
S239
STORE FILE ID
S236
* CREATE NEW ENTRY
* SET USE COUNT TO 1
* STORE FILE ID
* SET OTHER FIELDS

FIG. 11

FIG. 12

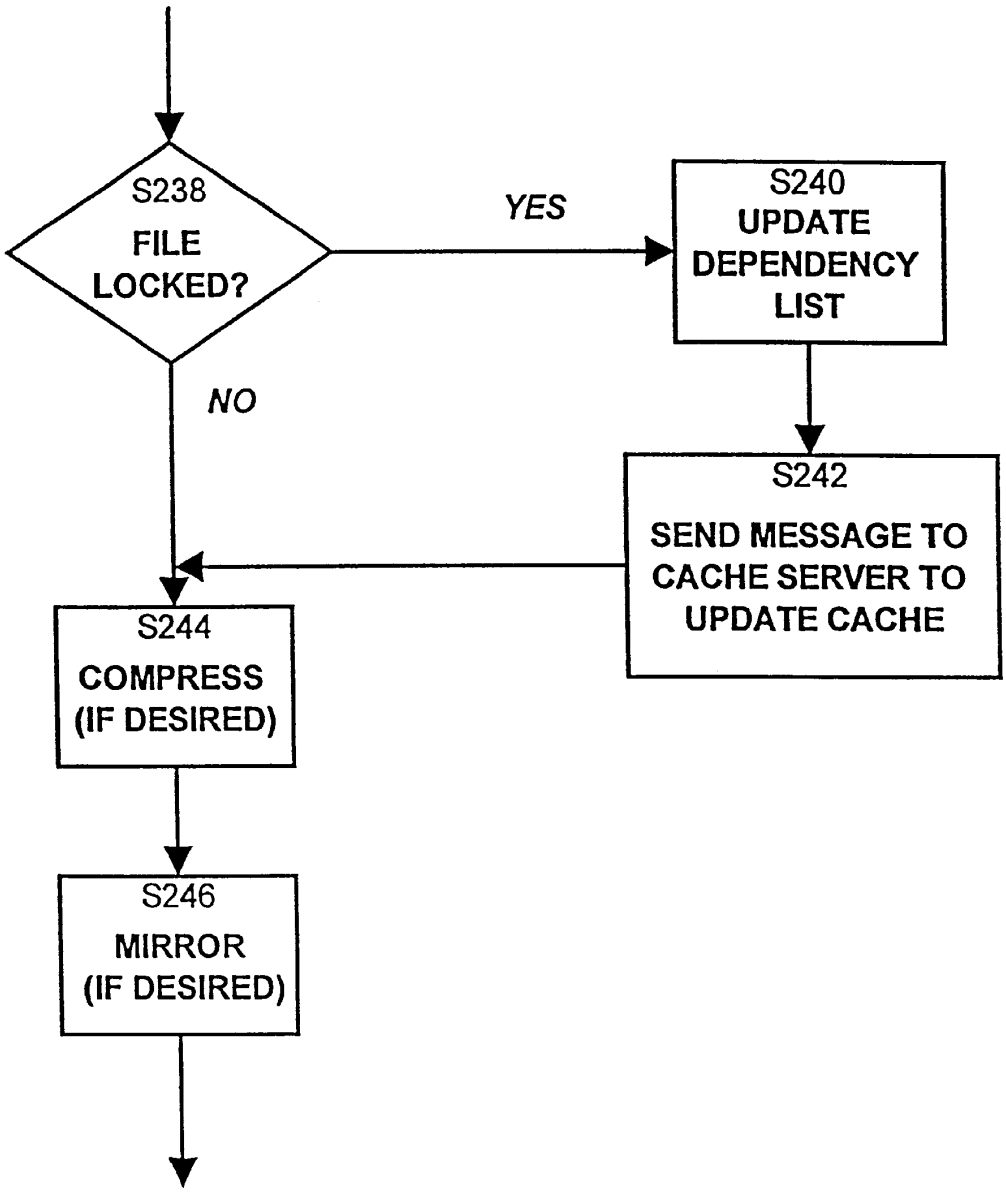
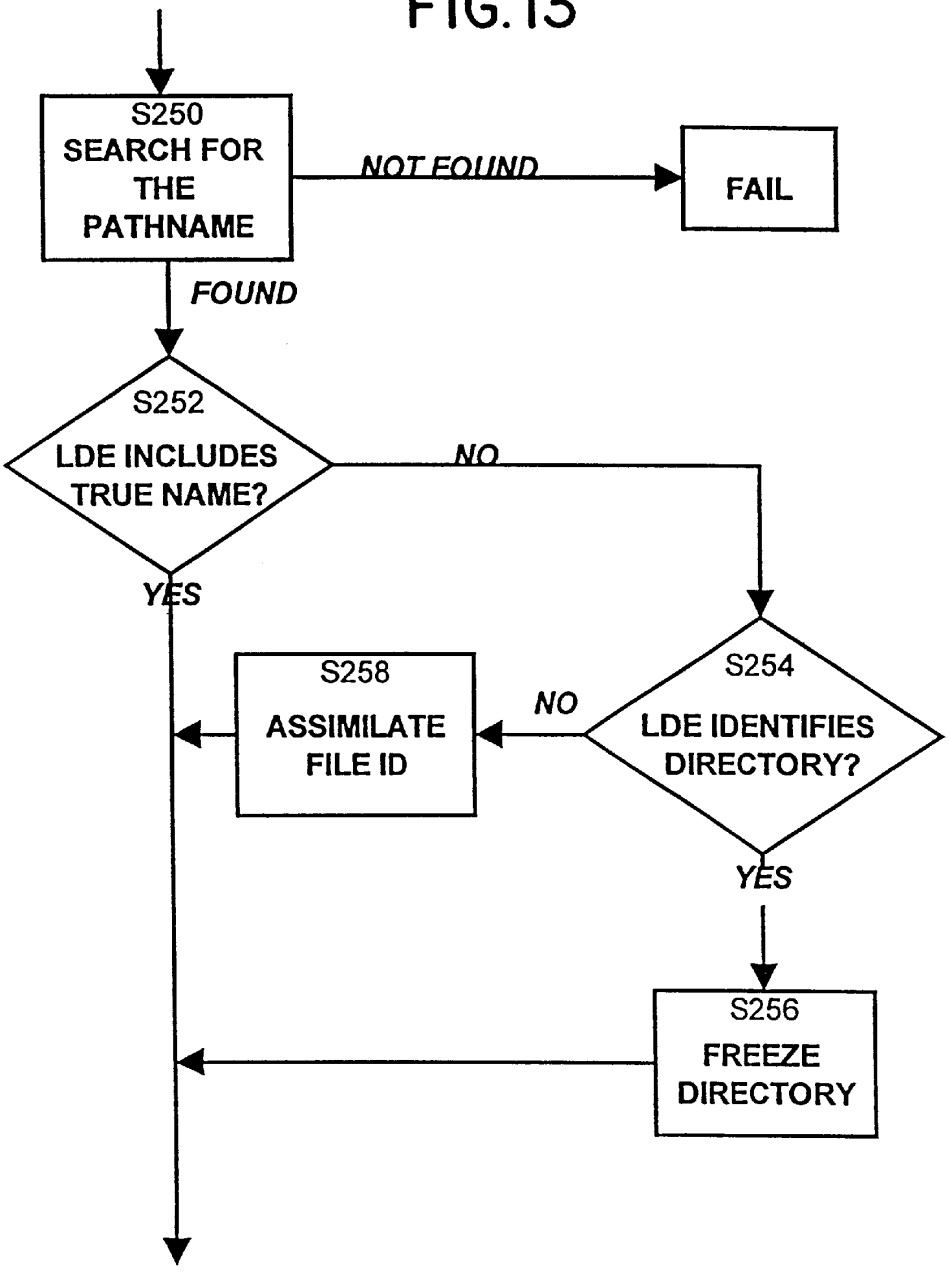

S238
FILE LOCKED?
YES
S240
UPDATE DEPENDENCY LIST
NO
S242
SEND MESSAGE TO CACHE SERVER TO UPDATE CACHE
S244
COMPRESS (IF DESIRED)
S246
MIRROR (IF DESIRED)

FIG. 13


S250
SEARCH FOR THE PATHNAME
NOT FOUND
FAIL
FOUND
S252
LDE INCLUDES TRUE NAME?
NO
YES
S254
LDE IDENTIFIES DIRECTORY?
NO
S258
ASSIMILATE FILE ID
YES
S256
FREEZE DIRECTORY

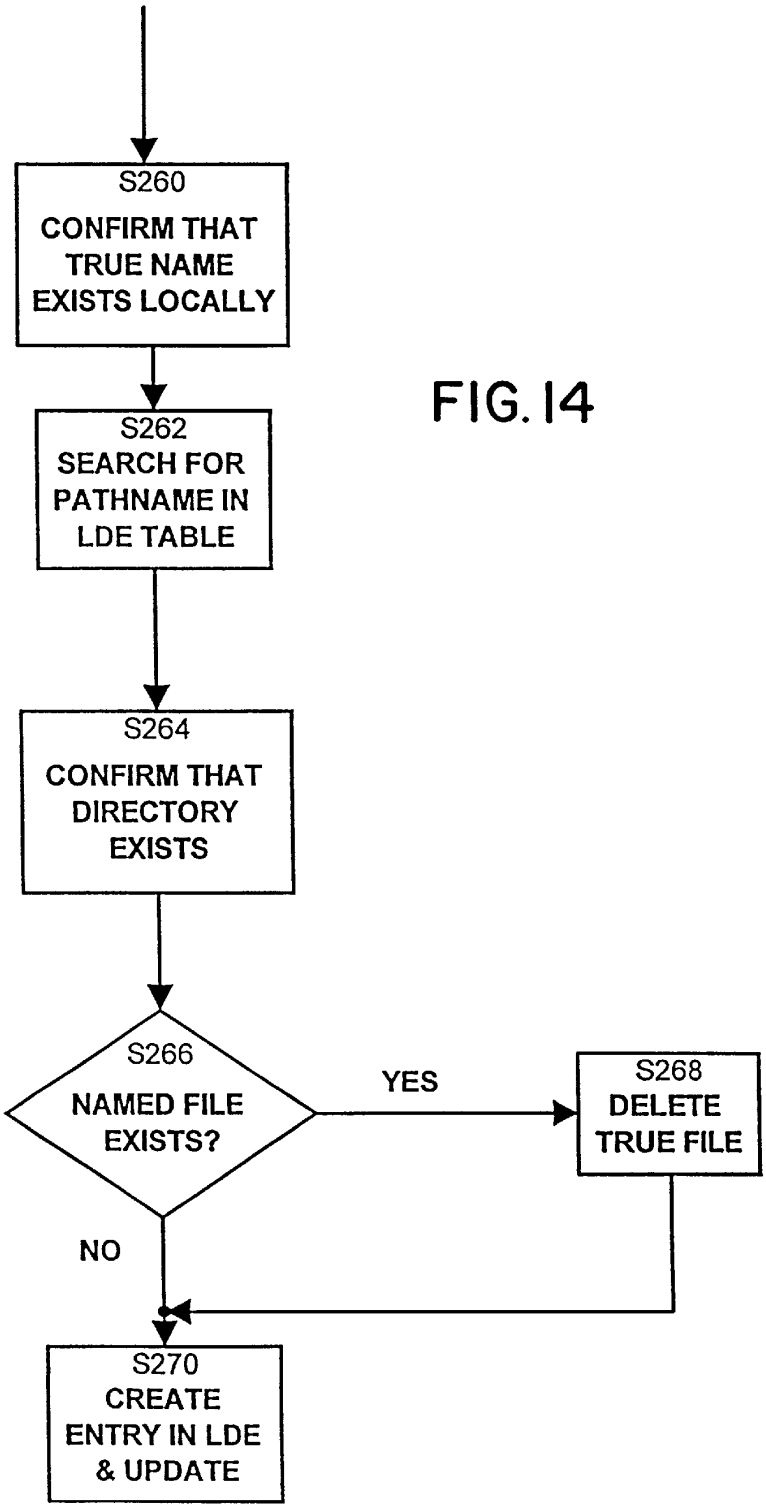

S260
CONFIRM THAT TRUE NAME EXISTS LOCALLY
S262
SEARCH FOR PATHNAME IN LDE TABLE
S264
CONFIRM THAT DIRECTORY EXISTS
S266
NAMED FILE EXISTS?
YES
S268
DELETE TRUE FILE
NO
S270
CREATE ENTRY IN LDE & UPDATE

FIG. 14

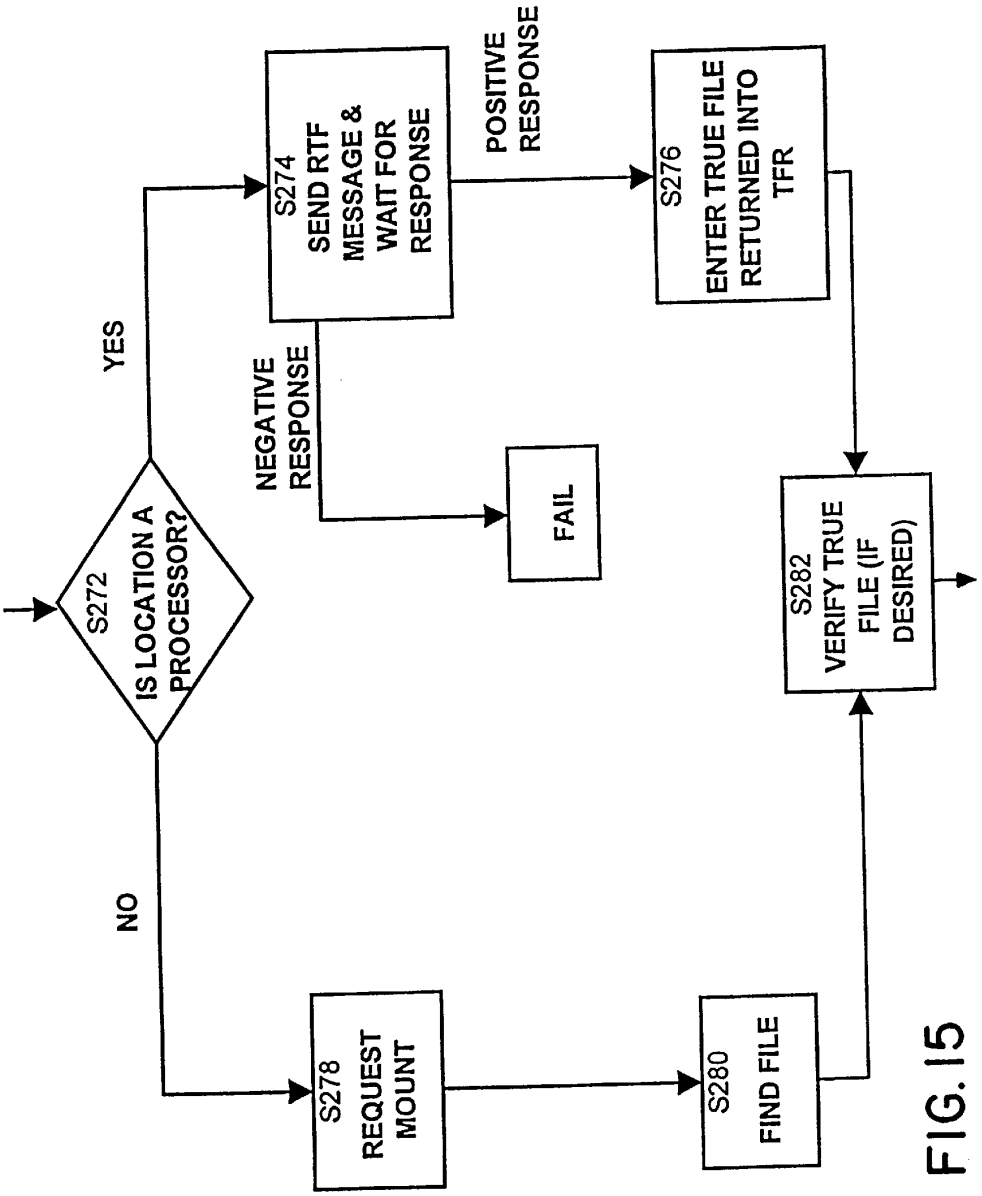

S272
IS LOCATION A PROCESSOR?
YES
NO
S274
SEND RTF MESSAGE & WAIT FOR RESPONSE
POSITIVE RESPONSE
NEGATIVE RESPONSE
FAIL
S276
ENTER TRUE FILE RETURNED INTO TFR
S278
REQUEST MOUNT
S280
FIND FILE
S282
VERIFY TRUE FILE (IF DESIRED)

FIG. 15

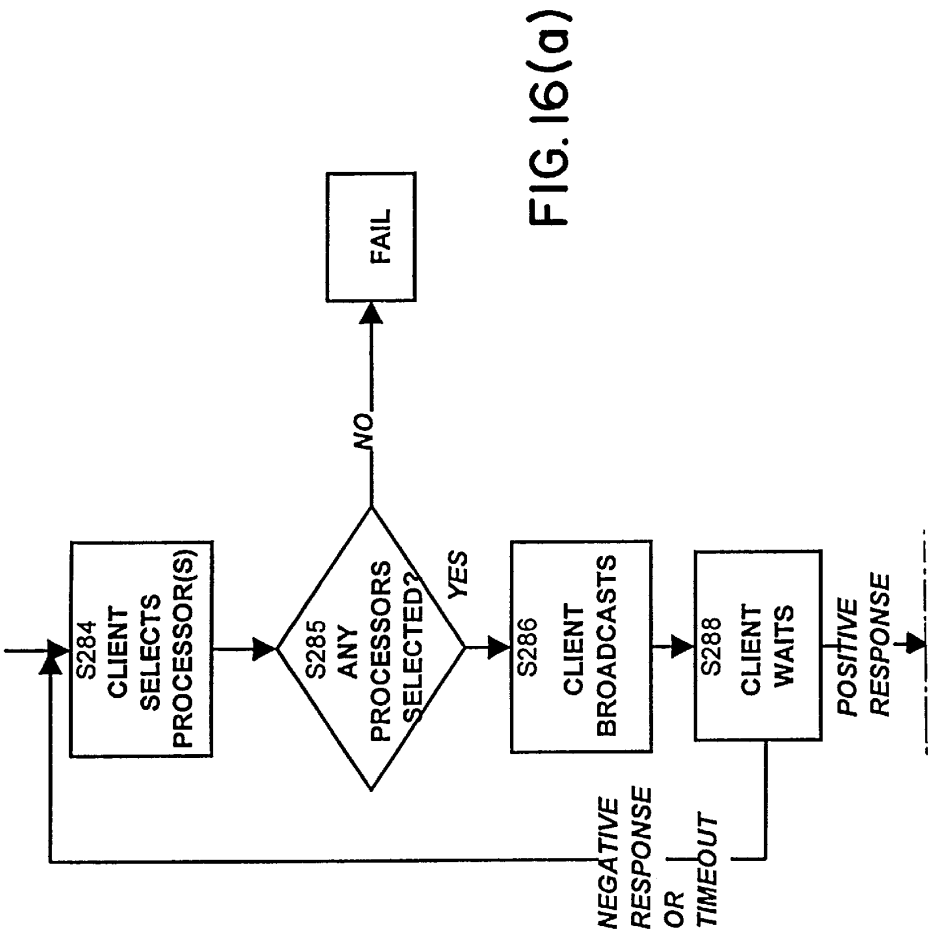

S284
CLIENT
SELECTS
PROCESSOR(S)
S285
ANY
PROCESSORS
SELECTED?
NO
FAIL
YES
S286
CLIENT
BROADCASTS
S288
CLIENT
WAITS
POSITIVE
RESPONSE
NEGATIVE
RESPONSE
OR
TIMEOUT

FIG. 16(a)

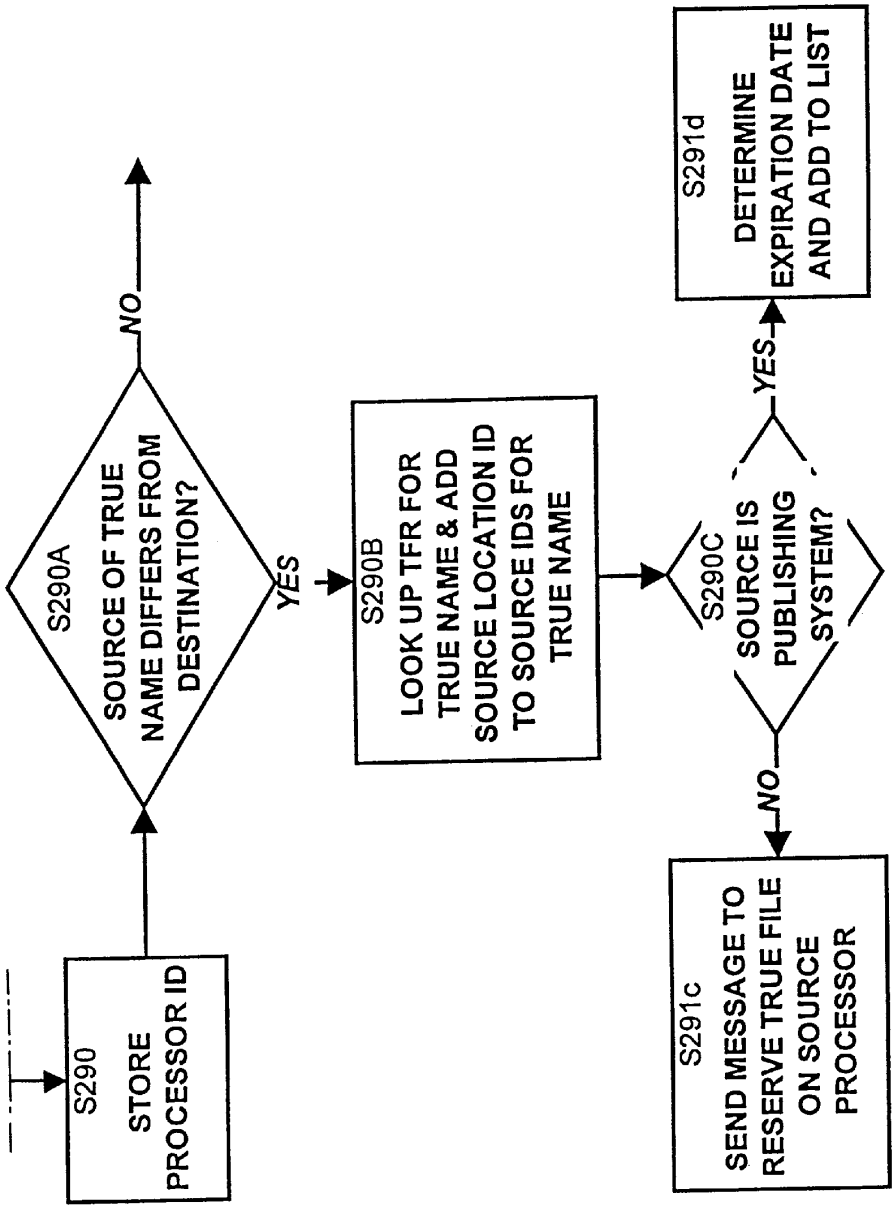

S290
STORE PROCESSOR ID
S290A
SOURCE OF TRUE NAME DIFFERS FROM DESTINATION?
NO
YES
S290B
LOOK UP TFR FOR TRUE NAME & ADD SOURCE LOCATION ID TO SOURCE IDS FOR TRUE NAME
S290C
SOURCE IS PUBLISHING SYSTEM?
NO
YES
S291c
SEND MESSAGE TO RESERVE TRUE FILE ON SOURCE PROCESSOR
S291d
DETERMINE EXPIRATION DATE AND ADD TO LIST

FIG. 16(b)

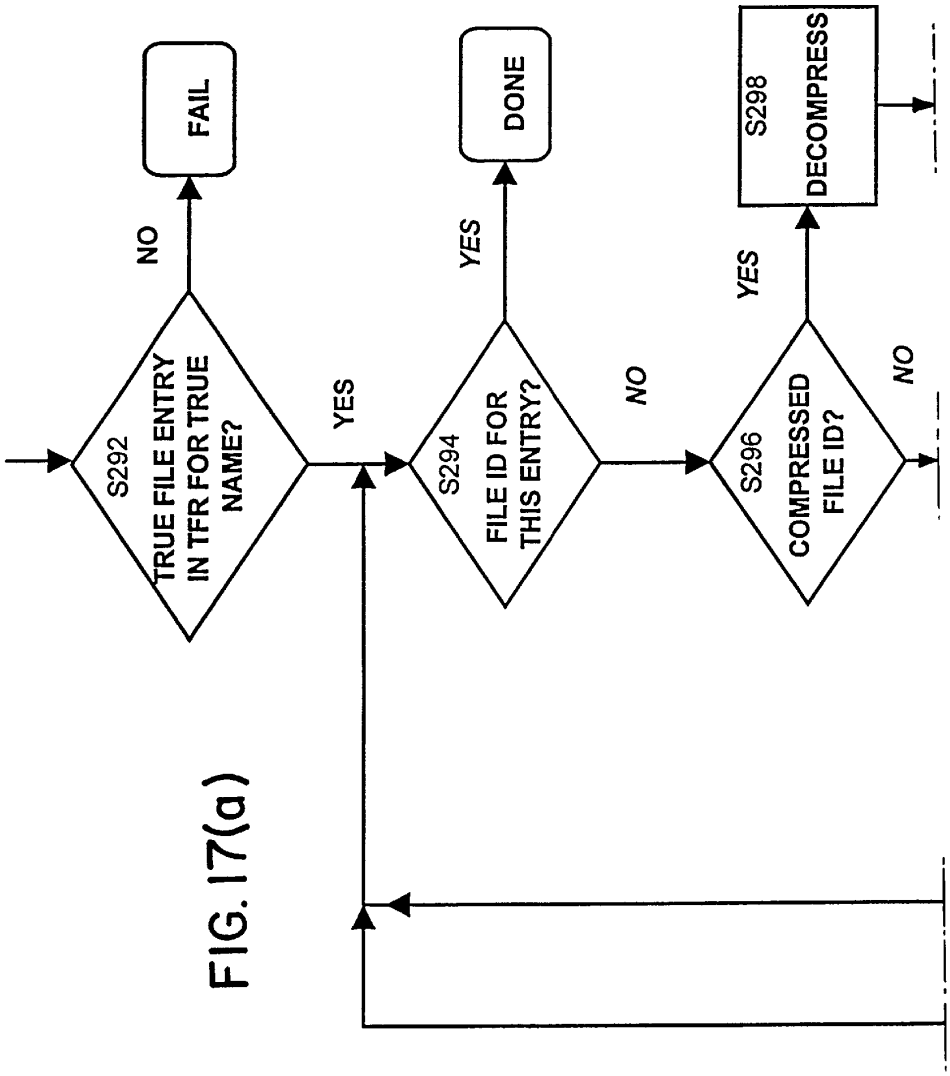

S292
TRUE FILE ENTRY IN TFR FOR TRUE NAME?
NO
FAIL
YES
S294
FILE ID FOR THIS ENTRY?
YES
DONE
NO
S296
COMPRESSED FILE ID?
YES
S298
DECOMPRESS
NO

FIG. 17(a)

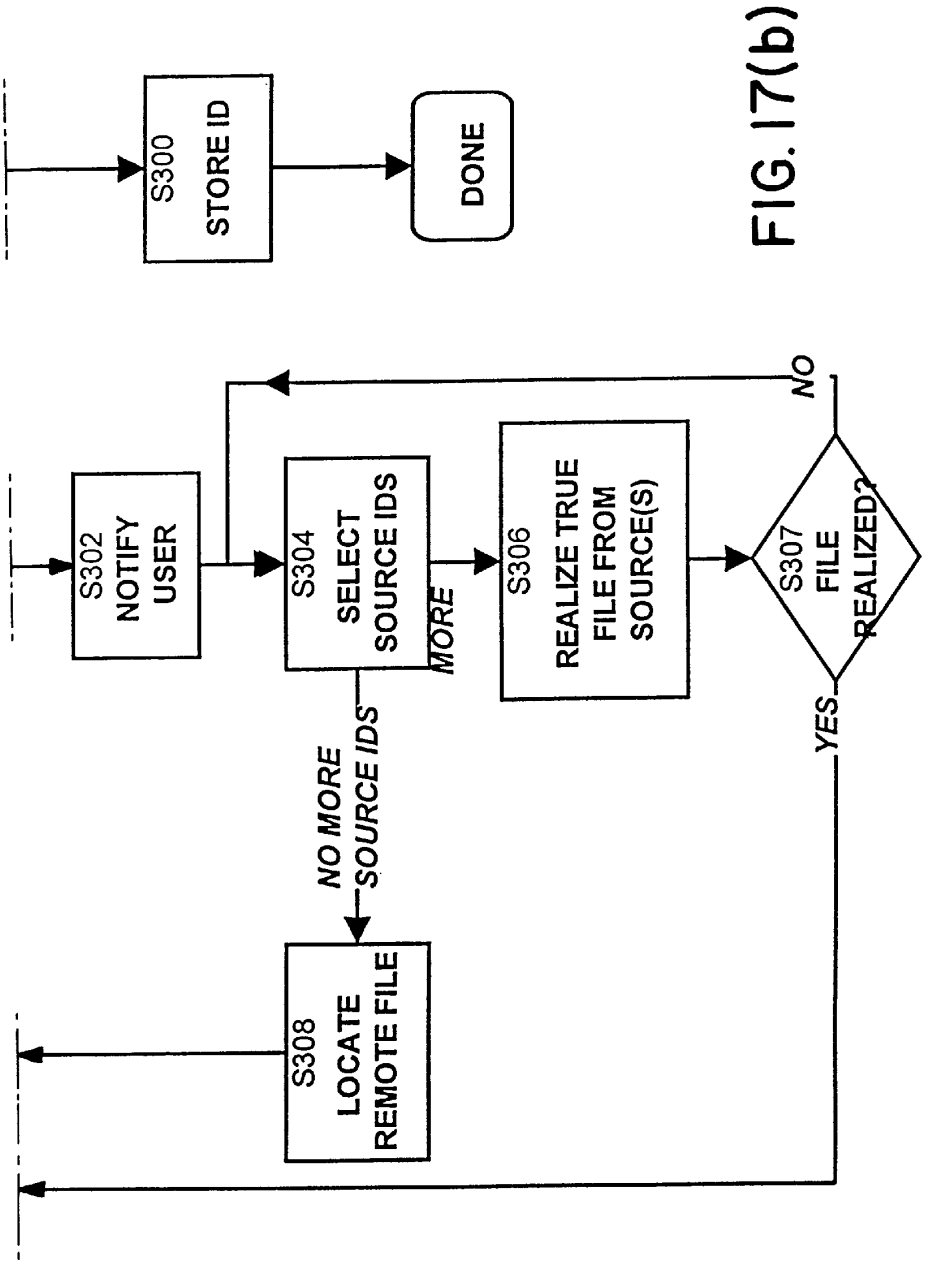
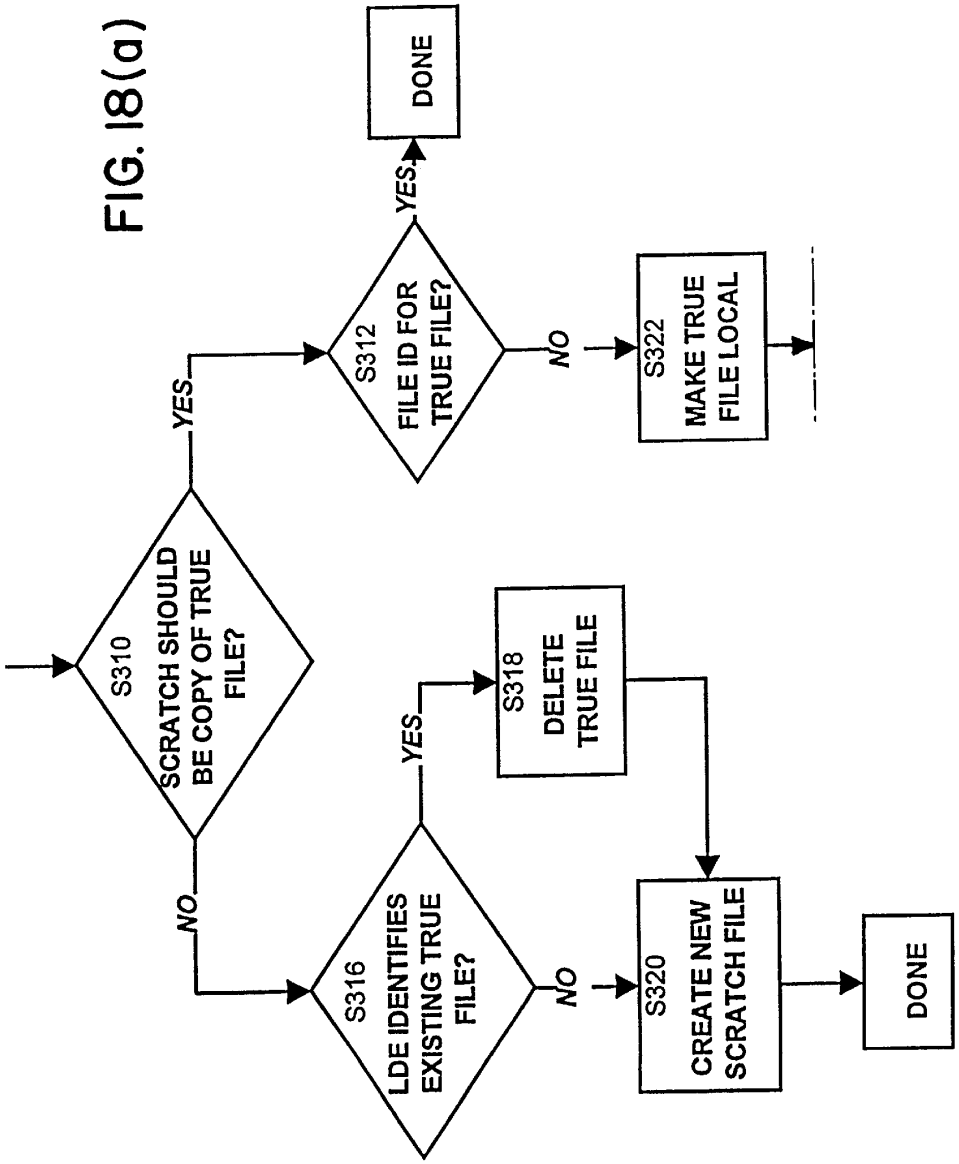

S300
STORE ID
DONE
S302
NOTIFY USER
S304
SELECT SOURCE IDS
MORE
S306
REALIZE TRUE FILE FROM SOURCE(S)
S307
FILE REALIZED?
NO
YES
NO MORE SOURCE IDS
S308
LOCATE REMOTE FILE

FIG. 17(b)

FIG. 18(a)


S310
SCRATCH SHOULD BE COPY OF TRUE FILE?
YES
NO
S312
FILE ID FOR TRUE FILE?
YES
NO
DONE
S322
MAKE TRUE FILE LOCAL
S316
LDE IDENTIFIES EXISTING TRUE FILE?
YES
NO
S318
DELETE TRUE FILE
S320
CREATE NEW SCRATCH FILE
DONE

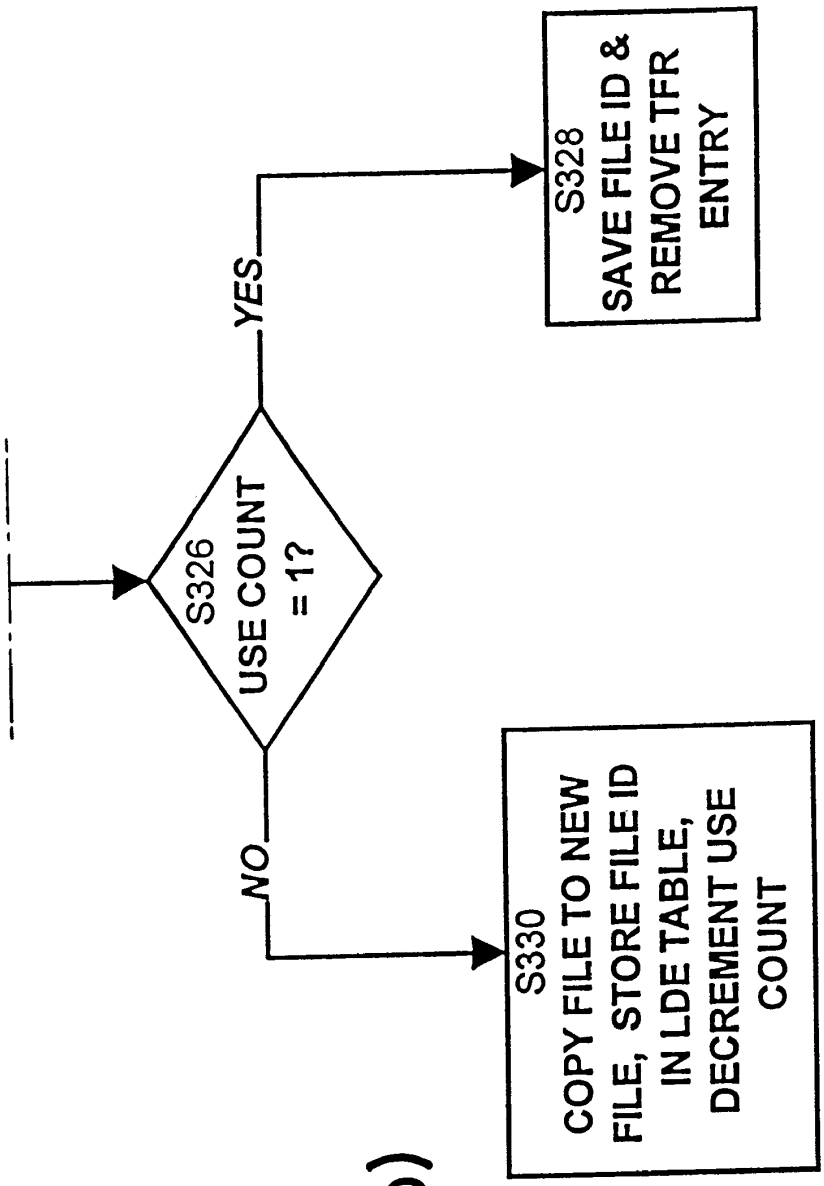
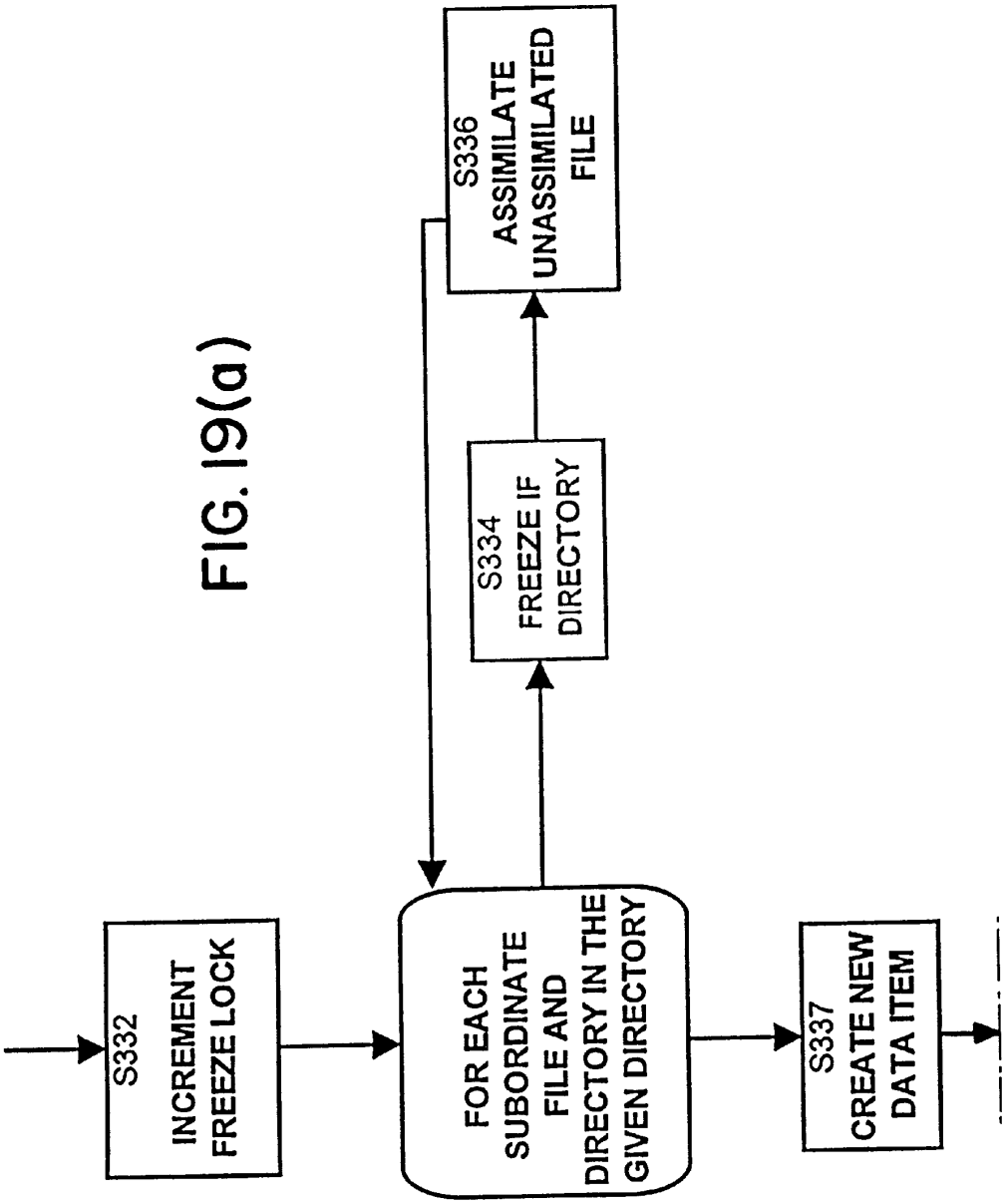

S326
USE COUNT
= 1?
YES
S328
SAVE FILE ID &
REMOVE TFR
ENTRY
NO
S330
COPY FILE TO NEW
FILE, STORE FILE ID
IN LDE TABLE,
DECREMENT USE
COUNT

FIG. 18(b)

FIG. 19(a)


S332
INCREMENT FREEZE LOCK
FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S334
FREEZE IF DIRECTORY
S336
ASSIMILATE UNASSIMILATED FILE
S337
CREATE NEW DATA ITEM

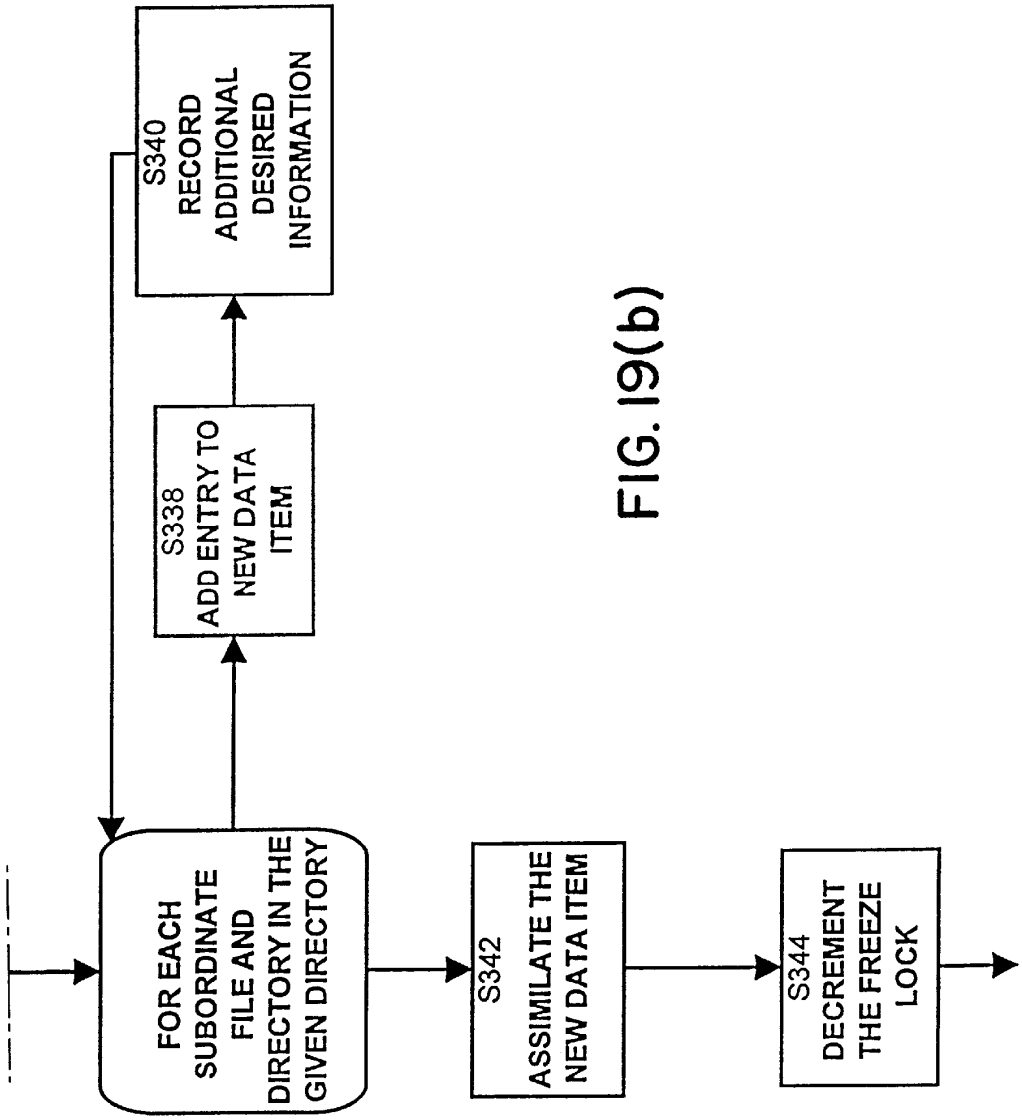

FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S338 ADD ENTRY TO NEW DATA ITEM
S340 RECORD ADDITIONAL DESIRED INFORMATION
S342 ASSIMILATE THE NEW DATA ITEM
S344 DECREMENT THE FREEZE LOCK

FIG. 19(b)

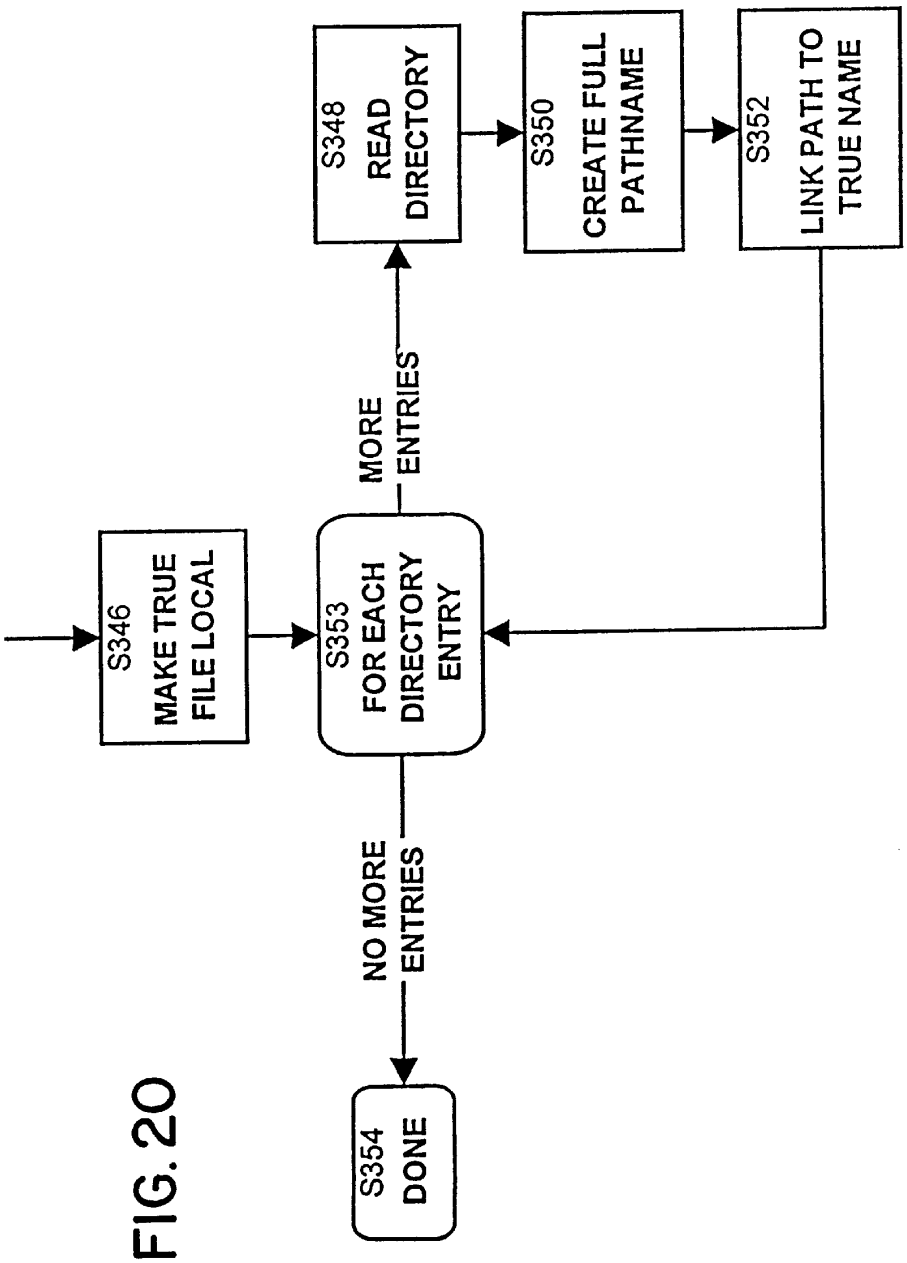

S346
MAKE TRUE FILE LOCAL
S353
FOR EACH DIRECTORY ENTRY
MORE ENTRIES
S348
READ DIRECTORY
S350
CREATE FULL PATHNAME
S352
LINK PATH TO TRUE NAME
NO MORE ENTRIES
S354
DONE

FIG. 20

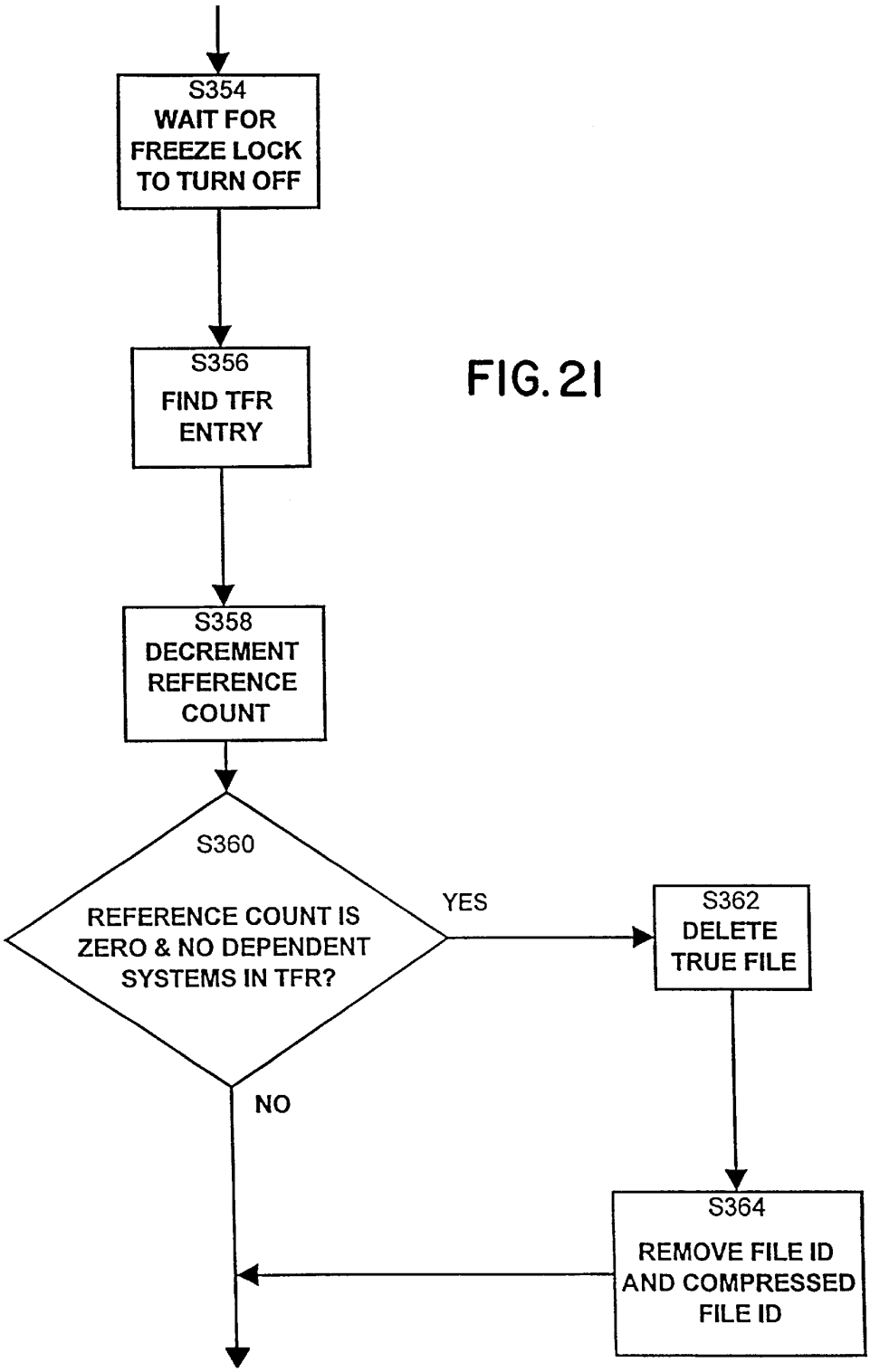

S354
WAIT FOR FREEZE LOCK TO TURN OFF
S356
FIND TFR ENTRY
S358
DECREMENT REFERENCE COUNT
S360
REFERENCE COUNT IS ZERO & NO DEPENDENT SYSTEMS IN TFR?
YES
NO
S362
DELETE TRUE FILE
S364
REMOVE FILE ID AND COMPRESSED FILE ID

FIG. 21

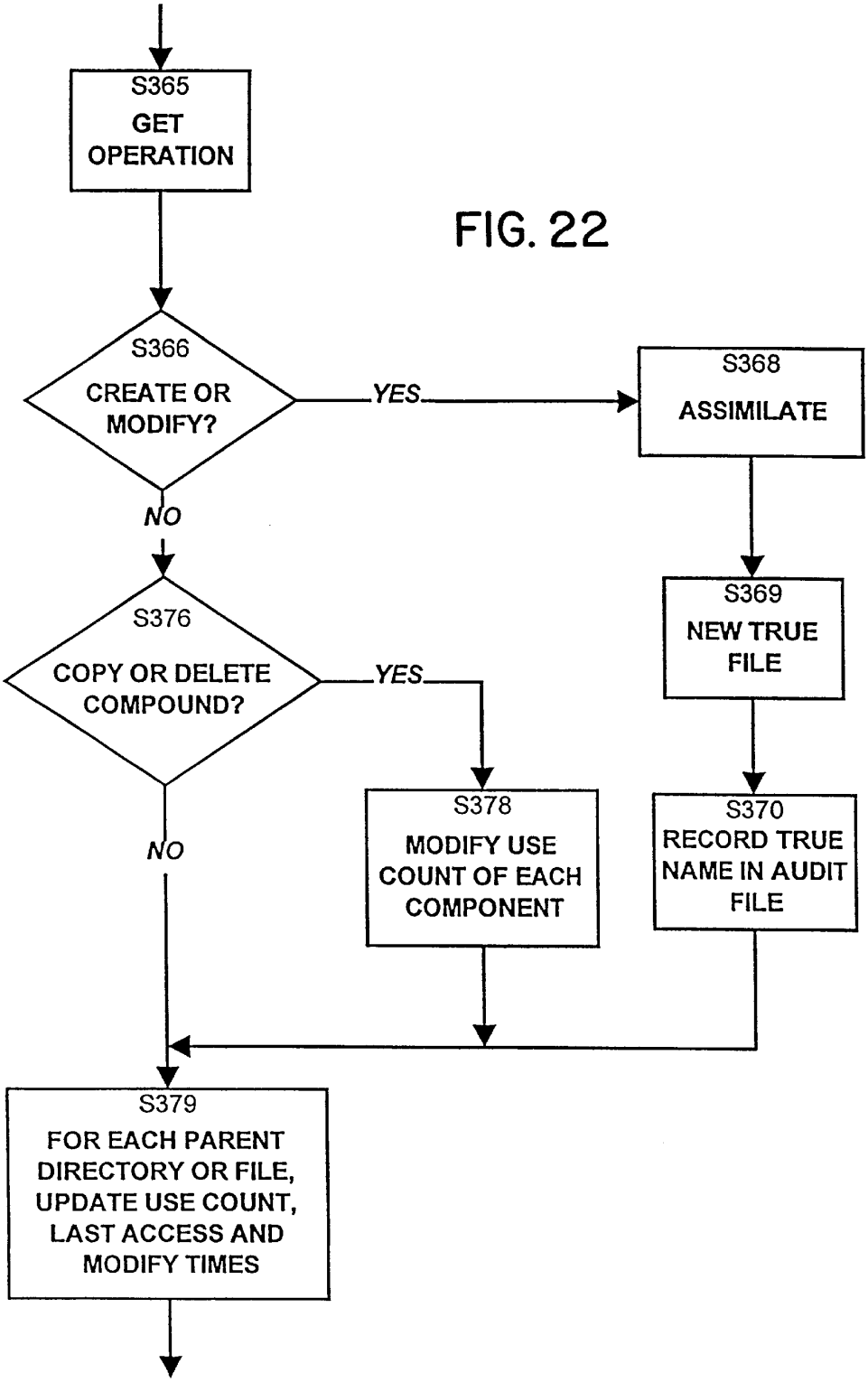
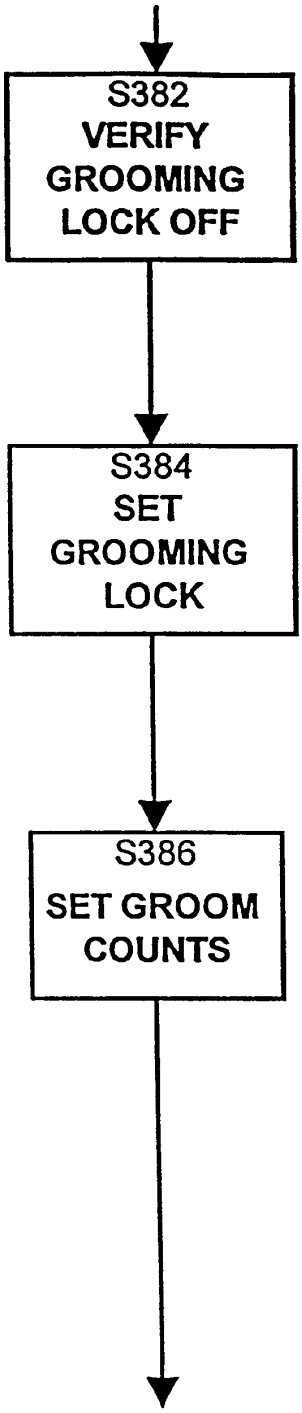

S365
GET OPERATION
S366
CREATE OR MODIFY?
YES
S368
ASSIMILATE
NO
S369
NEW TRUE FILE
S376
COPY OR DELETE COMPOUND?
YES
S378
MODIFY USE COUNT OF EACH COMPONENT
S370
RECORD TRUE NAME IN AUDIT FILE
NO
S379
FOR EACH PARENT DIRECTORY OR FILE, UPDATE USE COUNT, LAST ACCESS AND MODIFY TIMES

FIG. 22

FIG. 23


S382
VERIFY
GROOMING
LOCK OFF
S384
SET
GROOMING
LOCK
S386
SET GROOM
COUNTS

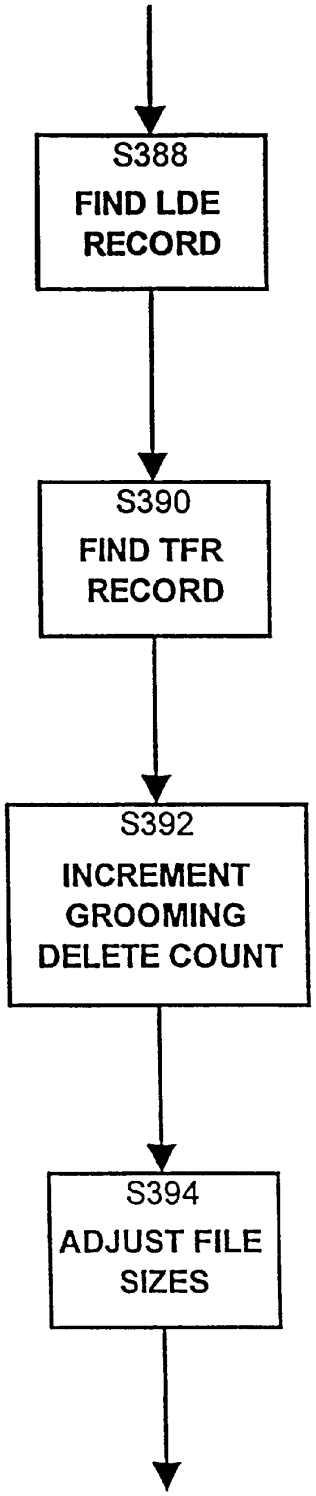

S388
FIND LDE RECORD
S390
FIND TFR RECORD
S392
INCREMENT GROOMING DELETE COUNT
S394
ADJUST FILE SIZES

FIG. 24

FIG. 25

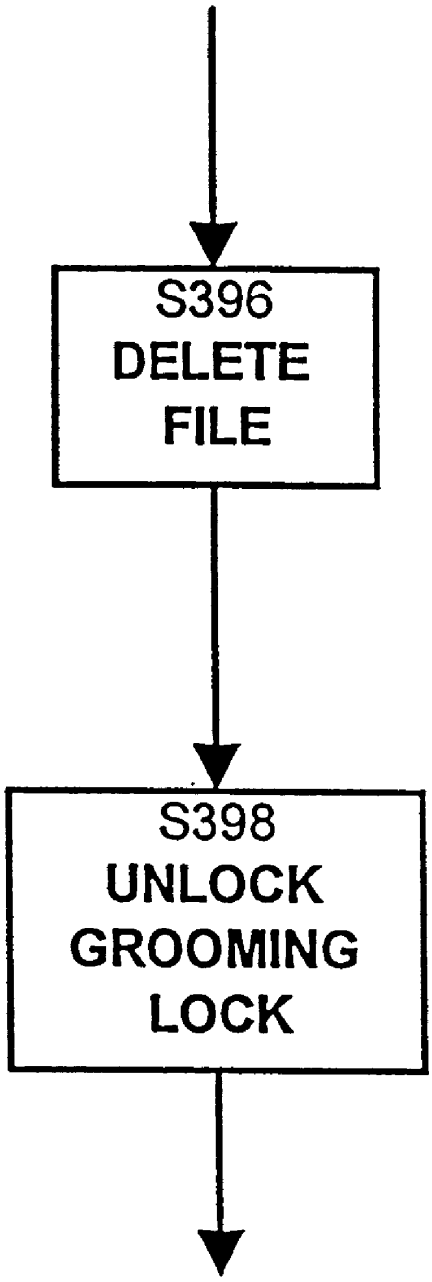
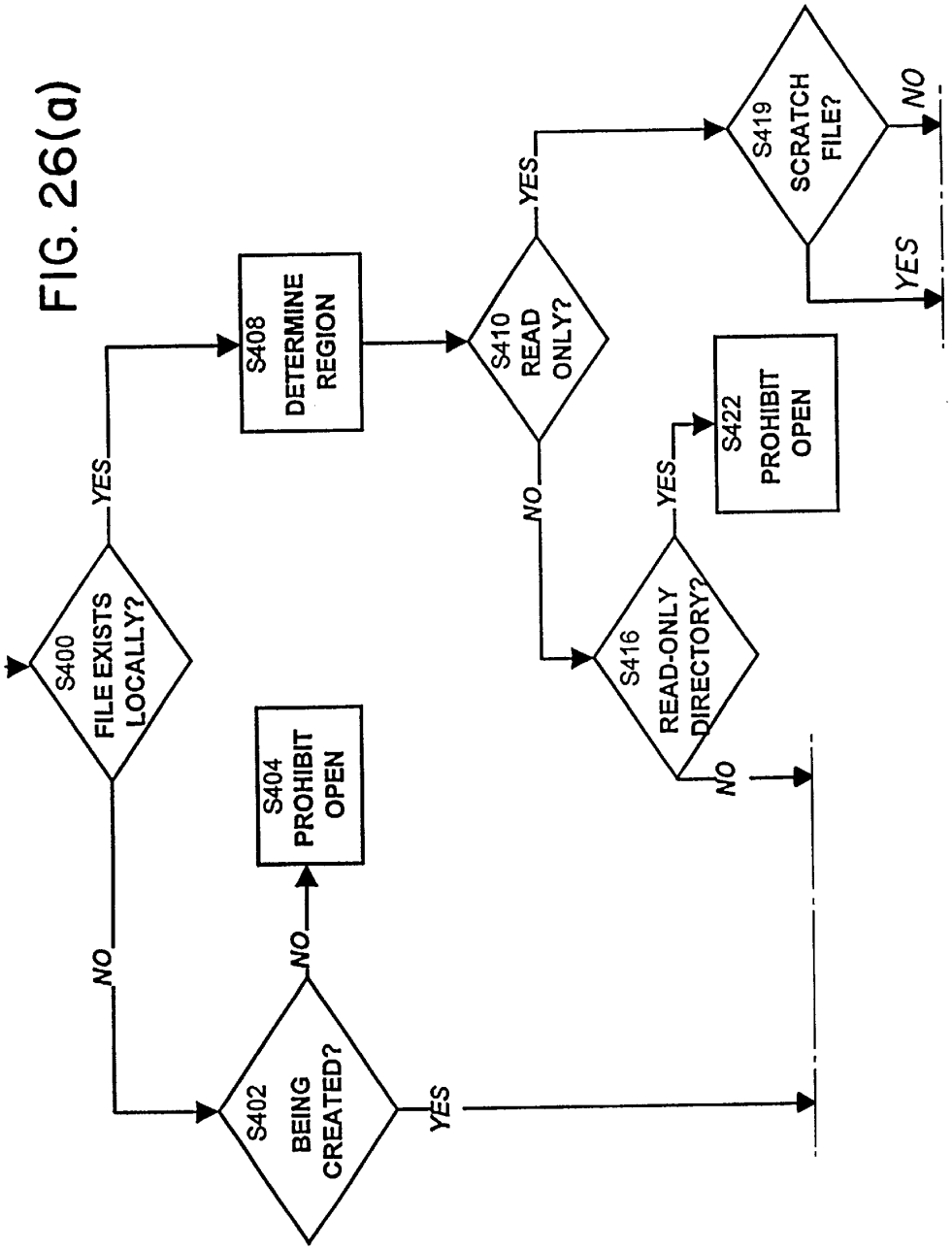


FIG. 26(a)


S400
FILE EXISTS LOCALLY?
YES
NO
S408
DETERMINE REGION
S410
READ ONLY?
YES
NO
S419
SCRATCH FILE?
NO
YES
S422
PROHIBIT OPEN
YES
S416
READ-ONLY DIRECTORY?
NO
S404
PROHIBIT OPEN
NO
S402
BEING CREATED?
YES

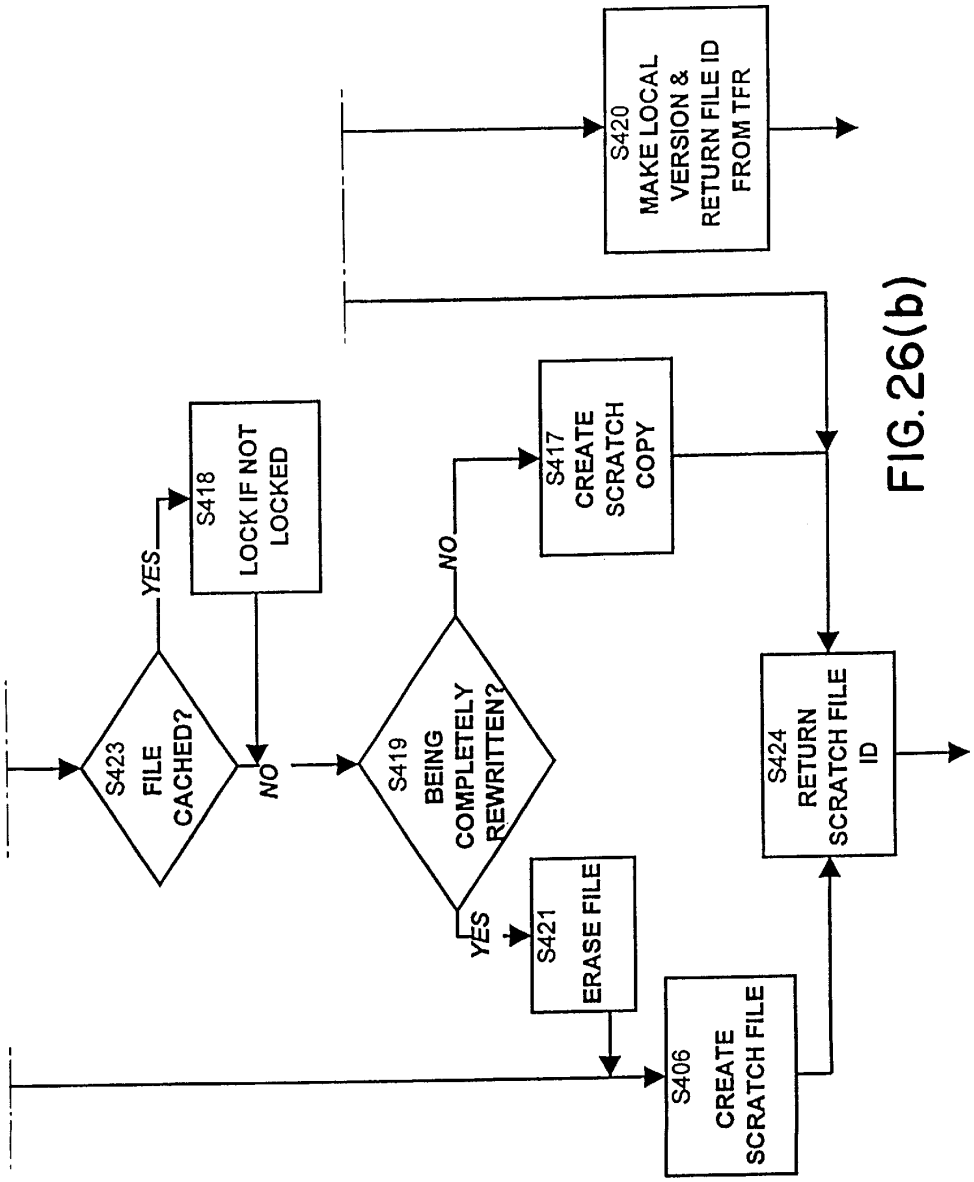

S420
MAKE LOCAL VERSION & RETURN FILE ID FROM TFR
S417
CREATE SCRATCH COPY
S418
LOCK IF NOT LOCKED
YES
NO
S423
FILE CACHED?
NO
S419
BEING COMPLETELY REWRITTEN?
YES
S424
RETURN SCRATCH FILE ID
S421
ERASE FILE
S406
CREATE SCRATCH FILE

FIG. 26(b)

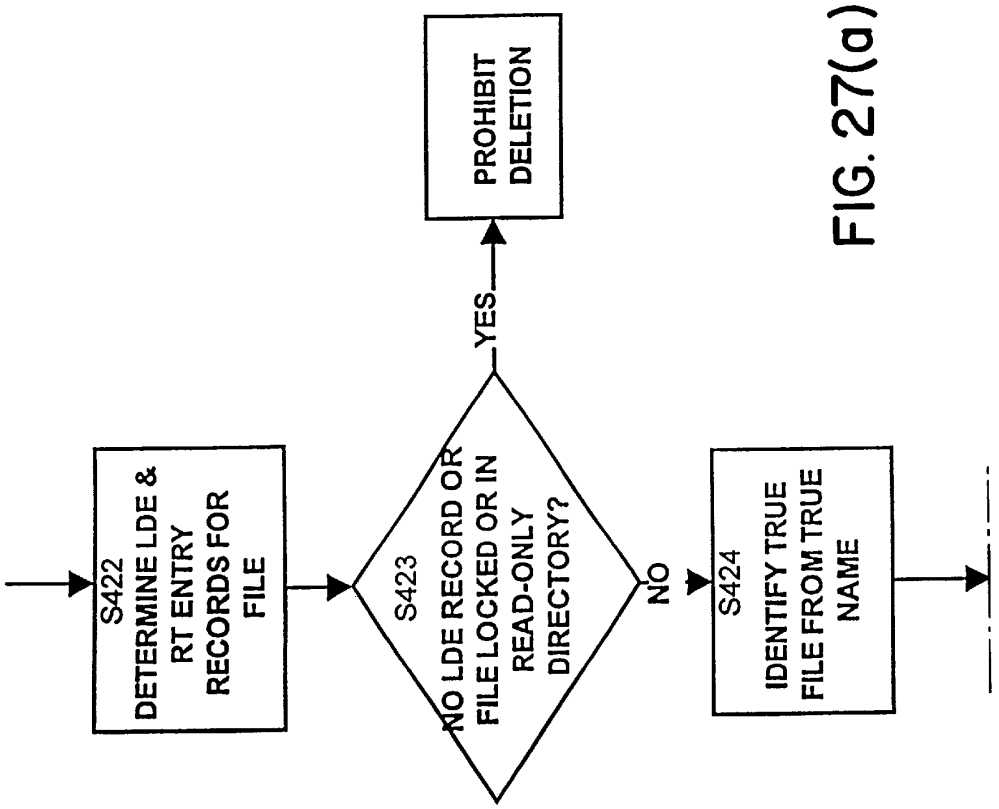

S422
DETERMINE LDE & RT ENTRY RECORDS FOR FILE
S423
NO LDE RECORD OR FILE LOCKED OR IN READ-ONLY DIRECTORY?
YES
PROHIBIT DELETION
NO
S424
IDENTIFY TRUE FILE FROM TRUE NAME

FIG. 27(a)

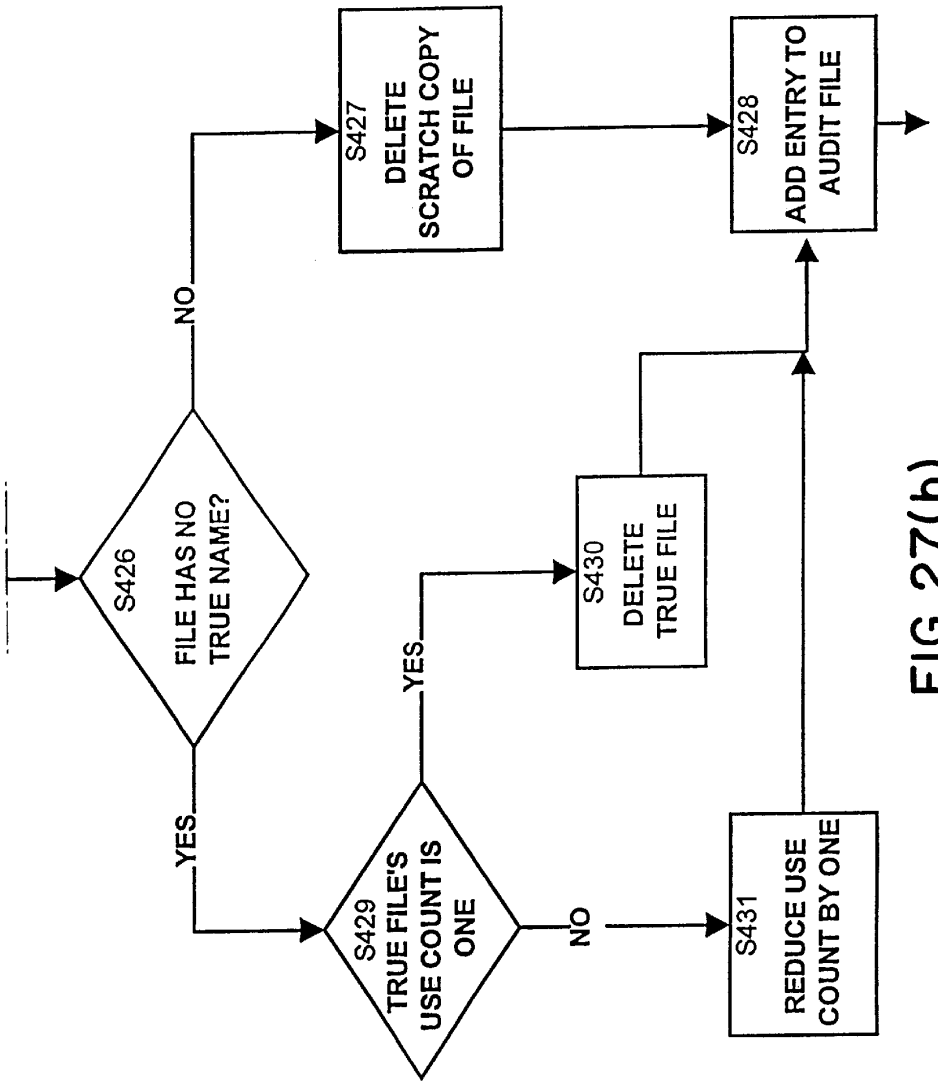
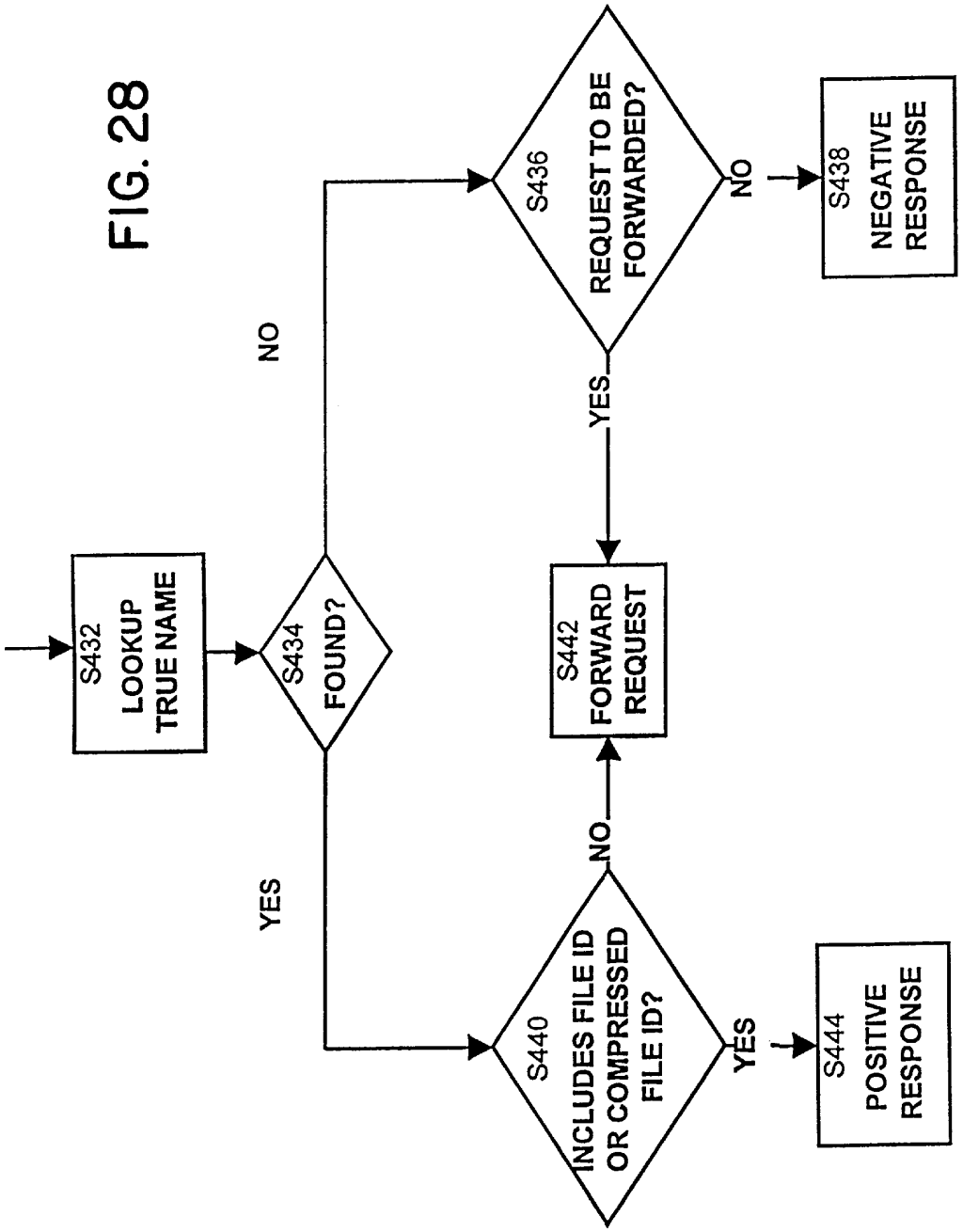

S426
FILE HAS NO TRUE NAME?
NO
S427
DELETE SCRATCH COPY OF FILE
S428
ADD ENTRY TO AUDIT FILE
YES
S429
TRUE FILE'S USE COUNT IS ONE
YES
S430
DELETE TRUE FILE
NO
S431
REDUCE USE COUNT BY ONE

FIG. 27(b)

FIG. 28


S432
LOOKUP TRUE NAME
S434
FOUND?
NO
YES
S436
REQUEST TO BE FORWARDED?
NO
S438
NEGATIVE RESPONSE
YES
S442
FORWARD REQUEST
NO
S440
INCLUDES FILE ID OR COMPRESSED FILE ID?
YES
S444
POSITIVE RESPONSE

DATA PROCESSING SYSTEM USING SUBSTANTIALLY UNIQUE IDENTIFIERS TO IDENTIFY DATA ITEMS, WHEREBY IDENTICAL DATA ITEMS HAVE THE SAME IDENTIFIERS

This is a continuation of application Ser. No. 08/425,160, filed on Apr. 11, 1995, which was abandoned upon the filing hereof.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to data processing systems and, more particularly, to data processing systems wherein data items are identified by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items.

2. Background of the Invention

Data processing (DP) systems, computers, networks of computers, or the like, typically offer users and programs various ways to identify the data in the systems.

Users typically identify data in the data processing system by giving the data some form of name. For example, a typical operating system (OS) on a computer provides a file system in which data items are named by alphanumeric identifiers. Programs typically identify data in the data processing system using a location or address. For example, a program may identify a record in a file or database by using a record number which serves to locate that record.

In all but the most primitive operating systems, users and programs are able to create and use collections of named data items, these collections themselves being named by identifiers. These named collections can then, themselves, be made part of other named collections. For example, an OS may provide mechanisms to group files (data items) into directories (collections). These directories can then, themselves be made part of other directories. A data item may thus be identified relative to these nested directories using a sequence of names, or a so-called pathname, which defines a path through the directories to a particular data item (file or directory).

As another example, a database management system may group data records (data items) into tables and then group these tables into database files (collections). The complete address of any data record can then be specified using the database file name, the table name, and the record number of that data record.

Other examples of identifying data items include: identifying files in a network file system, identifying objects in an object-oriented database, identifying images in an image database, and identifying articles in a text database.

In general, the terms "data" and "data item" as used herein refer to sequences of bits. Thus a data item may be the contents of a file, a portion of a file, a page in memory, an object in an object-oriented program, a digital message, a digital scanned image, a part of a video or audio signal, or any other entity which can be represented by a sequence of bits. The term "data processing" herein refers to the processing of data items, and is sometimes dependent on the type of data item being processed. For example, a data processor for a digital image may differ from a data processor for an audio signal.

In all of the prior data processing systems the names or identifiers provided to identify data items (the data items being files, directories, records in the database, objects in

object-oriented programming, locations in memory or on a physical device, or the like) are always defined relative to a specific context. For instance, the file identified by a particular file name can only be determined when the directory containing the file (the context) is known. The file identified by a pathname can be determined only when the file system (context) is known. Similarly, the addresses in a process address space, the keys in a database table, or domain names on a global computer network such as the Internet are meaningful only because they are specified relative to a context.

In prior art systems for identifying data items there is no direct relationship between the data names and the data item. The same data name in two different contexts may refer to different data items, and two different data names in the same context may refer to the same data item.

In addition, because there is no correlation between a data name and the data it refers to, there is no a priori way to confirm that a given data item is in fact the one named by a data name. For instance, in a DP system, if one processor requests that another processor deliver a data item with a given data name, the requesting processor cannot, in general, verify that the data delivered is the correct data (given only the name). Therefore it may require further processing, typically on the part of the requestor, to verify that the data item it has obtained is, in fact, the item it requested.

A common operation in a DP system is adding a new data item to the system. When a new data item is added to the system, a name can be assigned to it only by updating the context in which names are defined. Thus such systems require a centralized mechanism for the management of names. Such a mechanism is required even in a multiprocessing system when data items are created and identified at separate processors in distinct locations, and in which there is no other need for communication when data items are added.

In many data processing systems or environments, data items are transferred between different locations in the system. These locations may be processors in the data processing system, storage devices, memory, or the like. For example, one processor may obtain a data item from another processor or from an external storage device, such as a floppy disk, and may incorporate that data item into its system (using the name provided with that data item).

However, when a processor (or some location) obtains a data item from another location in the DP system, it is possible that this obtained data item is already present in the system (either at the location of the processor or at some other location accessible by the processor) and therefore a duplicate of the data item is created. This situation is common in a network data processing environment where proprietary software products are installed from floppy disks onto several processors sharing a common file server. In these systems, it is often the case that the same product will be installed on several systems, so that several copies of each file will reside on the common file server.

In some data processing systems in which several processors are connected in a network, one system is designated as a cache server to maintain master copies of data items, and other systems are designated as cache clients to copy local copies of the master data items into a local cache on an as-needed basis. Before using a cached item, a cache client must either reload the cached item, be informed of changes to the cached item, or confirm that the master item corresponding to the cached item has not changed. In other words,

a cache client must synchronize its data items with those on the cache server. This synchronization may involve reloading data items onto the cache client. The need to keep the cache synchronized or reload it adds significant overhead to existing caching mechanisms.

In view of the above and other problems with prior art systems, it is therefore desirable to have a mechanism which allows each processor in a multiprocessor system to determine a common and substantially unique identifier for a data item, using only the data in the data item and not relying on any sort of context.

It is further desirable to have a mechanism for reducing multiple copies of data items in a data processing system and to have a mechanism which enables the identification of identical data items so as to reduce multiple copies. It is further desirable to determine whether two instances of a data item are in fact the same data item, and to perform various other systems' functions and applications on data items without relying on any context information or properties of the data item.

It is also desirable to provide such a mechanism in such a way as to make it transparent to users of the data processing system, and it is desirable that a single mechanism be used to address each of the problems described above.

SUMMARY OF THE INVENTION

This invention provides, in a data processing system, a method and apparatus for identifying a data item in the system, where the identity of the data item depends on all of the data in the data item and only on the data in the data item. Thus the identity of a data item is independent of its name, origin, location, address, or other information not derivable directly from the data, and depends only on the data itself.

This invention further provides an apparatus and a method for determining whether a particular data item is present in the system or at a location in the system, by examining only the data identities of a plurality of data items.

Using the method or apparatus of the present invention, the efficiency and integrity of a data processing system can be improved. The present invention improves the design and operation of a data storage system, file system, relational database, object-oriented database, or the like that stores a plurality of data items, by making possible or improving the design and operation of at least some or all of the following features:

the system stores at most one copy of any data item at a given location, even when multiple data names in the system refer to the same contents;

the system avoids copying data from source to destination locations when the destination locations already have the data;

the system provides transparent access to any data item by reference only to its identity and independent of its present location, whether it be local, remote, or offline;

the system caches data items from a server, so that only the most recently accessed data items need be retained;

when the system is being used to cache data items, problems of maintaining cache consistency are avoided;

the system maintains a desired level of redundancy of data items in a network of servers, to protect against failure by ensuring that multiple copies of the data items are present at different locations in the system;

the system automatically archives data items as they are created or modified;

the system provides the size, age, and location of groups of data items in order to decide whether they can be safely removed from a local file system;

the system can efficiently record and preserve any collection of data items;

the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items;

the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of the data items;

the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM, the inventory is independent of other properties of the data items such as their name, location, and date of creation;

the system allows closely related sets of data items, such as matching or corresponding directories on disconnected computers, to be periodically resynchronized with one another;

the system can verify that data retrieved from another location is the desired or requested data, using only the data identifier used to retrieve the data;

the system can prove possession of specific data items by content without disclosing the content of the data items, for purposes of later legal verification and to provide anonymity;

the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes.

Other objects, features, and characteristics of the present invention as well as the methods of operation and functions of the related elements of structure, and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1**(*a*) and **1**(*b*) depicts a typical data processing system in which a preferred embodiment of the present invention operates;

FIG. **2** depicts a hierarchy of data items stored at any location in such a data processing system;

FIGS. **3**–**9** depict data structures used to implement an embodiment of the present invention; and

FIGS. **10**(*a*)–**28** are flow charts depicting operation of various aspects of the present invention.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

An embodiment of the present invention is now described with reference to a typical data processing system **100**, which, with reference to FIGS. **1**(*a*) and **1**(*b*), includes one or more processors (or computers) **102** and various storage devices **104** connected in some way, for example by a bus **106**.

Each processor **102** includes a CPU **108**, a memory **110** and one or more local storage devices **112**. The CPU **108**, memory **110**, and local storage device **112** may be internally connected, for example by a bus **114**. Each processor **102**

may also include other devices (not shown), such as a keyboard, a display, a printer, and the like.

In a data processing system **100**, wherein more than one processor **102** is used, that is, in a multiprocessor system, the processors may be in one of various relationships. For example, two processors **102** may be in a client/server, client/client, or a server/server relationship. These inter-processor relationships may be dynamic, changing depending on particular situations and functions. Thus, a particular processor **102** may change its relationship to other processors as needed, essentially setting up a peer-to-peer relationship with other processors. In a peer-to-peer relationship, sometimes a particular processor **102** acts as a client processor, whereas at other times the same processor acts as a server processor. In other words, there is no hierarchy imposed on or required of processors **102**.

In a multiprocessor system, the processors **102** may be homogeneous or heterogeneous. Further, in a multiprocessor data processing system **100**, some or all of the processors **102** may be disconnected from the network of processors for periods of time. Such disconnection may be part of the normal operation of the system **100** or it may be because a particular processor **102** is in need of repair.

Within a data processing system **100**, the data may be organized to form a hierarchy of data storage elements, wherein lower level data storage elements are combined to form higher level elements. This hierarchy can consist of, for example, processors, file systems, regions, directories, data files, segments, and the like. For example, with reference to FIG. **2**, the data items on a particular processor **102** may be organized or structured as a file system **116** which comprises regions **117**, each of which comprises directories **118**, each of which can contain other directories **118** or files **120**. Each file **120** being made up of one or more data segments **122**.

In a typical data processing system, some or all of these elements can be named by users given certain implementation specific naming conventions, the name (or pathname) of an element being relative to a context. In the context of a data processing system **100**, a pathname is fully specified by a processor name, a filesystem name, a sequence of zero or more directory names identifying nested directories, and a final file name. (Usually the lowest level elements, in this case segments **122**, cannot be named by users.)

In other words, a file system **116** is a collection of directories **118**. A directory **118** is a collection of named files **120**—both data files **120** and other directory files **118**. A file **120** is a named data item which is either a data file (which may be simple or compound) or a directory file **118**. A simple file **120** consists of a single data segment **122**. A compound file **120** consists of a sequence of data segments **122**. A data segment **122** is a fixed sequence of bytes. An important property of any data segment is its size, the number of bytes in the sequence.

A single processor **102** may access one or more file systems **116**, and a single storage device **104** may contain one or more file systems **116**, or portions of a file system **116**. For instance, a file system **116** may span several storage devices **104**.

In order to implement controls in a file system, file system **116** may be divided into distinct regions, where each region is a unit of management and control. A region consists of a given directory **118** and is identified by the pathname (user defined) of the directory.

In the following, the term "location", with respect to a data processing system **100**, refers to any of a particular processor **102** in the system, a memory of a particular processor, a storage device, a removable storage medium (such as a floppy disk or compact disk), or any other physical location in the system. The term "local" with respect to a particular processor **102** refers to the memory and storage devices of that particular processor.

In the following, the terms "True Name", "data identity" and "data identifier" refer to the substantially unique data identifier for a particular data item. The term "True File" refers to the actual file, segment, or data item identified by a True Name.

A file system for a data processing system **100** is now described which is intended to work with an existing operating system by augmenting some of the operating system's file management system codes. The embodiment provided relies on the standard file management primitives for actually storing to and retrieving data items from disk, but uses the mechanisms of the present invention to reference and access those data items.

The processes and mechanisms (services) provided in this embodiment are grouped into the following categories: primitive mechanisms, operating system mechanisms, remote mechanisms, background mechanisms, and extended mechanisms.

Primitive mechanisms provide fundamental capabilities used to support other mechanisms. The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

Operating system mechanisms provide typical familiar file system mechanisms, while maintaining the data structures required to offer the mechanisms of the present invention. Operating system mechanisms are designed to augment existing operating systems, and in this way to make the present invention compatible with, and generally transparent to, existing applications. The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

Remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a

peer-to-peer network mode of operation. The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

Background mechanisms are intended to run occasionally and at a low priority. These provide automated management capabilities with respect to the present invention. The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links; and
4. Verify Region; and
5. Groom Source List.

Extended mechanisms run within application programs over the operating system. These mechanisms provide solutions to specific problems and applications. The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at location;
7. Verify True File;
8. Track for accounting purposes; and
9. Track for licensing purposes.

The file system herein described maintains sufficient information to provide a variety of mechanisms not ordinarily offered by an operating system, some of which are listed and described here. Various processing performed by this embodiment of the present invention will now be described in greater detail.

In some embodiments, some files **120** in a data processing system **100** do not have True Names because they have been recently received or created or modified, and thus their True Names have not yet been computed. A file that does not yet have a True Name is called a scratch file. The process of assigning a True Name to a file is referred to as assimilation, and is described later. Note that a scratch file may have a user provided name.

Some of the processing performed by the present invention can take place in a background mode or on a delayed or as-needed basis. This background processing is used to determine information that is not immediately required by the system or which may never be required. As an example, in some cases a scratch file is being changed at a rate greater than the rate at which it is useful to determine its True Name. In these cases, determining the True Name of the file can be postponed or performed in the background.

Data Structures

The following data structures, stored in memory **110** of one of more processors **102** are used to implement the mechanisms described herein. The data structures can be local to each processor **102** of the system **100**, or they can reside on only some of the processors **102**.

The data structures described are assumed to reside on individual peer processors **102** in the data processing system **100**. However, they can also be shared by placing them on a remote, shared file server (for instance, in a local area network of machines). In order to accommodate sharing data structures, it is necessary that the processors accessing the shared database use the appropriate locking techniques to ensure that changes to the shared database do not interfere with one another but are appropriately serialized. These locking techniques are well understood by ordinarily skilled programmers of distributed applications.

It is sometimes desirable to allow some regions to be local to a particular processor **102** and other regions to be shared among processors **102**. (Recall that a region is a unit of file system management and control consisting of a given directory identified by the pathname of the directory.) In the case of local and shared regions, there would be both local and shared versions of each data structure. Simple changes to the processes described below must be made to ensure that appropriate data structures are selected for a given operation.

The local directory extensions (LDE) table **124** is a data structure which provides information about files **120** and directories **118** in the data processing system **100**. The local directory extensions table **124** is indexed by a pathname or contextual name (that is, a user provided name) of a file and includes the True Name for most files. The information in local directory extension table **124** is in addition to that provided by the native file system of the operating system.

The True File registry (TFR) **126** is a data store for listing actual data items which have True Names, both files **120** and segments **122**. When such data items occur in the True File registry **126** they are known as True Files. True Files are identified in True File registry **126** by their True Names or identities. The table True File registry **126** also stores location, dependency, and migration information about True Files.

The region table (RT) **128** defines areas in the network storage which are to be managed separately. Region table **128** defines the rules for access to and migration of files **120** among various regions with the local file system **116** and remote peer file systems.

The source table (ST) **130** is a list of the sources of True Files other than the current True File registry **126**. The source table **130** includes removable volumes and remote processors.

The audit file (AF) **132** is a list of records indicating changes to be made in local or remote files, these changes to be processed in background.

The accounting log (AL) **134** is a log of file transactions used to create accounting information in a manner which preserves the identity of files being tracked independent of their name or location.

The license table (LT) **136** is a table identifying files, which may only be used by licensed users, in a manner independent of their name or location, and the users licensed to use them.

Detailed Descriptions of the Data Structures

The following table summarizes the fields of an local directory extensions table entry, as illustrated by record **138** in FIG. **3**.

Field	Description
Region ID	identifies the region in which this file is contained.
Pathname	the user provided name or contextural name

-continued

Field	Description
	of the file or directory, relative to the region in which it occurs.
True Name	the computed True Name or identity of the file or directory. This True Name is not always up th date, and it is set to a special value when a file is modified and is later recomputed in the background.
Type	indicates whether the file is a data file or a directory.
Scratch File ID	the physical location of the file in the file system, when no True Name has been calculated for the file. As noted above, such a file is called a scratch file.
Time of last access	the last access time to this file. If this file is a directory, this is the last access time to any file in the directory.
Time of last modification	the time of last change of this file. If this file is a directory, this is the last modification time of any file in the directory.
Safe flag	indicates that this file (and, if this file is a directory, all of its subordinate files) have been backed up on some other system, and it is therefore safe to remove them.
Lock flag	indicates whether a file is locked, that is, it is being modified by the local processor or a remote processor. Only one processor may modify a file at a time.
Size	the full size of this directory (including all subordinate files), if all files in it were fully expanded and duplicated. For a file that is not a directory this is the size of the actual True File.
Owner	the identity of the user who owns this file, for accounting and license tracking purposes.

Each record of the True File registry **126** has the fields shown in the True File registry record **140** in FIG. **4**. The True File registry **126** consists of the database described in the table below as well as the actual True Files identified by the True File IDs below.

Field	Description
True Name	computed True Name or identity of the file.
Compressed File ID	compressed version of the True File may be stored insteaded of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.
Grooming delete count	tentative count of how many references have been selected for deletion during a grooming operation.
Time of last access	most recent date and time the content of this file was accessed.
Expiration	date and time after which this file may be deleted by this server.
Dependent processors	processor IDs of other processors which contain references to this True File.
Source IDs	source ID(s) of zero or more sources form which this file or data item may be retrieved.
True File ID	identity or disk location of the actual physical representation of the file or file segment. It is sufficient to use a filename in the registration directory of the underlying operation system. The True File ID is absent if the actual file is not currently present at the current location.
Use count	number of other records on this processor which identify this True File.

A region table **128**, specified by a directory pathname, records storage policies which allow files in the file system to be stored, accessed and migrated in different ways. Storage policies are programmed in a configurable way using a set of rules described below.

Each region table record **142** of region table **128** includes the fields described in the following table (with reference to FIG. **5**):

Field	Description
Region ID	internally used identifier for this region.
Region file system	file system on the local processor of which this region is a part.
Region pathname	a pathname relative to the region file system which defines the location of this region. The region consists of all files and directories subordinate to this pathname, except those in a region subordinate to this region.
Mirror processor(s)	zero or more identifiers of processors which are to keep mirror or archival copies of all files in the current region. Multiple mirror processors can be defined to form a mirror group.
Mirror duplication count	number of copies of each file in this region that should be retained in a mirror group.
Region status	specifies whether this region is local to a single processor 102, shared by several processors 102 (if, for instance, it resides on a shared file server), or managed by a remote processor.
Policy	the migration policy to apply to this region. A single region might participate in several policies. The policies are as follows (parameters in brackets are specified as part of the policy): region is a cached version from [processor ID]; region is a member of a mirror set defined by [processor ID]. region is to be archived on [processor ID]. region is to be backed up locally, by placing new copies in [region ID]. region is read only and may not be changed. region is published and expires on [date]. Files in this region should be compressed.

A source table **130** identifies a source location for True Files. The source table **130** is also used to identify client processors making reservations on the current processor. Each source record **144** of the source table **130** includes the fields summarized in the following table, with reference to FIG. **6**:

Field	Description
source ID	internal identifier used to identify a particular source.
source type	type of source location: Removable Storage Volume Local Region Cache Server Mirror Group Server Cooperative Server Publishing Server Client
source rights	includes information about the rights of this processor, such as whether it can ask the local processor to store data items for it.
source availability	measurement of the bandwidth, cost, and reliability of the connection to this source of True Files. The availability is used to select from among several possible sources.
source location	information on how the local processor is to access the source. This may be, for example, the name of a removable storage volume, or the processor ID and region path of a region on a remote processor.

The audit file **132** is a table of events ordered by timestamp, each record **146** in audit file **132** including the fields summarized in the following table (with reference to FIG. **7**):

Field	Description
Original Name	path of the file in question.
Operation	whether the file was created, read, written, copied or deleted.
Type	specifies whether the source is a file or a directory.
Processor ID	ID of the remote processor generating this event (if not local).
Timestamp	time and date file was closed (required only for accessed/modified files).
Pathname	Name of the file (required only for rename).
True Name	computed True Name of the file. This is used by remote systems to mirror changes to the directory and is filled in during background processing.

Each record **148** of the accounting log **134** records an event which may later be used to provide information for billing mechanisms. Each accounting log entry record **148** includes at least the information summarized in the following table, with reference to FIG. **8**:

Field	Description
date of entry	date and time of this log entry.
type of entry	Entry types include create file, delete file, and transmit file.
True Name	True Name of data item in question.
owner	identity of the user responsible for this action.

Each record **150** of the license table **136** records a relationship between a licensable data item and the user licensed to have access to it. Each license table record **150** includes the information summarized in the following table, with reference to FIG. **9**:

Field	Description
True Name	True Name of a data item subject to license validation.
licensee	identity of a user authorized to have access to this object.

Various other data structures are employed on some or all of the processors **102** in the data processing system **100**. Each processor **102** has a global freeze lock (GFL) **152** (FIG. **1**), which is used to prevent synchronization errors when a directory is frozen or copied. Any processor **102** may include a special archive directory (SAD) **154** into which directories may be copied for the purposes of archival. Any processor **102** may include a special media directory (SMD) **156**, into which the directories of removable volumes are stored to form a media inventory. Each processor has a grooming lock **158**, which is set during a grooming operation. During this period the grooming delete count of True File registry entries **140** is active, and no True Files should be deleted until grooming is complete. While grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

Primitive Mechanisms

The first of the mechanisms provided by the present invention, primitive mechanisms, are now described. The mechanisms described here depend on underlying data management mechanisms to create, copy, read, and delete data items in the True File registry **126**, as identified by a True File ID. This support may be provided by an underlying operating system or disk storage manager.

The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link Path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

1. Calculate True Name

A True Name is computed using a function, MD, which reduces a data block B of arbitrary length to a relatively small, fixed size identifier, the True Name of the data block, such that the True Name of the data block is virtually guaranteed to represent the data block B and only data block B.

The function MD must have the following properties:

1. The domain of the function MD is the set of all data items. The range of the function MD is the set of True Names.
2. The function MD must take a data item of arbitrary length and reduce it to an integer value in the range 0 to N−1, where N is the cardinality of the set of True

Names. That is, for an arbitrary length data block B, $0 \leqq MD(B) \leqq N$.

3. The results of MD(B) must be evenly and randomly distributed over the range of N, in such a way that simple or regular changes to B are virtually guaranteed to produce a different value of MD(B).
4. It must be computationally difficult to find a different value B' such that MD(B)=MD(B').
5. The function MD(B) must be efficiently computed.

A family of functions with the above properties are the so-called message digest functions, which are used in digital security systems as techniques for authentification of data. These functions (or algorithms) include MD**4**, MD**5**, and SHA.

In the presently preferred embodiments, either MD**5** or SHA is employed as the basis for the computation of True Names. Whichever of these two message digest functions is employed, that same function must be employed on a system-wide basis.

It is impossible to define a function having a unique output for each possible input when the number of elements in the range of the function is smaller than the number of elements in its domain. However, a crucial observation is that the actual data items that will be encountered in the operation of any system embodying this invention form a very sparse subset of all the possible inputs.

A colliding set of data items is defined as a set wherein, for one or more pairs x and y in the set, MD(x)=MD(y). Since a function conforming to the requirements for MD must evenly and randomly distribute its outputs, it is possible, by making the range of the function large enough, to make the probability arbitrarily small that actual inputs encountered in the operation of an embodiment of this invention will form a colliding set.

To roughly quantify the probability of a collision, assume that there are no more than 2^{30} storage devices in the world, and that each storage device has an average of at most 2^{20} different data items. Then there are at most 2^{50} data items in the world. If the outputs of MD range between 0 and 2^{128}, it can be demonstrated that the probability of a collision is approximately 1 in 2^{29}. Details on the derivation of these probability values are found, for example, in P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," *Lecture Notes in Computer Science* 434*: Advances in Cryptology—Eurocrypt* '89 *Proceedings*, Springer-Verlag, pp. 329–354.

Note that for some less-preferred embodiments of the present invention, lower probabilities of uniqueness may be acceptable, depending on the types of applications and mechanisms used. In some embodiments it may also be useful to have more than one level of True Names, with some of the True Names having different degrees of uniqueness. If such a scheme is implemented, it is necessary to ensure that less unique True Names are not propagated in the system.

While the invention is described herein using only the True Name of a data item as the identifier for the data item, other preferred embodiments use tagged, typed, categorized or classified data items and use a combination of both the True Name and the tag, type, category or class of the data item as an identifier. Examples of such categorizations are files, directories, and segments; executable files and data files, and the like. Examples of classes are classes of objects in an object-oriented system. In such a system, a lower degree of True Name uniqueness is acceptable over the entire universe of data items, as long as sufficient uniqueness is provided per category of data items. This is because the tags provide an additional level of uniqueness.

A mechanism for calculating a True Name given a data item is now described, with reference to FIGS. **10**(*a*) and **10**(*b*).

A simple data item is a data item whose size is less than a particular given size (which must be defined in each particular implementation of the invention). To determine the True Name of a simple data item, with reference to FIG. **10**(*a*), first compute the MD function (described above) on the given simple data item (Step S**212**). Then append to the resulting 128 bits, the byte length modulo **32** of the data item (Step S**214**). The resulting 160-bit value is the True Name of the simple data item.

A compound data item is one whose size is greater than the particular given size of a simple data item. To determine the True Name of an arbitrary (simple or compound) data item, with reference to FIG. **10**(*b*), first determine if the data item is a simple or a compound data item (Step S**216**). If the data item is a simple data item, then compute its True Name in step S**218** (using steps S**212** and S**214** described above), otherwise partition the data item into segments (Step S**220**) and assimilate each segment (Step S**222**) (the primitive mechanism, Assimilate a Data Item, is described below), computing the True Name of the segment. Then create an indirect block consisting of the computed segment True Names (Step S**224**). An indirect block is a data item which consists of the sequence of True Names of the segments. Then, in step S**226**, assimilate the indirect block and compute its True Name. Finally, replace the final thirty-two (32) bits of the resulting True Name (that is, the length of the indirect block) by the length modulo **32** of the compound data item (Step S**228**). The result is the True Name of the compound data item.

Note that the compound data item may be so large that the indirect block of segment True Names is itself a compound data item. In this case the mechanism is invoked recursively until only simple data items are being processed.

Both the use of segments and the attachment of a length to the True Name are not strictly required in a system using the present invention, but are currently considered desirable features in the preferred embodiment.

2. Assimilate Data Item

A mechanism for assimilating a data item (scratch file or segment) into a file system, given the scratch file ID of the data item, is now described with reference to FIG. **11**. The purpose of this mechanism is to add a given data item to the True File registry **126**. If the data item already exists in the True File registry **126**, this will be discovered and used during this process, and the duplicate will be eliminated.

Thereby the system stores at most one copy of any data item or file by content, even when multiple names refer to the same content.

First, determine the True Name of the data item corresponding to the given scratch File ID using the Calculate True Name primitive mechanism (Step S**230**). Next, look for an entry for the True Name in the True File registry **126** (Step S**232**) and determine whether a True Name entry, record **140**, exists in the True File registry **126**. If the entry record includes a corresponding True File ID or compressed File ID (Step S**237**), delete the file with the scratch File ID (Step S**238**). Otherwise store the given True File ID in the entry record (step S**239**).

If it is determined (in step S**232**) that no True Name entry exists in the True File registry **126**, then, in Step S**236**, create a new entry in the True File registry **126** for this True Name. Set the True Name of the entry to the calculated True Name, set the use count for the new entry to one, store the given True File ID in the entry and set the other fields of the entry as appropriate.

Because this procedure may take some time to compute, it is intended to run in background after a file has ceased to change. In the meantime, the file is considered an unassimilated scratch file.

3. New True File

The New True File process is invoked when processing the audit file **132**, some time after a True File has been assimilated (using the Assimilate Data Item primitive mechanism). Given a local directory extensions table entry record **138** in the local directory extensions table **124**, the New True File process can provide the following steps (with reference to FIG. **12**), depending on how the local processor is configured:

First, in step S**238**, examine the local directory extensions table entry record **138** to determine whether the file is locked by a cache server. If the file is locked, then add the ID of the cache server to the dependent processor list of the True File registry table **126**, and then send a message to the cache server to update the cache of the current processor using the Update Cache remote mechanism (Step **242**).

If desired, compress the True File (Step S**246**), and, if desired, mirror the True File using the Mirror True File background mechanism (Step S**248**).

4. Get True Name from Path

The True Name of a file can be used to identify a file by contents, to confirm that a file matches its original contents, or to compare two files. The mechanism to get a True Name given the pathname of a file is now described with reference to FIG. **13**.

First, search the local directory extensions table **124** for the entry record **138** with the given pathname (Step S**250**). If the pathname is not found, this process fails and no True Name corresponding to the given pathname exists. Next, determine whether the local directory extensions table entry record **138** includes a True Name (Step S**252**), and if so, the mechanism's task is complete. Otherwise, determine whether the local directory extensions table entry record **138** identifies a directory (Step S**254**), and if so, freeze the directory (Step S**256**) (the primitive mechanism Freeze Directory is described below).

Otherwise, in step S**258**, assimilate the file (using the Assimilate Data Item primitive mechanism) defined by the File ID field to generate its True Name and store its True Name in the local directory extensions entry record. Then return the True Name identified by the local directory extensions table **124**.

5. Link Path to True Name

The mechanism to link a path to a True Name provides a way of creating a new directory entry record identifying an existing, assimilated file. This basic process may be used to copy, move, and rename files without a need to copy their contents. The mechanism to link a path to a True Name is now described with reference to FIG. **14**.

First, if desired, confirm that the True Name exists locally by searching for it in the True Name registry or local directory extensions table **135** (Step S**260**). Most uses of this mechanism will require this form of validation. Next, search for the path in the local directory extensions table **135** (Step S**262**). Confirm that the directory containing the file named in the path already exists (Step S**264**). If the named file itself exists, delete the File using the Delete True File operating system mechanism (see below) (Step S**268**).

Then, create an entry record in the local directory extensions with the specified path (Step S**270**) and update the entry record and other data structures as follows: fill in the True Name field of the entry with the specified True Name; increment the use count for the True File registry entry record **140** of the corresponding True Name; note whether the entry is a directory by reading the True File to see if it contains a tag (magic number) indicating that it represents a frozen directory (see also the description of the Freeze Directory primitive mechanism regarding the tag); and compute and set the other fields of the local directory extensions appropriately. For instance, search the region table **128** to identify the region of the path, and set the time of last access and time of last modification to the current time.

6. Realize True File from Location

This mechanism is used to try to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. This mechanism is now described with reference to FIG. **15**.

First, in step S**272**, determine whether the location specified is a processor. If it is determined that the location specified is a processor, then send a Request True File message (using the Request True File remote mechanism) to the remote processor and wait for a response (Step S**274**). If a negative response is received or no response is received after a timeout period, this mechanism fails. If a positive response is received, enter the True File returned in the True File registry **126** (Step S**276**). (If the file received was compressed, enter the True File ID in the compressed File ID field.)

If, on the other hand, it is determined in step S**272** that the location specified is not a processor, then, if necessary, request the user or operator to mount the indicated volume (Step S**278**). Then (Step S**280**) find the indicated file on the given volume and assimilate the file using the Assimilate Data Item primitive mechanism. If the volume does not contain a True File registry **126**, search the media inventory to find the path of the file on the volume. If no such file can be found, this mechanism fails.

At this point, whether or not the location is determined (in step S**272**) to be a processor, if desired, verify the True File (in step S**282**).

7. Locate Remote File

This mechanism allows a processor to locate a file or data item from a remote source of True Files, when a specific source is unknown or unavailable. A client processor system may ask one of several or many sources whether it can supply a data object with a given True Name. The steps to perform this mechanism are as follows (with reference to FIGS. **16**(*a*) and **16**(*b*).

The client processor **102** uses the source table **145** to select one or more source processors (Step S**284**). If no source processor can be found, the mechanism fails. Next, the client processor **102** broadcasts to the selected sources a request to locate the file with the given True Name using the Locate True File remote mechanism (Step S**286**). The request to locate may be augmented by asking to propagate this request to distant servers. The client processor then waits for one or more servers to respond positively (Step S**288**). After all servers respond negatively, or after a timeout period with no positive response, the mechanism repeats selection (Step S**284**) to attempt to identify alternative sources. If any selected source processor responds, its processor ID is the result of this mechanism. Store the processor ID in the source field of the True File registry entry record **140** of the given True Name (Step S**290**).

If the source location of the True Name is a different processor or medium than the destination (Step S**290***a*), perform the following steps:

(i) Look up the True File registry entry record **140** for the corresponding True Name, and add the source location ID to the list of sources for the True Name (Step S**290***b*); and

(ii) If the source is a publishing system, determine the expiration date on the publishing system for the True Name and add that to the list of sources. If the source is not a publishing system, send a message to reserve the True File on the source processor (Step S290*c*).

Source selection in step S284 may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost, and ignoring previously selected processors which did not respond in step S288.

8. Make True File Local

This mechanism is used when a True Name is known and a locally accessible copy of the corresponding file or data item is required. This mechanism makes it possible to actually read the data in a True File. The mechanism takes a True Name and returns when there is a local, accessible copy of the True File in the True File registry **126**. This mechanism is described here with reference to the flow chart of FIGS. **17**(*a*) and **17**(*b*).

First, look in the True File registry **126** for a True File entry record **140** for the corresponding True Name (Step S292). If no such entry is found this mechanism fails. If there is already a True File ID for the entry (Step S294), this mechanism's task is complete. If there is a compressed file ID for the entry (Step S296), decompress the file corresponding to the file ID (Step S298) and store the decompressed file ID in the entry (Step S300). This mechanism is then complete.

If there is no True File ID for the entry (Step S294) and there is no compressed file ID for the entry (Step S296), then continue searching for the requested file. At this time it may be necessary to notify the user that the system is searching for the requested file.

If there are one or more source IDs, then select an order in which to attempt to realize the source ID (Step S304). The order may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost. For each source in the order chosen, realize the True File from the source location (using the Realize True File from Location primitive mechanism), until the True File is realized (Step S306). If it is realized, continue with step S294. If no known source can realize the True File, use the Locate Remote File primitive mechanism to attempt to find the True File (Step S308). If this succeeds, realize the True File from the identified source location and continue with step S296.

9. Create Scratch File

A scratch copy of a file is required when a file is being created or is about to be modified. The scratch copy is stored in the file system of the underlying operating system. The scratch copy is eventually assimilated when the audit file record entry **146** is processed by the Process Audit File Entry primitive mechanism. This Create Scratch File mechanism requires a local directory extensions table entry record **138**. When it succeeds, the local directory extensions table entry record **138** contains the scratch file ID of a scratch file that is not contained in the True File registry **126** and that may be modified. This mechanism is now described with reference to FIGS. **18**(*a*) and **18**(*b*).

First determine whether the scratch file should be a copy of the existing True File (Step S310). If so, continue with step S312. Otherwise, determine whether the local directory extensions table entry record **138** identifies an existing True File (Step S316), and if so, delete the True File using the Delete True File primitive mechanism (Step S318). Then create a new, empty scratch file and store its scratch file ID in the local directory extensions table entry record **138** (Step S320). This mechanism is then complete.

If the local directory extensions table entry record **138** identifies a scratch file ID (Step S312), then the entry already has a scratch file, so this mechanism succeeds.

If the local directory extensions table entry record **138** identifies a True File (S316), and there is no True File ID for the True File (S312), then make the True File local using the Make True File Local primitive mechanism (Step S322). If there is still no True File ID, this mechanism fails.

There is now a local True File for this file. If the use count in the corresponding True File registry entry record **140** is one (Step S326), save the True File ID in the scratch file ID of the local directory extensions table entry record **138**, and remove the True File registry entry record **140** (Step S328). (This step makes the True File into a scratch file.) This mechanism's task is complete.

Otherwise, if the use count in the corresponding True File registry entry record **140** is not one (in step S326), copy the file with the given True File ID to a new scratch file, using the Read File OS mechanism and store its file ID in the local directory extensions table entry record **138** (Step S330), and reduce the use count for the True File by one. If there is insufficient space to make a copy, this mechanism fails.

10. Freeze Directory

This mechanism freezes a directory in order to calculate its True Name. Since the True Name of a directory is a function of the files within the directory, they must not change during the computation of the True Name of the directory. This mechanism requires the pathname of a directory to freeze. This mechanism is described with reference to FIGS. **19**(*a*) and **19**(*b*).

In step S332, add one to the global freeze lock. Then search the local directory extensions table **124** to find each subordinate data file and directory of the given directory, and freeze each subordinate directory found using the Freeze Directory primitive mechanism (Step S334). Assimilate each unassimilated data file in the directory using the Assimilate Data Item primitive mechanism (Step S336). Then create a data item which begins with a tag or marker (a "magic number") being a unique data item indicating that this data item is a frozen directory (Step S337). Then list the file name and True Name for each file in the current directory (Step S338). Record any additional information required, such as the type, time of last access and modification, and size (Step S340). Next, in step S342, using the Assimilate Data Item primitive mechanism, assimilate the data item created in step S338. The resulting True Name is the True Name of the frozen directory. Finally, subtract one from the global freeze lock (Step S344).

11. Expand Frozen Directory

This mechanism expands a frozen directory in a given location. It requires a given pathname into which to expand the directory, and the True Name of the directory and is described with reference to FIG. **20**.

First, in step S346, make the True File with the given True Name local using the Make True File Local primitive mechanism. Then read each directory entry in the local file created in step S346 (Step S348). For each such directory entry, do the following:

Create a full pathname using the given pathname and the file name of the entry (Step S350); and

link the created path to the True Name (Step S352) using the Link Path to True Name primitive mechanism.

12. Delete True File

This mechanism deletes a reference to a True Name. The underlying True File is not removed from the True File registry **126** unless there are no additional references to the file. With reference to FIG. **21**, this mechanism is performed as follows:

If the global freeze lock is on, wait until the global freeze lock is turned off (Step S354). This prevents deleting a True File while a directory which might refer to it is being frozen. Next, find the True File registry entry record **140** given the True Name (Step S356). If the reference count field of the True File registry **126** is greater than zero, subtract one from the reference count field (Step S358). If it is determined (in step S360) that the reference count field of the True File registry entry record **140** is zero, and if there are no dependent systems listed in the True File registry entry record **140**, then perform the following steps:

(i) If the True File is a simple data item, then delete the True File, otherwise,

(ii) (the True File is a compound data item) for each True Name in the data item, recursively delete the True File corresponding to the True Name (Step S362).

(iii) Remove the file indicated by the True File ID and compressed file ID from the True File registry **126**, and remove the True File registry entry record **140** (Step S364).

13. Process Audit File Entry

This mechanism performs tasks which are required to maintain information in the local directory extensions table **124** and True File registry **126**, but which can be delayed while the processor is busy doing more time-critical tasks. Entries **142** in the audit file **132** should be processed at a background priority as long as there are entries to be processed. With reference to FIG. **22**, the steps for processing an entry are as follows:

Determine the operation in the entry **142** currently being processed (Step S365). If the operation indicates that a file was created or written (Step S366), then assimilate the file using the Assimilate Data Item primitive mechanism (Step S368), use the New True File primitive mechanism to do additional desired processing (such as cache update, compression, and mirroring) (Step S369), and record the newly computed True Name for the file in the audit file record entry (Step S370).

Otherwise, if the entry being processed indicates that a compound data item or directory was copied (or deleted) (Step S376), then for each component True Name in the compound data item or directory, add (or subtract) one to the use count of the True File registry entry record **140** corresponding to the component True Name (Step S378).

In all cases, for each parent directory of the given file, update the size, time of last access, and time of last modification, according to the operation in the audit record (Step S379).

Note that the audit record is not removed after processing, but is retained for some reasonable period so that it may be used by the Synchronize Directory extended mechanism to allow a disconnected remote processor to update its representation of the local system.

14. Begin Grooming

This mechanism makes it possible to select a set of files for removal and determine the overall amount of space to be recovered. With reference to FIG. **23**, first verify that the global grooming lock is currently unlocked (Step S382). Then set the global grooming lock, set the total amount of space freed during grooming to zero and empty the list of files selected for deletion (Step S384). For each True File in the True File registry **126**, set the delete count to zero (Step S386).

15. Select For Removal

This grooming mechanism tentatively selects a pathname to allow its corresponding True File to be removed. With reference to FIG. **24**, first find the local directory extensions table entry record **138** corresponding to the given pathname (Step S388). Then find the True File registry entry record **140** corresponding to the True File name in the local directory extensions table entry record **138** (Step S390). Add one to the grooming delete count in the True File registry entry record **140** and add the pathname to a list of files selected for deletion (Step S392). If the grooming delete count of the True File registry entry record **140** is equal to the use count of the True File registry entry record **140**, and if the there are no entries in the dependency list of the True File registry entry record **140**, then add the size of the file indicated by the True File ID and or compressed file ID to the total amount of space freed during grooming (Step S394).

16. End Grooming

This grooming mechanism ends the grooming phase and removes all files selected for removal. With reference to FIG. **25**, for each file in the list of files selected for deletion, delete the file (Step S396) and then unlock the global grooming lock (Step S398).

Operating System Mechanisms

The next of the mechanisms provided by the present invention, operating system mechanisms, are now described.

The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

1. open File

A mechanism to open a file is described with reference to FIGS. **26**(*a*) and **26**(*b*). This mechanism is given as input a pathname and the type of access required for the file (for example, read, write, read/write, create, etc.) and produces either the File ID of the file to be opened or an indication that no file should be opened. The local directory extensions table record **138** and region table record **142** associated with the opened file are associated with the open file for later use in other processing functions which refer to the file, such as read, write, and close.

First, determine whether or not the named file exists locally by examining the local directory extensions table **124** to determine whether there is an entry corresponding to the given pathname (Step S400). If it is determined that the file name does not exist locally, then, using the access type, determine whether or not the file is being created by this opening process (Step S402). If the file is not being created, prohibit the open (Step S404). If the file is being created, create a zero-length scratch file using an entry in local directory extensions table **124** and produce the scratch file ID of this scratch file as the result (Step S406).

If, on the other hand, it is determined in step S400 that the file name does exist locally, then determine the region in which the file is located by searching the region table **128** to find the record **142** with the longest region path which is a prefix of the file pathname (Step S408). This record identifies the region of the specified file.

Next, determine using the access type, whether the file is being opened for writing or whether it is being opened only for reading (Step S410). If the file is being opened for reading only, then, if the file is a scratch file (Step S419), return the scratch File ID of the file (Step S424). Otherwise

get the True Name from the local directory extensions table **124** and make a local version of the True File associated with the True Name using the Make True File Local primitive mechanism, and then return the True File ID associated with the True Name (Step S**420**).

If the file is not being opened for reading only (Step S**410**), then, if it is determined by inspecting the region table entry record **142** that the file is in a read-only directory (Step S**416**), then prohibit the opening (Step S**422**).

If it is determined by inspecting the region table **128** that the file is in a cached region (Step S**423**), then send a Lock Cache message to the corresponding cache server, and wait for a return message (Step S**418**). If the return message says the file is already locked, prohibit the opening.

If the access type indicates that the file being modified is being rewritten completely (Step S**419**), so that the original data will not be required, then Delete the File using the Delete File OS mechanism (Step S**421**) and perform step S**406**. Otherwise, make a scratch copy of the file (Step S**417**) and produce the scratch file ID of the scratch file as the result (Step S**424**).

2. Close File

This mechanism takes as input the local directory extensions table entry record **138** of an open file and the data maintained for the open file. To close a file, add an entry to the audit file indicating the time and operation (create, read or write). The audit file processing (using the Process Audit File Entry primitive mechanism) will take care of assimilating the file and thereby updating the other records.

3. Read File

To read a file, a program must provide the offset and length of the data to be read, and the location of a buffer into which to copy the data read.

The file to be read from is identified by an open file descriptor which includes a File ID as computed by the Open File operating system mechanism defined above. The File ID may identify either a scratch file or a True File (or True File segment). If the File ID identifies a True File, it may be either a simple or a compound True File. Reading a file is accomplished by the following steps:

In the case where the File ID identifies a scratch file or a simple True File, use the read capabilities of the underlying operating system.

In the case where the File ID identifies a compound file, break the read operation into one or more read operations on component segments as follows:

A. Identify the segment(s) to be read by dividing the specified file offset and length each by the fixed size of a segment (a system dependent parameter), to determine the segment number and number of segments that must be read.

B. For each segment number computed above, do the following:

i. Read the compound True File index block to determine the True Name of the segment to be read.

ii. Use the Realize True File from Location primitive mechanism to make the True File segment available locally. (If that mechanism fails, the Read File mechanism fails).

iii. Determine the File ID of the True File specified by the True Name corresponding to this segment.

iv. Use the Read File mechanism (recursively) to read from this segment into the corresponding location in the specified buffer.

4. Write File

File writing uses the file ID and data management capabilities of the underlying operating system. File access (Make File Local described above) can be deferred until the first read or write.

5. Delete File or Directory

The process of deleting a file, for a given pathname, is described here with reference to FIGS. **27**(*a*) and **27**(*b*).

First, determine the local directory extensions table entry record **138** and region table entry record **142** for the file (Step S**422**). If the file has no local directory extensions table entry record **138** or is locked or is in a read-only region, prohibit the deletion.

Identify the corresponding True File given the True Name of the file being deleted using the True File registry **126** (Step S**424**). If the file has no True Name, (Step S**426**) then delete the scratch copy of the file based on its scratch file ID in the local directory extensions table **124** (Step S**427**), and continue with step S**428**.

If the file has a True Name and the True File's use count is one (Step S**429**), then delete the True File (Step S**430**), and continue with step S**428**.

If the file has a True Name and the True File's use count is greater than one, reduce its use count by one (Step S**431**). Then proceed with step S**428**.

In Step S**428**, delete the local directory extensions table entry record, and add an entry to the audit file **132** indicating the time and the operation performed (delete).

6. Copy File or Directory

A mechanism is provided to copy a file or directory given a source and destination processor and pathname. The Copy File mechanism does not actually copy the data in the file, only the True Name of the file. This mechanism is performed as follows:

(A) Given the source path, get the True Name from the path. If this step fails, the mechanism fails.

(B) Given the True Name and the destination path, link the destination path to the True Name.

(C) If the source and destination processors have different True File registries, find (or, if necessary, create) an entry for the True Name in the True File registry table **126** of the destination processor. Enter into the source ID field of this new entry the source processor identity.

(D) Add an entry to the audit file **132** indicating the time and operation performed (copy).

This mechanism addresses capability of the system to avoid copying data from a source location to a destination location when the destination already has the data. In addition, because of the ability to freeze a directory, this mechanism also addresses capability of the system immediately to make a copy of any collection of files, thereby to support an efficient version control mechanisms for groups of files.

7. Move File or Directory

A mechanism is described which moves (or renames) a file from a source path to a destination path. The move operation, like the copy operation, requires no actual transfer of data, and is performed as follows:

(A) Copy the file from the source path to the destination path.

(B) If the source path is different from the destination path, delete the source path.

8. Get File Status

This mechanism takes a file pathname and provides information about the pathname. First the local directory extensions table entry record **138** corresponding to the pathname given is found. If no such entry exists, then this mechanism fails, otherwise, gather information about the file and its corresponding True File from the local directory extensions table **124**. The information can include any information shown in the data structures, including the size, type, owner, True Name, sources, time of last access, time of

last modification, state (local or not, assimilated or not, compressed or not), use count, expiration date, and reservations.

9. Get Files in Directory

This mechanism enumerates the files in a directory. It is used (implicitly) whenever it is necessary to determine whether a file exists (is present) in a directory. For instance, it is implicitly used in the Open File, Delete File, Copy File or Directory, and Move File operating system mechanisms, because the files operated on are referred to by pathnames containing directory names. The mechanism works as follows:

The local directory extensions table **124** is searched for an entry **138** with the given directory pathname. If no such entry is found, or if the entry found is not a directory, then this mechanism fails.

If there is a corresponding True File field in the local directory extensions table record, then it is assumed that the True File represents a frozen directory. The Expand Frozen Directory primitive mechanism is used to expand the existing True File into directory entries in the local directory extensions table.

Finally, the local directory extensions table **124** is again searched, this time to find each directory subordinate to the given directory. The names found are provided as the result.

Remote Mechanisms

The remote mechanisms provided by the present invention are now described. Recall that remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation.

In a presently preferred embodiment, processors communicate with each other using a remote procedure call (RPC) style interface, running over one of any number of communication protocols such as IPX/SPX or TCP/IP. Each peer processor which provides access to its True File registry **126** or file regions, or which depends on another peer processor, provides a number of mechanisms which can be used by its peers.

The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

1. Locate True File

This mechanism allows a remote processor to determine whether the local processor contains a copy of a specific True File. The mechanism begins with a True Name and a flag indicating whether to forward requests for this file to other servers. This mechanism is now described with reference to FIG. **28**.

First determine if the True File is available locally or if there is some indication of where the True File is located (for example, in the Source IDs field). Look up the requested True Name in the True File registry **126** (Step S**432**).

If a True File registry entry record **140** is not found for this True Name (Step S**434**), and the flag indicates that the request is not to be forwarded (Step S**436**), respond negatively (Step S**438**). That is, respond to the effect that the True File is not available.

One the other hand, if a True File registry entry record **140** is not found (Step S**434**), and the flag indicates that the request for this True File is to be forwarded (Step S**436**), then forward a request for this True File to some other processors in the system (Step S**442**). If the source table for the current processor identifies one or more publishing servers which should have a copy of this True File, then forward the request to each of those publishing servers (Step S**436**).

If a True File registry entry record **140** is found for the required True File (Step S**434**), and if the entry includes a True File ID or Compressed File ID (Step S**440**), respond positively (Step S**444**). If the entry includes a True File ID then this provides the identity or disk location of the actual physical representation of the file or file segment required. If the entry include a Compressed File ID, then a compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.

If the True File registry entry record **140** is found (Step S**434**) but does not include a True File ID (the File ID is absent if the actual file is not currently present at the current location) (Step S**440**), and if the True File registry entry record **140** includes one or more source processors, and if the request can be forwarded, then forward the request for this True File to one or more of the source processors (Step S**444**).

2. Reserve True File

This mechanism allows a remote processor to indicate that it depends on the local processor for access to a specific True File. It takes a True Name as input. This mechanism is described here.

(A) Find the True File registry entry record **140** associated with the given True File. If no entry exists, reply negatively.

(B) If the True File registry entry record **140** does not include a True File ID or compressed File ID, and if the True File registry entry record **140** includes no source IDs for removable storage volumes, then this processor does not have access to a copy of the given file. Reply negatively.

(C) Add the ID of the sending processor to the list of dependent processors for the True File registry entry record **140**. Reply positively, with an indication of whether the reserved True File is on line or off line.

3. Request True File

This mechanism allows a remote processor to request a copy of a True File from the local processor. It requires a True Name and responds positively by sending a True File back to the requesting processor. The mechanism operates as follows:

(A) Find the True File registry entry record **140** associated with the given True Name. If there is no such True File registry entry record **140**, reply negatively.

(B) Make the True File local using the Make True File Local primitive mechanism. If this mechanism fails, the Request True File mechanism also fails.

(C) Send the local True File in either it is uncompressed or compressed form to the requesting remote processor. Note that if the True File is a compound file, the components are not sent.

(D) If the remote file is listed in the dependent process list of the True File registry entry record **140**, remove it.

4. Retire True File

This mechanism allows a remote processor to indicate that it no longer plans to maintain a copy of a given True File. An alternate source of the True File can be specified, if, for instance, the True File is being moved from one server

to another. It begins with a True Name, a requesting processor ID, and an optional alternate source. This mechanism operates as follows:

(A) Find a True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Find the requesting processor on the source list and, if it is there, remove it.

(C) If an alternate source is provided, add it to the source list for the True File registry entry record **140**.

(D) If the source list of the True File registry entry record **140** has no items in it, use the Locate Remote File primitive mechanism to search for another copy of the file. If it fails, raise a serious error.

5. Cancel Reservation

This mechanism allows a remote processor to indicate that it no longer requires access to a True File stored on the local processor. It begins with a True Name and a requesting processor ID and proceeds as follows:

(A) Find the True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Remove the identity of the requesting processor from the list of dependent processors, if it appears.

(C) If the list of dependent processors becomes zero and the use count is also zero, delete the True File.

6. Acquire True File

This mechanism allows a remote processor to insist that a local processor make a copy of a specified True File. It is used, for example, when a cache client wants to write through a new version of a file. The Acquire True File mechanism begins with a data item and an optional True Name for the data item and proceeds as follows:

(A) Confirm that the requesting processor has the right to require the local processor to acquire data items. If not, send a negative reply.

(B) Make a local copy of the data item transmitted by the remote processor.

(C) Assimilate the data item into the True File registry of the local processor.

(D) If a True Name was provided with the file, the True Name calculation can be avoided, or the mechanism can verify that the file received matches the True Name sent.

(E) Add an entry in the dependent processor list of the true file registry record indicating that the requesting processor depends on this copy of the given True File.

(F) Send a positive reply.

7. Lock Cache

This mechanism allows a remote cache client to lock a local file so that local users or other cache clients cannot change it while the remote processor is using it. The mechanism begins with a pathname and proceeds as follows:

(A) Find the local directory extensions table entry record **138** of the specified pathname. If no such entry exists, reply negatively.

(B) If an local directory extensions table entry record **138** exists and is already locked, reply negatively that the file is already locked.

(C) If an local directory extensions table entry record **138** exists and is not locked, lock the entry. Reply positively.

8. Update Cache

This mechanism allows a remote cache client to unlock a local file and update it with new contents. It begins with a pathname and a True Name. The file corresponding to the True Name must be accessible from the remote processor. This mechanism operates as follows:

Find the local directory extensions table entry record **138** corresponding to the given pathname. Reply negatively if no such entry exists or if the entry is not locked.

Link the given pathname to the given True Name using the Link Path to True Name primitive mechanism.

Unlock the local directory extensions table entry record **138** and return positively.

9. Check Expiration Date

Return current or new expiration date and possible alternative source to caller.

Background Processes and Mechanisms

The background processes and mechanisms provided by the present invention are now described. Recall that background mechanisms are intended to run occasionally and at a low priority to provide automated management capabilities with respect to the present invention.

The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links;
4. Verify Region; and
5. Groom Source List.

1. Mirror True File

This mechanism is used to ensure that files are available in alternate locations in mirror groups or archived on archival servers. The mechanism depends on application-specific migration/archival criteria (size, time since last access, number of copies required, number of existing alternative sources) which determine under what conditions a file should be moved. The Mirror True File mechanism operates as follows, using the True File specified, perform the following steps:

(A) Count the number of available locations of the True File by inspecting the source list of the True File registry entry record **140** for the True File. This step determines how many copies of the True File are available in the system.

(B) If the True File meets the specified migration criteria, select a mirror group server to which a copy of the file should be sent. Use the Acquire True File remote mechanism to copy the True File to the selected mirror group server. Add the identity of the selected system to the source list for the True File.

2. Groom Region

This mechanism is used to automatically free up space in a processor by deleting data items that may be available elsewhere. The mechanism depends on application-specific grooming criteria (for instance, a file may be removed if there is an alternate online source for it, it has not been accessed in a given number of days, and it is larger than a given size). This mechanism operates as follows:

Repeat the following steps (i) to (iii) with more aggressive grooming criteria until sufficient space is freed or until all grooming criteria have been exercised. Use grooming information to determine how much space has been freed. Recall that, while grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

(i) Begin Grooming (using the primitive mechanism).

(ii) For each pathname in the specified region, for the True File corresponding to the pathname, if the True File is present, has at least one alternative source, and meets application specific grooming criteria for the region, select the file for removal (using the primitive mechanism).

(iii) End Grooming (using the primitive mechanism).

If the region is used as a cache, no other processors are dependent on True Files to which it refers, and all such True Files are mirrored elsewhere. In this case, True Files can be removed with impunity. For a cache region, the grooming

criteria would ordinarily eliminate the least recently accessed True Files first. This is best done by sorting the True Files in the region by the most recent access time before performing step (ii) above. The application specific criteria would thus be to select for removal every True File encountered (beginning with the least recently used) until the required amount of free space is reached.

3. Check for Expired Links

This mechanism is used to determine whether dependencies on published files should be refreshed. The following steps describe the operation of this mechanism:

For each pathname in the specified region, for each True File corresponding to the pathname, perform the following step:

If the True File registry entry record **140** corresponding to the True File contains at least one source which is a publishing server, and if the expiration date on the dependency is past or close, then perform the following steps:

(A) Determine whether the True File registry entry record contains other sources which have not expired.

(B) Check the True Name expiration of the server. If the expiration date has been extended, or an alternate source is suggested, add the source to the True File registry entry record **140**.

(C) If no acceptable alternate source was found in steps (A) or (B) above, make a local copy of the True File.

(D) Remove the expired source.

4. Verify Region

This mechanism can be used to ensure that the data items in the True File registry **126** have not been damaged accidentally or maliciously. The operation of this mechanism is described by the following steps:

(A) Search the local directory extensions table **124** for each pathname in the specified region and then perform the following steps:

(i) Get the True File name corresponding to the pathname;

(ii) If the True File registry entry **140** for the True File does not have a True File ID or compressed file ID, ignore it.

(iii) Use the Verify True File mechanism (see extended mechanisms below) to confirm that the True File specified is correct.

5. Groom Source List

The source list in a True File entry should be groomed sometimes to ensure there are not too many mirror or archive copies. When a file is deleted or when a region definition or its mirror criteria are changed, it may be necessary to inspect the affected True Files to determine whether there are too many mirror copies. This can be done with the following steps:

For each affected True File,

(A) Search the local directory extensions table to find each region that refers to the True File.

(B) Create a set of "required sources", initially empty.

(C) For each region found,

(a) determine the mirroring criteria for that region,

(b) determine which sources for the True File satisfy the mirroring criteria, and

(c) add these sources to the set of required sources.

(D) For each source in the True File registry entry, if the source identifies a remote processor (as opposed to removable media), and if the source is not a publisher, and if the source is not in the set of required sources, then eliminate the source, and use the Cancel Reservation remote mechanism to eliminate the given processor from the list of dependent processors recorded at the remote processor identified by the source.

Extended Mechanisms

The extended mechanisms provided by the present invention are now described. Recall that extended mechanisms run within application programs over the operating system to provide solutions to specific problems and applications.

The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize Directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at Location;
7. Verify True File;
8. Track for Accounting Purposes; and
9. Track for Licensing Purposes.

1. Inventory Existing Directory

This mechanism determines the True Names of files in an existing on-line directory in the underlying operating system. One purpose of this mechanism is to install True Name mechanisms in an existing file system.

An effect of such an installation is to eliminate immediately all duplicate files from the file system being traversed. If several file systems are inventoried in a single True File registry, duplicates across the volumes are also eliminated.

(A) Traverse the underlying file system in the operating system. For each file encountered, excluding directories, perform the following:

(i) Assimilate the file encountered (using the Assimilate File primitive mechanism). This process computes its True Name and moves its data into the True File registry **126**.

(ii) Create a pathname consisting of the path to the volume directory and the relative path of the file on the media. Link this path to the computed True Name using the Link Path to True Name primitive mechanism.

2. Inventory Removable, Read-only Files

A system with access to removable, read-only media volumes (such as WORM disks and CD-ROMs) can create a usable inventory of the files on these disks without having to make online copies. These objects can then be used for archival purposes, directory overlays, or other needs. An operator must request that an inventory be created for such a volume.

This mechanism allows for maintaining inventories of the contents of files and data items on removable media, such as diskettes and CD-ROMS, independent of other properties of the files such as name, location, and date of creation.

The mechanism creates an online inventory of the files on one or more removable volumes, such as a floppy disk or CD-ROM, when the data on the volume is represented as a directory. The inventory service uses a True Name to identify each file, providing a way to locate the data independent of its name, date of creation, or location.

The inventory can be used for archival of data (making it possible to avoid archiving data when that data is already on a separate volume), for grooming (making it possible to delete infrequently accessed files if they can be retrieved from removable volumes), for version control (making it possible to generate a new version of a CD-ROM without having to copy the old version), and for other purposes.

The inventory is made by creating a volume directory in the media inventory in which each file named identifies the data item on the volume being inventoried. Data items are not copied from the removable volume during the inventory process.

An operator must request that an inventory be created for a specific volume. Once created, the volume directory can be frozen or copied like any other directory. Data items from either the physical volume or the volume directory can be accessed using the Open File operating system mechanism which will cause them to be read from the physical volume using the Realize True File from Location primitive mechanism.

To create an inventory the following steps are taken:

(A) A volume directory in the media inventory is created to correspond to the volume being inventoried. Its contextual name identifies the specific volume.

(B) A source table entry **144** for the volume is created in the source table **130**. This entry **144** identifies the physical source volume and the volume directory created in step (A).

(C) The filesystem on the volume is traversed. For each file encountered, excluding directories, the following steps are taken:

(i) The True Name of the file is computed. An entry is created in the True Name registry **124**, including the True Name of the file using the primitive mechanism. The source field of the True Name registry entry **140** identifies the source table entry **144**.

(ii) A pathname is created consisting of the path to the volume directory and the relative path of the file on the media. This path is linked to the computed True Name using Link Path to True Name primitive mechanism.

(D) After all files have been inventoried, the volume directory is frozen. The volume directory serves as a table of contents for the volume. It can be copied using the Copy File or Directory primitive mechanism to create an "overlay" directory which can then be modified, making it possible to edit a virtual copy of a read-only medium.

3. Synchronize Directories

Given two versions of a directory derived from the same starting point, this mechanism creates a new, synchronized version which includes the changes from each. Where a file is changed in both versions, this mechanism provides a user exit for handling the discrepancy. By using True Names, comparisons are instantaneous, and no copies of files are necessary.

This mechanism lets a local processor synchronize a directory to account for changes made at a remote processor. Its purpose is to bring a local copy of a directory up to date after a period of no communication between the local and remote processor. Such a period might occur if the local processor were a mobile processor detached from its server, or if two distant processors were run independently and updated nightly.

An advantage of the described synchronization process is that it does not depend on synchronizing the clocks of the local and remote processors. However, it does require that the local processor track its position in the remote processor's audit file.

This mechanism does not resolve changes made simultaneously to the same file at several sites. If that occurs, an external resolution mechanism such as, for example, operator intervention, is required.

The mechanism takes as input a start time, a local directory pathname, a remote processor name, and a remote directory pathname name, and it operates by the following steps:

(A) Request a copy of the audit file **132** from the remote processor using the Request True File remote mechanism.

(B) For each entry **146** in the audit file **132** after the start time, if the entry indicates a change to a file in the remote directory, perform the following steps:

(i) Compute the pathname of the corresponding file in the local directory. Determine the True Name of the corresponding file.

(ii) If the True Name of the local file is the same as the old True Name in the audit file, or if there is no local file and the audit entry indicates a new file is being created, link the new True Name in the audit file to the local pathname using the Link Path to True Name primitive mechanism.

(iii) Otherwise, note that there is a problem with the synchronization by sending a message to the operator or to a problem resolution program, indicating the local pathname, remote pathname, remote processor, and time of change.

(C) After synchronization is complete, record the time of the final change. This time is to be used as the new start time the next time this directory is synchronized with the same remote processor.

4. Publish Region

The publish region mechanism allows a processor to offer the files in a region to any client processors for a limited period of time.

The purpose of the service is to eliminate any need for client processors to make reservations with the publishing processor. This in turn makes it possible for the publishing processor to service a much larger number of clients.

When a region is published, an expiration date is defined for all files in the region, and is propagated into the publishing system's True File registry entry record **140** for each file.

When a remote file is copied, for instance using the Copy File operating system mechanism, the expiration date is copied into the source field of the client's True File registry entry record **140**. When the source is a publishing system, no dependency need be created.

The client processor must occasionally and in background, check for expired links, to make sure it still has access to these files. This is described in the background mechanism Check for Expired Links.

5. Retire Directory

This mechanism makes it possible to eliminate safely the True Files in a directory, or at least dependencies on them, after ensuring that any client processors depending on those files remove their dependencies. The files in the directory are not actually deleted by this process. The directory can be deleted with the Delete File operating system mechanism.

The mechanism takes the pathname of a given directory, and optionally, the identification of a preferred alternate source processor for clients to use. The mechanism performs the following steps:

(A) Traverse the directory. For each file in the directory, perform the following steps:

(i) Get the True Name of the file from its path and find the True File registry entry **140** associated with the True Name.

(ii) Determine an alternate source for the True File. If the source IDs field of the TFR entry includes the preferred alternate source, that is the alternate source. If it does not, but includes some other source, that is the alternate source. If it contains no alternate sources, there is no alternate source.

(iii) For each dependent processor in the True File registry entry **140**, ask that processor to retire the True File, specifying an alternate source if one was determined, using the remote mechanism.

6. Realize Directory at Location

This mechanism allows the user or operating system to force copies of files from some source location to the True File registry **126** at a given location. The purpose of the mechanism is to ensure that files are accessible in the event the source location becomes inaccessible. This can happen for instance if the source or given location are on mobile computers, or are on removable media, or if the network connection to the source is expected to become unavailable, or if the source is being retired.

This mechanism is provided in the following steps for each file in the given directory, with the exception of subdirectories:

(A) Get the local directory extensions table entry record **138** given the pathname of the file. Get the True Name of the local directory extensions table entry record **138**. This service assimilates the file if it has not already been assimilated.

(B) Realize the corresponding True File at the given location. This service causes it to be copied to the given location from a remote system or removable media.

7. Verify True File

This mechanism is used to verify that the data item in a True File registry **126** is indeed the correct data item given its True Name. Its purpose is to guard against device errors, malicious changes, or other problems.

If an error is found, the system has the ability to "heal" itself by finding another source for the True File with the given name. It may also be desirable to verify that the error has not propagated to other systems, and to log the problem or indicate it to the computer operator. These details are not described here.

To verify a data item that is not in a True File registry **126**, use the Calculate True Name primitive mechanism described above.

The basic mechanism begins with a True Name, and operates in the following steps:

(A) Find the True File registry entry record **140** corresponding to the given True Name.

(B) If there is a True File ID for the True File registry entry record **140** then use it. Otherwise, indicate that no file exists to verify.

(C) Calculate the True Name of the data item given the file ID of the data item.

(D) Confirm that the calculated True Name is equal to the given True Name.

(E) If the True Names are not equal, there is an error in the True File registry **126**. Remove the True File ID from the True File registry entry record **140** and place it somewhere else. Indicate that the True File registry entry record **140** contained an error.

8. Track for Accounting Purposes

This mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another. The mechanism can be used as a basis for a value-based accounting system in which charges are based on the identity of the data stored or transmitted, rather than simply on the number of bits.

This mechanism allows the system to track possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. True names make it possible to identify each file briefly yet uniquely for this purpose.

Tracking the identities of files requires maintaining an accounting log **134** and processing it for accounting or billing purposes. The mechanism operates in the following steps:

(A) Note every time a file is created or deleted, for instance by monitoring audit entries in the Process Audit File Entry primitive mechanism. When such an event is encountered, create an entry **148** in the accounting log **134** that shows the responsible party and the identity of the file created or deleted.

(B) Every time a file is transmitted, for instance when a file is copied with a Request True File remote mechanism or an Acquire True File remote mechanism, create an entry in the accounting log **134** that shows the responsible party, the identity of the file, and the source and destination processors.

(C) Occasionally run an accounting program to process the accounting log **134**, distributing the events to the account records of each responsible party. The account records can eventually be summarized for billing purposes.

9. Track for Licensing Purposes

This mechanism ensures that licensed files are not used by unauthorized parties. The True Name provides a safe way to identify licensed material. This service allows proof of possession of specific files according to their contents without disclosing their contents.

Enforcing use of valid licenses can be active (for example, by refusing to provide access to a file without authorization) or passive (for example, by creating a report of users who do not have proper authorization).

One possible way to perform license validation is to perform occasional audits of employee systems. The service described herein relies on True Names to support such an audit, as in the following steps:

(A) For each licensed product, record in the license table **136** the True Name of key files in the product (that is, files which are required in order to use the product, and which do not occur in other products) Typically, for a software product, this would include the main executable image and perhaps other major files such as clip-art, scripts, or online help. Also record the identity of each system which is authorized to have a copy of the file.

(B) Occasionally, compare the contents of each user processor against the license table **136**. For each True Name in the license table do the following:

(i) Unless the user processor is authorized to have a copy of the file, confirm that the user processor does not have a copy of the file using the Locate True File mechanism.

(ii) If the user processor is found to have a file that it is not authorized to have, record the user processor and True Name in a license violation table.

The System in Operation

Given the mechanisms described above, the operation of a typical DP system employing these mechanisms is now described in order to demonstrate how the present invention meets its requirements and capabilities.

In operation, data items (for example, files, database records, messages, data segments, data blocks, directories, instances of object classes, and the like) in a DP system employing the present invention are identified by substantially unique identifiers (True Names), the identifiers depending on all of the data in the data items and only on the data in the data items. The primitive mechanisms Calculate True Name and Assimilate Data Item support this property. For any given data item, using the Calculate True Name primitive mechanism, a substantially unique identifier or True Name for that data item can be determined.

Further, in operation of a DP system incorporating the present invention, multiple copies of data items are avoided (unless they are required for some reason such as backups or

mirror copies in a fault-tolerant system). Multiple copies of data items are avoided even when multiple names refer to the same data item. The primitive mechanisms Assimilate Data Items and New True File support this property. Using the Assimilate Data Item primitive mechanism, if a data item already exists in the system, as indicated by an entry in the True File registry **126**, this existence will be discovered by this mechanism, and the duplicate data item (the new data item) will be eliminated (or not added). Thus, for example, if a data file is being copied onto a system from a floppy disk, if, based on the True Name of the data file, it is determined that the data file already exists in the system (by the same or some other name), then the duplicate copy will not be installed. If the data item was being installed on the system by some name other than its current name, then, using the Link Path to True Name primitive mechanism, the other (or new) name can be linked to the already existing data item.

In general, the mechanisms of the present invention operate in such a way as to avoid recreating an actual data item at a location when a copy of that data item is already present at that location. In the case of a copy from a floppy disk, the data item (file) may have to be copied (into a scratch file) before it can be determined that it is a duplicate. This is because only one processor is involved. On the other hand, in a multiprocessor environment or DP system, each processor has a record of the True Names of the data items on that processor. When a data item is to be copied to another location (another processor) in the DP system, all that is necessary is to examine the True Name of the data item prior to the copying. If a data item with the same True Name already exists at the destination location (processor), then there is no need to copy the data item. Note that if a data item which already exists locally at a destination location is still copied to the destination location (for example, because the remote system did not have a True Name for the data item or because it arrives as a stream of un-named data), the Assimilate Data Item primitive mechanism will prevent multiple copies of the data item from being created.

Since the True Name of a large data item (a compound data item) is derived from and based on the True Names of components of the data item, copying of an entire data item can be avoided. Since some (or all) of the components of a large data item may already be present at a destination location, only those components which are not present there need be copied. This property derives from the manner in which True Names are determined.

When a file is copied by the Copy File or Directory operating system mechanism, only the True Name of the file is actually replicated.

When a file is opened (using the Open File operating system mechanism), it uses the Make True File Local primitive mechanism (either directly or indirectly through the Create Scratch File primitive mechanism) to create a local copy of the file. The Open File operating system mechanism uses the Make True File Local primitive mechanism, which uses the Realize True File from Location primitive mechanism, which, in turn uses the Request True File remote mechanism.

The Request True File remote mechanism copies only a single data item from one processor to another. If the data item is a compound file, its component segments are not copied, only the indirect block is copied. The segments are copied only when they are read (or otherwise needed).

The Read File operating system mechanism actually reads data. The Read File mechanism is aware of compound files and indirect blocks, and it uses the Realize True File from Location primitive mechanism to make sure that component segments are locally available, and then uses the operating system file mechanisms to read data from the local file.

Thus, when a compound file is copied from a remote system, only its True Name is copied. When it is opened, only its indirect block is copied. When the corresponding file is read, the required component segments are realized and therefore copied.

In operation data items can be accessed by reference to their identities (True Names) independent of their present location. The actual data item or True File corresponding to a given data identifier or True Name may reside anywhere in the system (that is, locally, remotely, offline, etc). If a required True File is present locally, then the data in the file can be accessed. If the data item is not present locally, there are a number of ways in which it can be obtained from wherever it is present. Using the source IDs field of the True File registry table, the location(s) of copies of the True File corresponding to a given True Name can be determined. The Realize True File from Location primitive mechanism tries to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. If, on the other hand, for some reason it is not known where there is a copy of the True File, or if the processors identified in the source IDs field do not respond with the required True File, the processor requiring the data item can make a general request for the data item using the Request True File remote mechanism from all processors in the system that it can contact.

As a result, the system provides transparent access to any data item by reference to its data identity, and independent of its present location.

In operation, data items in the system can be verified and have their integrity checked. This is from the manner in which True Names are determined. This can be used for security purposes, for instance, to check for viruses and to verify that data retrieved from another location is the desired and requested data. For example, the system might store the True Names of all executable applications on the system and then periodically redetermine the True Names of each of these applications to ensure that they match the stored True Names. Any change in a True Name potentially signals corruption in the system and can be further investigated. The Verify Region background mechanism and the Verify True File extended mechanisms provide direct support for this mode of operation. The Verify Region mechanism is used to ensure that the data items in the True File registry have not been damaged accidentally or maliciously. The Verify True File mechanism verifies that a data item in a True File registry is indeed the correct data item given its True Name.

Once a processor has determined where (that is, at which other processor or location) a copy of a data item is in the DP system, that processor might need that other processor or location to keep a copy of that data item. For example, a processor might want to delete local copies of data items to make space available locally while knowing that it can rely on retrieving the data from somewhere else when needed. To this end the system allows a processor to Reserve (and cancel the reservation of) True Files at remote locations (using the remote mechanism). In this way the remote locations are put on notice that another location is relying on the presence of the True File at their location.

A DP system employing the present invention can be made into a fault-tolerant system by providing a certain amount of redundancy of data items at multiple locations in the system. Using the Acquire True File and Reserve True File remote mechanisms, a particular processor can imple-

ment its own form of fault-tolerance by copying data items to other processors and then reserving them there. However, the system also provides the Mirror True File background mechanism to mirror (make copies) of the True File available elsewhere in the system. Any degree of redundancy (limited by the number of processors or locations in the system) can be implemented. As a result, this invention maintains a desired degree or level of redundancy in a network of processors, to protect against failure of any particular processor by ensuring that multiple copies of data items exist at different locations.

The data structures used to implement various features and mechanisms of this invention store a variety of useful information which can be used, in conjunction with the various mechanisms, to implement storage schemes and policies in a DP system employing the invention. For example, the size, age and location of a data item (or of groups of data items) is provided. This information can be used to decide how the data items should be treated. For example, a processor may implement a policy of deleting local copies of all data items over a certain age if other copies of those data items are present elsewhere in the system. The age (or variations on the age) can be determined using the time of last access or modification in the local directory extensions table, and the presence of other copies of the data item can be determined either from the Safe Flag or the source IDs, or by checking which other processors in the system have copies of the data item and then reserving at least one of those copies.

In operation, the system can keep track of data items regardless of how those items are named by users (or regardless of whether the data items even have names). The system can also track data items that have different names (in different or the same location) as well as different data items that have the same name. Since a data item is identified by the data in the item, without regard for the context of the data, the problems of inconsistent naming in a DP system are overcome.

In operation, the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of these data items. True Names are globally unique identifiers which can be published simply by copying them. For example, a user might create a textual representation of a file on system A with True Name N (for instance as a hexadecimal string), and post it on a computer bulletin board. Another user on system B could create a directory entry F for this True Name N by using the Link Path to True Name primitive mechanism. (Alternatively, an application could be developed which hides the True Name from the users, but provides the same public transfer service.)

When a program on system B attempts to open pathname F linked to True Name N, the Locate Remote File primitive mechanism would be used, and would use the Locate True File remote mechanism to search for True Name N on one or more remote processors, such as system A. If system B has access to system A, it would be able to realize the True File (using the Realize True File from Location primitive mechanism) and use it locally. Alternatively, system B could find True Name N by accessing any publicly available True Name server, if the server could eventually forward the request to system A.

Clients of a local server can indicate that they depend on a given True File (using the Reserve True File remote mechanism) so that the True File is not deleted from the server registry as long as some client requires access to it. (The Retire True File remote mechanism is used to indicate that a client no longer needs a given True File.)

A publishing server, on the other hand, may want to provide access to many clients, and possibly anonymous ones, without incurring the overhead of tracking dependencies for each client. Therefore, a public server can provide expiration dates for True Files in its registry. This allows client systems to safely maintain references to a True File on the public server. The Check For Expired Links background mechanism allows the client of a publishing server to occasionally confirm that its dependencies on the publishing server are safe.

In a variation of this aspect of the invention, a processor that is newly connected (or reconnected after some absence) to the system can obtain a current version of all (or of needed) data in the system by requesting it from a server processor. Any such processor can send a request to update or resynchronize all of its directories (starting at a root directory), simply by using the Synchronize Directories extended mechanism on the needed directories.

Using the accounting log or some other user provided mechanism, a user can prove the existence of certain data items at certain times. By publishing (in a public place) a list of all True Names in the system on a given day (or at some given time), a user can later refer back to that list to show that a particular data item was present in the system at the time that list was published. Such a mechanism is useful in tracking, for example, laboratory notebooks or the like to prove dates of conception of inventions. Such a mechanism also permits proof of possession of a data item at a particular date and time.

The accounting log file can also track the use of specific data items and files by content for accounting purposes. For instance, an information utility company can determine the data identities of data items that are stored and transmitted through its computer systems, and use these identities to provide bills to its customers based on the identities of the data items being transmitted (as defined by the substantially unique identifier). The assignment of prices for storing and transmitting specific True Files would be made by the information utility and/or its data suppliers; this information would be joined periodically with the information in the accounting log file to produce customer statements.

Backing up data items in a DP system employing the present invention can be done based on the True Names of the data items. By tracking backups using True Names, duplication in the backups is prevented. In operation, the system maintains a backup record of data identifiers of data items already backed up, and invokes the Copy File or Directory operating system mechanism to copy only those data items whose data identifiers are not recorded in the backup record. Once a data item has been backed up, it can be restored by retrieving it from its backup location, based on the identifier of the data item. Using the backup record produced by the backup to identify the data item, the data item can be obtained using, for example, the Make True File Local primitive mechanism.

In operation, the system can be used to cache data items from a server, so that only the most recently accessed data items need be retained. To operate in this way, a cache client is configured to have a local registry (its cache) with a remote Local Directory Extensions table (from the cache server). Whenever a file is opened (or read), the Local Directory Extensions table is used to identify the True Name, and the Make True File Local primitive mechanism inspects the local registry. When the local registry already has a copy, the file is already cached. Otherwise, the Locate True File remote mechanism is used to get a copy of the file. This mechanism consults the cache server and uses the

Request True File remote mechanism to make a local copy, effectively loading the cache.

The Groom Cache background mechanism flushes the cache, removing the least-recently-used files from the cache client's True File registry. While a file is being modified on a cache client, the Lock Cache and Update Cache remote mechanisms prevent other clients from trying to modify the same file.

In operation, when the system is being used to cache data items, the problems of maintaining cache consistency are avoided.

To access a cache and to fill it from its server, a key is required to identify the data item desired. Ordinarily, the key is a name or address (in this case, it would be the pathname of a file). If the data associated with such a key is changed, the client's cache becomes inconsistent; when the cache client refers to that name, it will retrieve the wrong data. In order to maintain cache consistency it is necessary to notify every client immediately whenever a change occurs on the server.

By using an embodiment of the present invention, the cache key uniquely identifies the data it represents. When the data associated with a name changes, the key itself changes. Thus, when a cache client wishes to access the modified data associated with a given file name, it will use a new key (the True Name of the new file) rather than the key to the old file contents in its cache. The client will always request the correct data, and the old data in its cache will be eventually aged and flushed by the Groom Cache background mechanism.

Because it is not necessary to immediately notify clients when changes on the cache server occur, the present invention makes it possible for a single server to support a much larger number of clients than is otherwise possible.

In operation, the system automatically archives data items as they are created or modified. After a file is created or modified, the Close File operating system mechanism creates an audit file record, which is eventually processed by the Process Audit File Entry primitive mechanism. This mechanism uses the New True File primitive mechanism for any file which is newly created, which in turn uses the Mirror True File background mechanism if the True File is in a mirrored or archived region. This mechanism causes one or more copies of the new file to be made on remote processors.

In operation, the system can efficiently record and preserve any collection of data items. The Freeze Directory primitive mechanism creates a True File which identifies all of the files in the directory and its subordinates. Because this True File includes the True Names of its constituents, it represents the exact contents of the directory tree at the time it was frozen. The frozen directory can be copied with its components preserved.

The Acquire True File remote mechanism (used in mirroring and archiving) preserves the directory tree structure by ensuring that all of the component segments and True Files in a compound data item are actually copied to a remote system. Of course, no transfer is necessary for data items already in the registry of the remote system.

In operation, the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items.

The Freeze Directory primitive mechanism is used to create a collection of data items. The constituent files and segments referred to by the frozen directory are maintained in the registry, without any need to make copies of the constituents each time the directory is frozen.

Whenever a pathname is traversed, the Get Files in Directory operating system mechanism is used, and when it encounters a frozen directory, it uses the Expand Frozen Directory primitive mechanism.

A frozen directory can be copied from one pathname to another efficiently, merely by copying its True Name. The Copy File operating system mechanism is used to copy a frozen directory.

Thus it is possible to efficiently create copies of different versions of a directory, thereby creating a record of its history (hence a version control system).

In operation, the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM. The inventory is independent of other properties of the data items such as their name, location, and date of creation.

The Inventory Existing Directory extended mechanism provides a way to create True File Registry entries for all of the files in a directory. One use of this inventory is as a way to pre-load a True File registry with backup record information. Those files in the registry (such as previously installed software) which are on the volumes inventoried need not be backed up onto other volumes.

The Inventory Removable, Read-only Files extended mechanism not only determines the True Names for the files on the medium, but also records directory entries for each file in a frozen directory structure. By copying and modifying this directory, it is possible to create an on line patch, or small modification of an existing read-only file. For example, it is possible to create an online representation of a modified CD-ROM, such that the unmodified files are actually on the CD-ROM, and only the modified files are online.

In operation, the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. Using the Track for Accounting Purposes extended mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another.

True Names in Relational and Object-Oriented Databases

Although the preferred embodiment of this invention has been presented in the context of a file system, the invention of True Names would be equally valuable in a relational or object-oriented database. A relational or object-oriented database system using True Names would have similar benefits to those of the file system employing the invention. For instance, such a database would permit efficient elimination of duplicate records, support a cache for records, simplify the process of maintaining cache consistency, provide location-independent access to records, maintain archives and histories of records, and synchronize with distant or disconnected systems or databases.

The mechanisms described above can be easily modified to serve in such a database environment. The True Name registry would be used as a repository of database records. All references to records would be via the True Name of the record. (The Local Directory Extensions table is an example of a primary index that uses the True Name as the unique identifier of the desired records.)

In such a database, the operations of inserting, updating, and deleting records would be implemented by first assimilating records into the registry, and then updating a primary key index to map the key of the record to its contents by using the True Name as a pointer to the contents.

The mechanisms described in the preferred embodiment, or similar mechanisms, would be employed in such a

system. These mechanisms could include, for example, the mechanisms for calculating true names, assimilating, locating, realizing, deleting, copying, and moving True Files, for mirroring True Files, for maintaining a cache of True Files, for grooming True Files, and other mechanisms based on the use of substantially unique identifiers.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiments, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

What is claimed is:

1. In a data processing system, an apparatus comprising:

identity means for determining, for any of a plurality of data items present in the system, a substantially unique identifier, the identifier being determined using and depending on all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier; and

existence means for determining whether a particular data item is present in the system, by examining the identifiers of the plurality of data items.

2. An apparatus as in claim **1**, further comprising:

local existence means for determining whether an instance of a particular data item is present at a particular location in the system, based on the identifier of the data item.

3. An apparatus as in claim **2**, wherein each location contains a distinct plurality of data items, and wherein said local existence means determines whether a particular data item is present at a particular location in the system by examining the identifiers of the plurality of data items at said particular location in the system.

4. An apparatus as in claim **2**, further comprising:

data associating means for making and maintaining, for a data item in the system, an association between the data item and the identifier of the data item; and

access means for accessing a particular data item using the identifier of the data item.

5. An apparatus as in claim **2**, further comprising:

duplication means for copying a data item from a source to a destination in the data processing system, by providing said destination with the data item only if it is determined using the data identifier that the data item is not present at the destination.

6. An apparatus as in claim **4**, further comprising:

assimilation means for assimilating a new data item into the system, said assimilation means invoking said identity means to determine the identifier of the new data item and invoking said data associating means to associate the new data item with its identifier.

7. An apparatus as in claim **4**, further comprising:

duplication means for duplicating a data item from a source location to a destination location in the data processing system, based on the identifier of the data item, said duplication means invoking said local existence means to determine whether an instance of the data item is present at the destination location, and invoking said access means to provide said destination with the data item only if said local existence means determines that no instance of the data item is present at the destination.

8. An apparatus as in claim **7**, further comprising:

backup means for making copies of data items in the system, said backup means maintaining a backup record of identifiers of data items backed up, and invoking duplication means to copy only those data items whose data identifiers are not recorded in the backup record.

9. An apparatus as in claim **8**, further comprising:

recovery means for retrieving a data item previously backed up by said backup means, based on the identifier of the data item, said recovery means using the backup record to identify the data item, and invoking access means to retrieve the data item.

10. An apparatus as in claim **2**, wherein a location is a computer among a network of computers, the apparatus further comprising:

remote existence means for determining whether a data item is present at a remote location in the system from a current location in the system, based on the identifier of the data item, said remote location using local existence means at the remote location to determine whether the data item is present at the remote location, and providing the current location with an indication of the presence of the data item at the remote location.

11. An apparatus as in claim **4**, wherein a location is a computer among a network of computers, the apparatus further comprising:

requesting means for requesting a data item at a current location in the system from a remote location in the system, based on the identifier of the data item, said remote location using access means at the remote location to obtain the data item and to send it to the current location if it is present.

12. An apparatus as in claim **1**, further comprising:

context means for making and maintaining a context association between at least one contextual name of a data item in the system and the identifier of the data item; and

referencing means for obtaining the identifier of a data item in the system given a contextual name for the data item, using said context association.

13. An apparatus as in claim **12**, further comprising:

assignment means for assigning a data item to a contextual name, invoking said identity means to determine the identifier of the data item, and invoking said context means to make or modify the context association between the contextual name of the data item and the identifier of the data item.

14. An apparatus as in claim **12**, further comprising:

data associating means for making and maintaining, for a data item in the system, an association between the data item and the identifier of the data item;

access means for accessing a particular data item using the identifier of the particular data item; and

contextual name access means for accessing a data item in the system for a given context name of the data item, determining the data identifier associated with the given context name, and invoking said access means to access the data item using the data identifier.

15. An apparatus as in claim **11**, further comprising:

transparent access means for accessing a data item from one of several locations, using the identifier of the data item, said transparent access means invoking said local existence means to determine if the particular data item is present at the current location, and, in the case when the particular data item is not present at the current location, invoking said requesting means to obtain the data item from a remote location.

16. An apparatus as in claim 15, further comprising:

identifier copy means for copying an identifier of a data item from a source location to a destination location.

17. An apparatus as in claim 15, further comprising:

context means for making and maintaining a context association between a contextual name of a data item in the system and the identifier of the data item;

context copy means for copying a data item from a source location to a destination location, given the contextual name of the data item, by copying only the context association between the contextual identifier and the data identifier from the source location to the destination location; and

transparent referencing means for obtaining a data item from one of several locations the system given a contextual name for the data item, said transparent referencing means invoking said context association to determine the data identifier of a data item given a contextual name, and invoking said transparent access means to access the data item from one of several locations given the identifier of the data item.

18. An apparatus as in claim 1, wherein at least some of said data items are compound data items, each compound data item including at least some component data items in a fixed sequence, and wherein the identity means determines the identifier of a compound data item based on each component data item of the compound data item.

19. An apparatus as in claim 18, wherein said compound data items are files and said component data items are segments, and wherein the identity means determines the identifier of a file based on the identifier of each data segment of the file.

20. An apparatus as in claim 18, wherein said compound data items are directories and said component data items are files or subordinate directories, and wherein the identity means determines the identifier of a given directory based on each file and subordinate directory within the given directory.

21. An apparatus as in claim 11, further comprising:

means for advertising a data item from a location in the system to at least one other location in the system, said means for advertising providing each of said at least one other location with the data identifier of the data item, and providing the data item to only those locations of said other locations that request said data item in response to said providing.

22. An apparatus as in claim 18, further comprising:

local existence means for determining whether a particular data item is present at a particular location in the system, based on the identifier of the data item; and

compound copy means for copying a data item from a source to a destination in the data processing system, said compound copy means invoking said local existence means to determine whether the data item is present at the destination, and to determine, when the data item is a compound data item, whether the component data items of the compound data item are present at the destination, and providing said destination with the data item only if said local existence means determines that the data item is not present at the destination, and providing said destination with each component data item only if said local existence means determines that the component data item is not present at the destination.

23. An apparatus as in claim 11, further comprising:

means for verifying the integrity of a data item obtained from the requesting means in response to providing the requesting with a particular data identifier, to confirm that the data item obtained from the requesting means is the same data item as the data item requested, the verifying means invoking the identity means to determine the data identifier of the obtained data item, and comparing the determined data identifier with the particular data identifier to verify the obtained data item.

24. An apparatus as in claim 2, wherein a location is at least one of a storage location and a processing location, and wherein a storage location is at least one of a data storage device and a data storage volume, and wherein a processing location is at least one of a data processor and a computer.

25. An apparatus as in claim 3, wherein at least some of said data items are compound data items, each compound data item including at least some component data items in a fixed sequence, and wherein the identity means determines the identifier of a compound data item based on the identifier of each component data item of the compound data item.

26. An apparatus as in claim 3, further comprising:

context associating means for making and maintaining a context association, for any data item in the system, between the identifier of the data item and at least one contextual name of the data item at a particular location in the system;

means for obtaining the identifier of a data item in the system given a contextual name for the data item at a particular location in the system; and

logical copy means for associating the data identifier corresponding to a contextual name at a source location with a contextual name at a destination location in the data processing system.

27. An apparatus as in claim 25, wherein said compound data items are files and said component data items are segments, and wherein the identity means determines the identifier of a file based on the identifier of each data segment of the file.

28. An apparatus as in claim 25, further comprising:

compound copy means for copying a data item from a source location to a destination location in the data processing system, said compound copy means invoking said local existence means to determine whether the data item is present at the destination, and to determine, when the data item is a compound data item, whether the component data items of the compound data item are present at the destination, and providing said destination with the data item only if said local existence means determines that the data item is not present at the destination, and providing said destination with each component data item only if said local existence means determines that the component data item is not present at the destination.

29. An apparatus as in any of claims 1–28, wherein a data item is at least one of a file, a database record, a message, a data segment, a data block, a directory, and an instance an object class.

30. A method of identifying a data item present in a data processing system for subsequent access to the data item, the method comprising:

determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier; and

accessing a data item in the system using the identifier of the data item.

31. A method as in claim 30, further comprising:

making and maintaining, for a plurality of data items present in the system, an association between each of the data items and the identifier of each of the data items, wherein said accessing a data item accesses a data item via the association.

32. A method as in claim 31, further comprising:

assimilating a new data item into the system, by determining the identifier of the new data item and associating the new data item with its identifier.

33. A method for duplicating a given data item present at a source location to a destination location in a data processing system, the method comprising:

determining a substantially unique identifier for the given data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier;

determining, using the data identifier, whether the data item is present at the destination location; and

based on the determining whether the data item is present, providing the destination location with the data item only if the data item is not present at the destination.

34. A method as in claim 33, wherein the given data item is a compound data item having a plurality of component data items, the method further comprising:

for each data item of the component data items,

obtaining the component data identifier of the data item by determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier;

determining, using the obtained component data identifier, whether the data item is present at the destination; and

based on the determining, providing the destination with the data item only if the data item is not present at the destination.

35. A method for determining whether a particular data item is present in a data processing system, the method comprising:

(A) for each data item of a plurality of data items present in the system,

(i) determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier; and

(ii) making and maintaining a set of identifiers of the plurality of data items; and

(B) for the particular data item,

(i) determining a particular substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier; and

(ii) determining whether the particular identifier is in the set of data items.

36. A method of backing up, of a plurality of data items present in a data processing system, data items modified since a previous backup time in the data processing system, the method comprising:

(A) maintaining a backup record of identifiers of data items backed up at the previous backup time; and

(B) for each of the plurality of data items present in the data processing system,

(i) determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier;

(ii) determining those data items of the plurality of data items whose identifiers are not in the backup record; and

(iii) based on the determining, copying only those data items whose data identities are not recorded in the backup record.

37. A method as in claim 36, further comprising:

recording in the backup record the identifiers of those data items copied in said copying.

38. A method of locating a particular data item at a location in a data processing system, the method comprising:

(A) determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier;

(B) requesting the particular data item by sending the data identifier of the data item from the requester location to at least one location of a plurality of provider locations in the system; and

(C) on at least some of the provider locations,

(a) for each data item of a plurality of data items at the provider locations,

(i) determining a substantially unique identifier for the data item, the identifier depending on and being determined using all of the data in the data item and only on the data in the data item, whereby two identical data items in the system will have the same identifier; and

(ii) making and maintaining a set of identifiers of data items,

(b) determining, based on the set of identifiers, whether the data item corresponding to the requested data identifier is present at the provider location; and

(c) based on the determining, when the provider location determines that the particular data item is present at the provider location, notifying the requestor that the provider has a copy of the given data item.

39. The method of claim 38, further comprising:

(a) for each data item of a plurality of data items present at said provider locations,

making and maintaining an association between the data item and the identifier of the data item,

(b) in response to said notifying, said client location copying said data item from one of said responding remote locations, using said association to access the data item given the data identifier.

40. A method of locating a particular data item among a plurality of locations, each of the locations having a plurality of data items, the method comprising:

determining, for the particular data item and for each data item of the plurality of data items, a substantially unique identifier for the data item, the identifier depending on and being determined using all of the

data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier; and

determining the presence of the particular data item in each of the plurality of locations by determining whether the identifier of the particular data item is present at each of the locations.

41. The method of claim **30**, wherein said accessing further comprises: for a given data identifier and for a given current location and a remote location in the system:

determining whether the data item corresponding to the given data identifier is present at the current location, and

based on said determining, if said data item is not present at the current location, fetching the data item from a remote location in the system to the current location.

42. The method of claim **41**, further comprising:

for each contextual name at a location,

making and maintaining a context association between the context name of a data item and the identifier of said data item, and when some context association changes at said current location, and

notifying said remote location of a modification to the context association.

43. The method of claim **42**, further comprising:

at said remote location, updating the association between the contextual identifier of the data item and the identifier of the data item.

44. The method of claim **43**, further comprising:

from said remote location, notifying all other locations that said data item has been modified, by providing the contextual identifier and data identifier of said data item to said other locations.

45. The method of claim **44**, further comprising, at each location notified that the data item has been modified:

modifying an association between the contextual identifier of the data item and the data identifier of the data item, to record that the data item has been modified.

46. A method of maintaining at least a predetermined number of copies of a given data item in a data processing system, at different locations in the data processing system, the data processing system being one wherein data is identified by a substantially unique identifier, the identifier depending on and being determined using all of the data in the data item and only the data in the data item, whereby two identical data items in the system will have the same identifier, and wherein any data item in the system may be accessed using only the identifier of the data item, the method comprising:

(i) sending, from a first location in the system, the data identifier of the given data item to other locations in the system; and

(ii) in response to the sending, at each of the other locations,

(A) determining whether the data item corresponding to the data identifier is present at the other location, and based on the determining, and

(B) informing the first location whether the data item is present at the other location; and

(iii) in response to the informing from the other locations, at the first location,

(A) determining whether the data item is present in at least the predetermined number of other locations, and based on the determining,

(B) when less than the predetermined number of other locations have a copy of the data item, requesting some locations that do not have a copy of the data item make a copy of the data item.

47. A method as in claim **46**, wherein said step (iii) further comprises:

(C) when more than the predetermined number of other locations have a copy of the data item present, requesting some locations that do have a copy of the data item present delete the copy of the data item.

48. A method as in any of claims **30**–**45**, **46** and **47**, wherein said data items are at least one of a file, a database record, a message, a data segment, a data block, a directory, and an instance of an object class.

* * * * *

US005978791C1

(12) **EX PARTE REEXAMINATION CERTIFICATE** (10907th)

United States Patent
Farber et al.

(10) **Number: US 5,978,791 C1**
(45) **Certificate Issued: Jul. 19, 2016**

(54) **DATA PROCESSING SYSTEM USING SUBSTANTIALLY UNIQUE IDENTIFIERS TO IDENTIFY DATA ITEMS, WHEREBY IDENTICAL DATA ITEMS HAVE THE SAME IDENTIFIERS**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignee: **PERSONALWEB TECHNOLOGIES, LLC**

Reexamination Request:
No. 90/012,931, Jul. 26, 2013

Reexamination Certificate for:
Patent No.: **5,978,791**
Issued: **Nov. 2, 1999**
Appl. No.: **08/960,079**
Filed: **Oct. 24, 1997**

Related U.S. Application Data

(63) Continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/30091*** (2013.01); ***G06F 17/3012*** (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

(56) **References Cited**

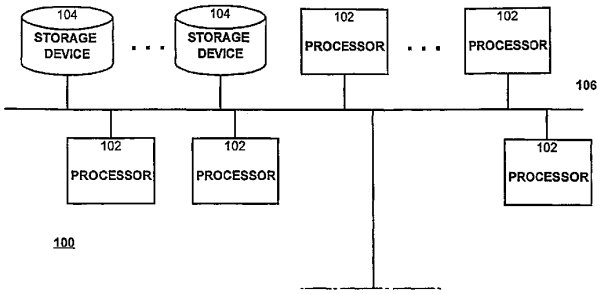
To view the complete listing of prior art documents cited during the proceeding for Reexamination Control Number 90/012,931, please refer to the USPTO's public Patent Application Information Retrieval (PAIR) system under the Display References tab.

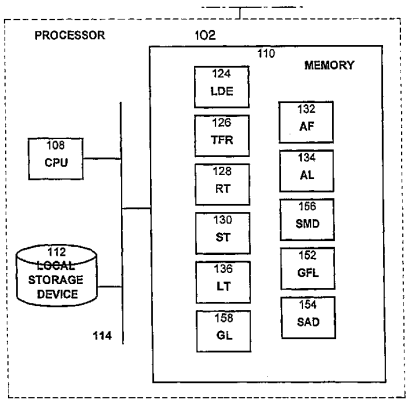
Primary Examiner — Karin Reichle

(57) **ABSTRACT**

In a data processing system, a mechanism identifies data items by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items. The system also determines whether a particular data item is present in the database by examining the identifiers of the plurality of data items.

Attention is directed to the decision of *Personal Web Technologies, LLC* v. *EMC Corporation*, VMWare Inc., 2014-1602, U.S. Court of Appeals for the Federal Circuit, Aug. 10, 2015. This reexamination may not have resolved all questions raised by this decision. See 37 CFR 1.552(c) for *ex parte* reexamination and 37 CFR 1.906(c) for *inter partes* reexamination.





EX PARTE REEXAMINATION CERTIFICATE

THE PATENT IS HEREBY AMENDED AS INDICATED BELOW.

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

Claim **35** is cancelled.

Claims **1-34** and **36-48** were not reexamined.

* * * * *

US006928442B2

(12) **United States Patent**
Farber et al.

(10) **Patent No.: US 6,928,442 B2**
(45) **Date of Patent: Aug. 9, 2005**

(54) **ENFORCEMENT AND POLICING OF LICENSED CONTENT USING CONTENT-BASED IDENTIFIERS**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignees: **Kinetech, Inc.**, Northbrook, IL (US); **Savvis, Inc.**, Town & Country, MO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/987,723**

(22) Filed: **Nov. 15, 2001**

(65) **Prior Publication Data**

US 2002/0052884 A1 May 2, 2002

Related U.S. Application Data

(63) Continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**[7] **G06F 17/30**

(52) **U.S. Cl.** **707/10**; 707/3; 707/101; 707/200; 709/203; 709/219; 709/229

(58) **Field of Search** 707/3, 6, 9, 10, 707/101, 200; 709/203, 219, 229

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,668,647	A	6/1972	Evangelisti
4,215,402	A	7/1980	Mitchell
4,290,105	A	9/1981	Cichelli
4,376,299	A	3/1983	Rivest
4,405,829	A	9/1983	Rivest
4,412,285	A	10/1983	Neches

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0592045	4/1994

OTHER PUBLICATIONS

Gwertzman, James, et al. "The Case for Geographical Push–Caching." Technical Report HU TR 34–94 (excerpt), Harvard University, DAS, Cambridge, MA 02138, 1994, 2 pgs.

Grigni, Michelangelo, et al. "Tight Bounds on Minimum Broadcasts Networks." SIAM Journal of Discrete Mathematics, vol. 4, No. 2, May 1991, pp. 207–222.

Devine, Robert. "Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm." In Proceedings of 4th International Conference on Foundations of Data Organizations and Algorithms, 1993, pp. 101–114.

Deering, Stephen, et al. "Multicast Routing in Datagram Internetworks and Extended LANs." ACM Transactions on Computer Systems, vol. 8, No. 2, May 1990, pp. 85–110.

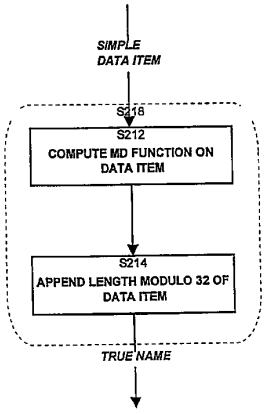
(Continued)

Primary Examiner—Luke S Wossum
Assistant Examiner—Khanh Pham
(74) *Attorney, Agent, or Firm*—Davidson Berquist Jackson & Gowdey, LLP

(57) **ABSTRACT**

Data files are distributed across a plurality of computers. The computers may form a network such as a content delivery network (CDN) or a peer-to-peer network. The network may operate as a TCP/IP network such as the Internet. Data files may represent may represent digital messages, images, videos or audio signals. For content—data items or files in the system—a name is obtained (or determined), where the name is based, at least in part, on a given function of the data in a data item or file. The given function may be a message digest or hash function, and it may be MD4, MD5, and SHA. A cony of a requested file is only provided to licensed (or authorized) parties. The system may check one or more computers for unauthorized or unlicensed content. Content is served based on a measure of availability of servers.

56 Claims, 31 Drawing Sheets



U.S. PATENT DOCUMENTS

4,414,624	A		11/1983	Summer, Jr.
4,441,155	A		4/1984	Fletcher
4,464,713	A		8/1984	Benhase
4,490,782	A		12/1984	Dixon
4,571,700	A		2/1986	Emry, Jr.
4,577,293	A		3/1986	Matick
4,642,793	A		2/1987	Meaden
4,675,810	A		6/1987	Gruner
4,691,299	A		9/1987	Rivest
4,725,945	A		2/1988	Kronstadt
4,773,039	A		9/1988	Zamora
4,887,235	A		12/1989	Holloway
4,888,681	A		12/1989	Barnes
4,922,414	A		5/1990	Holloway
4,922,417	A		5/1990	Churm et al. 707/1
4,972,367	A		11/1990	Burke
5,025,421	A		6/1991	Cho
5,050,074	A		9/1991	Marca
5,050,212	A		9/1991	Dyson
5,057,837	A		10/1991	Colwell
5,077,658	A		12/1991	Bendert
5,129,081	A		7/1992	Kobayashi
5,129,082	A		7/1992	Tirfing
5,144,667	A		9/1992	Pogue, Jr.
5,179,680	A		1/1993	Colwell
5,202,982	A		4/1993	Gramlich et al. 707/2
5,208,858	A		5/1993	Vollert
5,276,901	A		1/1994	Howell
5,287,499	A		2/1994	Nemes 707/2
5,301,286	A		4/1994	Rajani
5,301,316	A		4/1994	Hamilton
5,341,477	A		8/1994	Pitkin et al. 709/226
5,343,527	A		8/1994	Moore
5,357,623	A		10/1994	Megory-Cohen
5,384,565	A		1/1995	Cannon
5,404,508	A		4/1995	Konrad
5,452,447	A		9/1995	Nelson et al. 707/205
5,459,860	A		10/1995	Burnett
5,542,087	A		7/1996	Neimat et al. 707/10
5,581,758	A		12/1996	Burnett
5,638,443	A		6/1997	Stefik et al. 705/54
5,640,564	A		6/1997	Hamilton et al. 709/303
5,781,629	A	*	7/1998	Haber et al. 713/177
5,802,291	A		9/1998	Balick et al. 709/202
5,809,494	A		9/1998	Nguyen 707/1
5,835,087	A	*	11/1998	Herz et al. 345/810
5,907,704	A		5/1999	Gudmundson et al.
6,006,018	A		12/1999	Burnett et al. 395/200.49
6,134,603	A		10/2000	Jones et al. 709/330

OTHER PUBLICATIONS

Cormen, Thomas H., et al. *Introduction to Algorithms,* The MIT Press, Cambridge, Massachusetts, 1994, pp. 219–243, 991–993.

Naor, Moni, et al. "The Load, Capacity and Availability of Quorum Systems." In Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Nov. 1994, pp. 214–225.

Nisan, Noam. "Pseudorandom Generators for Space–Bounded Computation." In Proceedings of the Twenty–Second Annual ACM Symposium on Theory of Computing, May 1990, pp. 204–212.

Palmer, Mark, et al. "Fido: A Cache that Learns to Fetch." In Proceedings of the 17th International Conference on Very Large Data Bases, Sep. 1991, pp. 255–264.

Peleg, David, et al. "The Availability of Quorum Systems." Information and Computation 123, 1995, 210–223.

Rabin, Michael. "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." Journal of the ACM, vol. 36, No. 2, Apr. 1989, pp. 335–348.

Ravi, R., "Rapid Rumor Ramification: Approximating the Minimum Broadcast Time." In Proceedings of the 35th IEEE Symposium on Foundation of Computer Science, Nov. 1994, pp. 202–213.

Schmidt, Jeanette, et al. "Chernoff–Hoeffding Bounds for Applications with Limited Independence." In Proceedings of the 4th ACS–SIAM Symposium on Discrete Algorithms, 1993, pp. 331–340.

Tarjan, Robert Endre, et al. "Storing a Sparse Table." Communications of the ACM, vol. 22, No. 11, Nov. 1979, pp. 606–611.

Wegman, Mark, et al. "New Hash Functions and Their Use in Authentication and Set Equality." Journal of Computer and System Sciences vol. 22, Jun. 1981, pp. 265–279.

Vitter, Jeffrey Scott, et al. "Optimal Prefetching via Data Compression." In Proceedings of 32nd IEEE Symposium on Foundations of Computer Science, Nov. 1991, pp. 121–130.

Fredman, Michael, et al. "Storing a Sparse Table with 0(1) Worst Case Access Time." Journal of the Association for Computing Machinery, vol. 31, No. 3, Jul. 1984, pp. 538–544.

Yao, Andrew Chi–Chih. "Should Tables be Sorted?" Journal of the Association for Computing Machinery, vol. 28, No. 3, Jul. 1981, pp. 615–628.

Floyd, Sally, et al. "A reliable Multicast Framework for Light–Weight Sessions and Application Level Framing." In Proceeding of ACM SIGCOMM '95, pp. 342–356.

Feeley, Michael, et al. "Implementing Global Memory Management in a Workstation Cluster." In Proceedings of the 15th ACM Symposium on Operating Systems Principles, 1995, pp. 201–212.

Carter, J. Lawrence, et al. "Universal Classes of Hash Functions." Journal of Computer and System Sciences, vol. 18, No. 2, Apr. 1979, pp. 143–154.

Patent Abstracts of Japan, "Electronic Mail Multiplexing System and Communication Control Method in The System." Jun. 30, 1993, JP 05162529.

Kim et al., "Experiences with Tripwire: Using Integrity Checkers For Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1–12.

Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1–21.

Bert dem Boer et al., Collisions for the compression function of MD_5, pp. 292–304.

Sakti Pramanik et al., Multi–Directory Hasing, 1993, Info. Sys., vol. 18, No. 1, pp. 63–74.

Murlidhar Koushik, Dynamic Hashing with Distributed Overflow Space: A File Organization with Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299–317.

Witold Litwin et al., LH*–Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL–93–21, Jun. 1993, pp. 1–22.

Yuliang Zheng et al., HAVAL—A One–Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83–105.

Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, pp. 156–164.

Witold Litwin et al., Linear Hashing for Distributed Files, ACM SIGMOD, May 1993, pp. 327–336.
Ming–Ling Lo et al., On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69–78, May 1993.
Thomas A. Berson, Differential Cryptanalysis Mod 2^{32} with Applications to MD5, pp. 69–81.
William Perrizo et al., Distributed Join Processing Performance Evaluation, Twenty–Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236–244.
Vijay Kumar, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109–113.
Birgit Pfitzman, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93, p. 74–85.
Zhiyu Tian et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3–13.
G. L. Friedman, Digital Camera with Apparatu for Authentication of Images Produced from an Image File, NASA Case No. NPO–19108–1–CU, U.S. Appl. No. 08/159,980, filed Nov. 24, 1993.
H. Goodman, Ada, Object–Oriented Techniques, and Concurrency in Teaching Data Structures and File Management Report Documentation p. AD–A275 385 – 94–04277.
Advances in Cryptology—Eurocrypt'93, Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23–27, 1993 Proceedings.
Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.
Advances in Cryptology–AUSCRYPT '92—Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, Dec. 13–16, 1992 Proceedings.
Peter Deutsch (peterd@bunyip.com), "Re: MD5 and LiFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri–94q2.messages/0106.html, Apr. 26, 1994.
Alexander Dupuy (dupuy@smarts.com), "RE: MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri–94q2.messages/0113.html, Apr. 26, 1994.
Alexander Dupuy (dupuy@smarts.com), "MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri–94q2.messages/0081.html, Apr. 17, 1994.
Albert Langer (cmf851@anu.oz.au), http://groups.google.com/groups?selm=1991Aug7.225159.786%40newshost.anu.edu.au&oe=UTF–8&output=gplain, Aug. 7, 1991.
Clifford Lynch (Calur@uccmvsa.bitnet), "ietf url/uri overview draft paper (long)", www.acl.lanl.gov/URI/archive/uri–93q1.messages/0015.html, Mar. 25, 1993.
K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names", www.w3.org/Addressing/rfc1737.txt, Dec. 1994, pp. 1–7.
W3C:ID, HTTP: A protocol for networked information, "Basic HTTP as defined in 1992", www.w3.org/Protocols/HTTP2.html, 1992.
European Search Report issued Dec. 23, 2004.

* cited by examiner

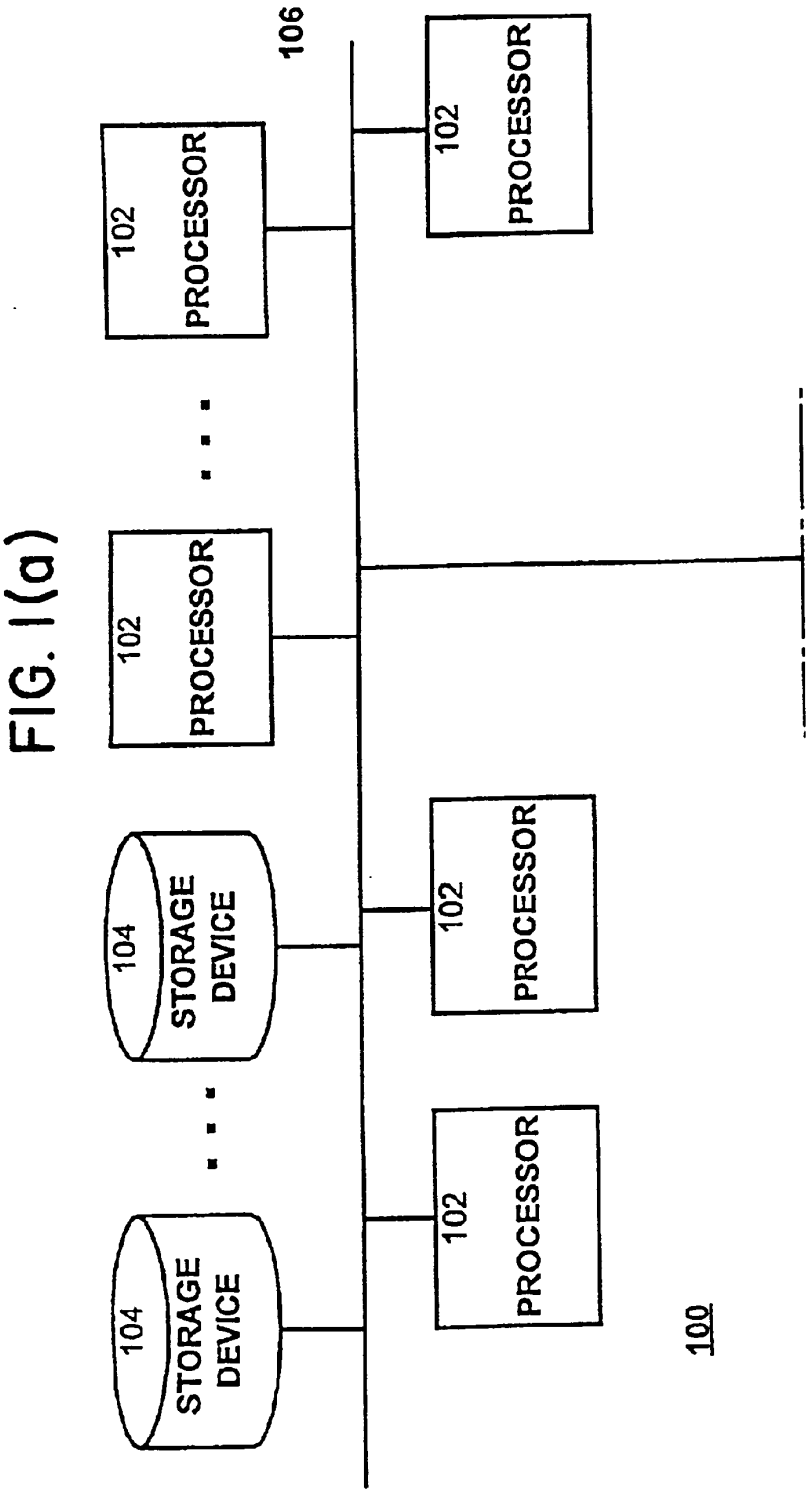

106
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
104
STORAGE DEVICE
104
STORAGE DEVICE
100

FIG. 1(a)

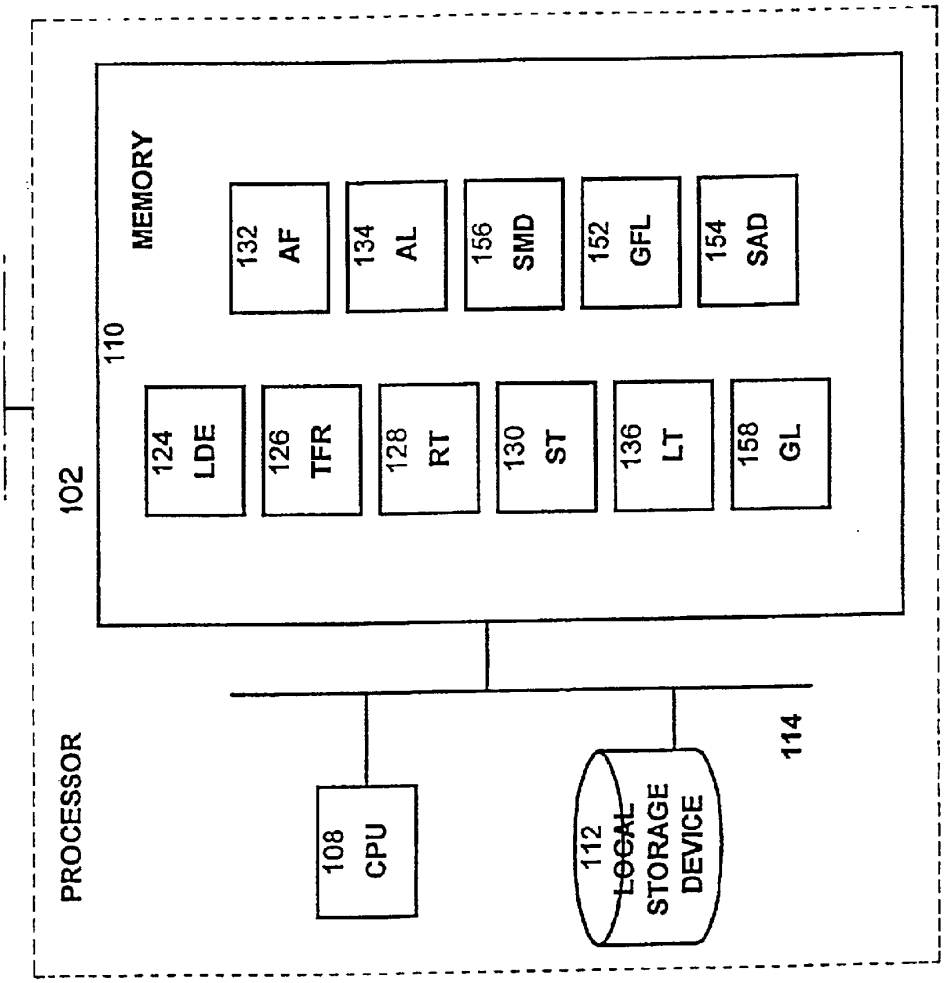

PROCESSOR
102
MEMORY
110
132 AF
134 AL
156 SMD
152 GFL
154 SAD
124 LDE
126 TFR
128 RT
130 ST
136 LT
158 GL
108 CPU
112 LOCAL STORAGE DEVICE
114

FIG. 1(b)

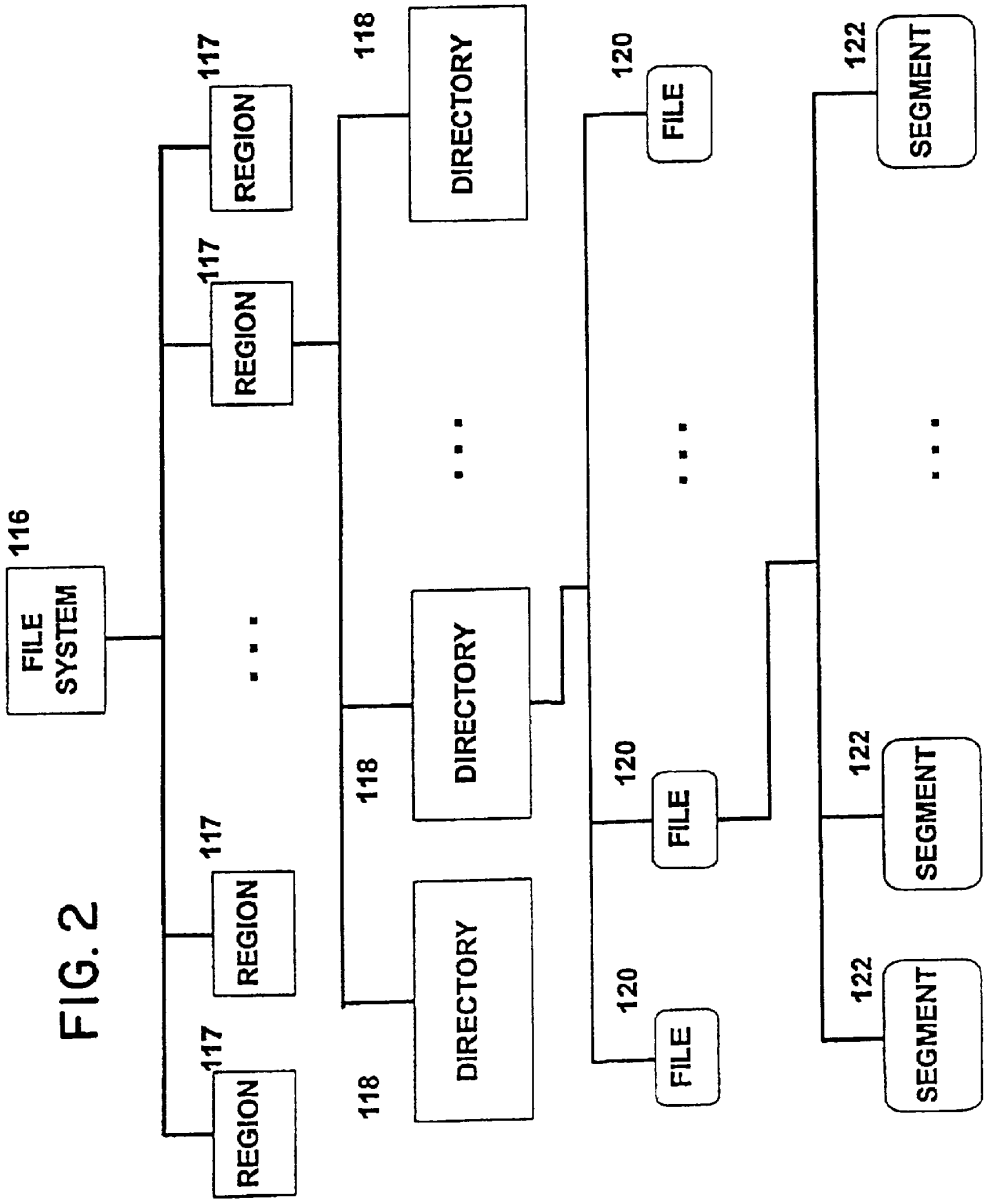

116
FILE SYSTEM
117
REGION
117
REGION
117
REGION
117
REGION
118
DIRECTORY
118
DIRECTORY
118
DIRECTORY
120
FILE
120
FILE
120
FILE
122
SEGMENT
122
SEGMENT
122
SEGMENT

FIG. 2

FIG. 3

138

Region ID
Pathname
True Name
Type
File ID
Time of last access
Time of last modification
Safe flag
Lock flag
Size
Owner

FIG. 4

140

True Name
File ID
Compressed File ID
Source IDs
Dependent processors
Use count
Time of last access
Expiration
Grooming delete count

142

Region ID
Region file system
Region pathname
Region status
Mirror processor(s)
Mirror duplication count
Policy

FIG. 5

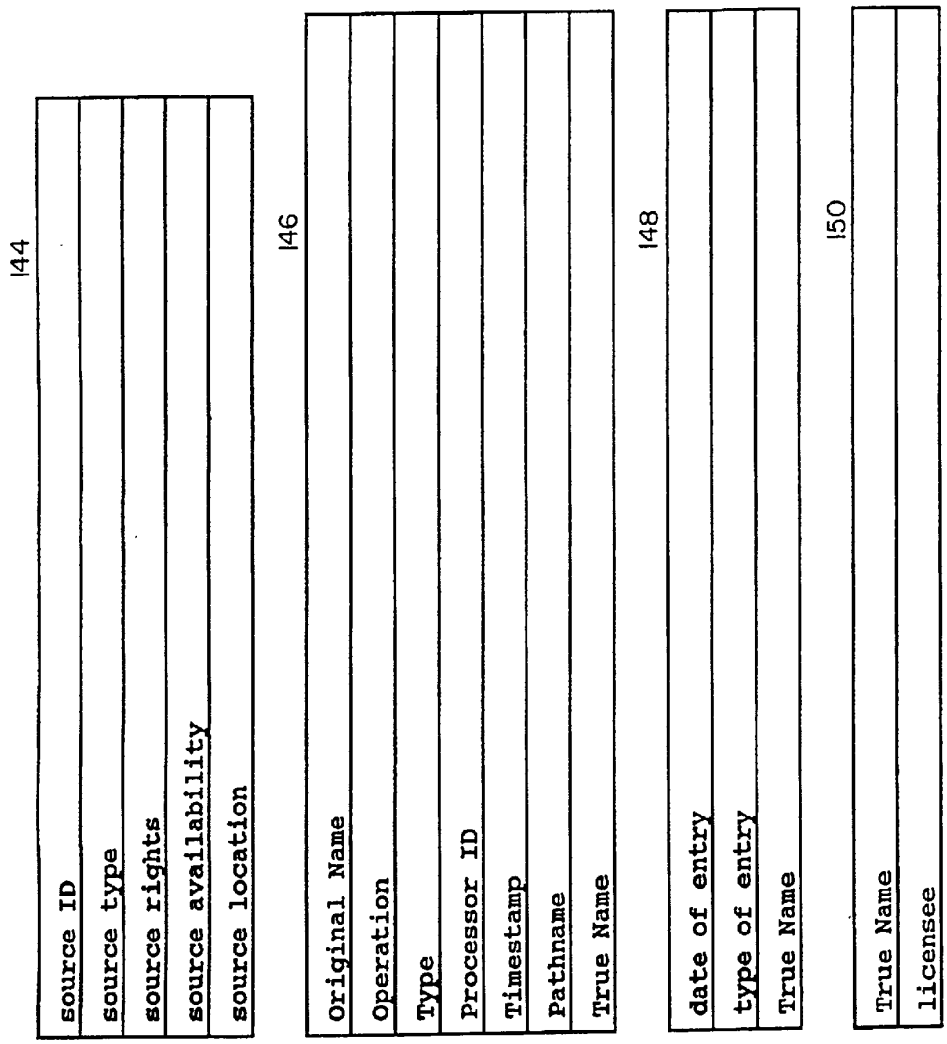


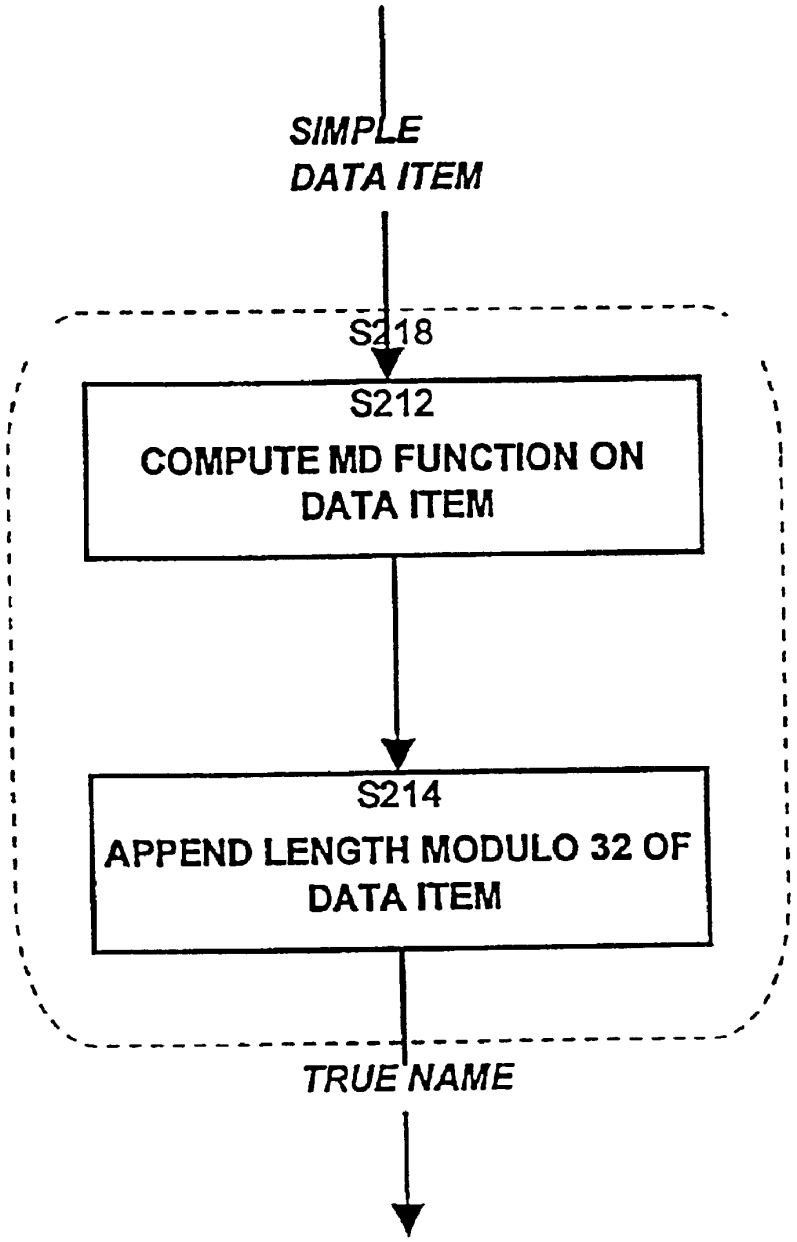
FIG. 6

FIG. 7

FIG. 8

FIG. 9

FIG. 10(a)


SIMPLE
DATA ITEM
S218
S212
COMPUTE MD FUNCTION ON DATA ITEM
S214
APPEND LENGTH MODULO 32 OF DATA ITEM
TRUE NAME

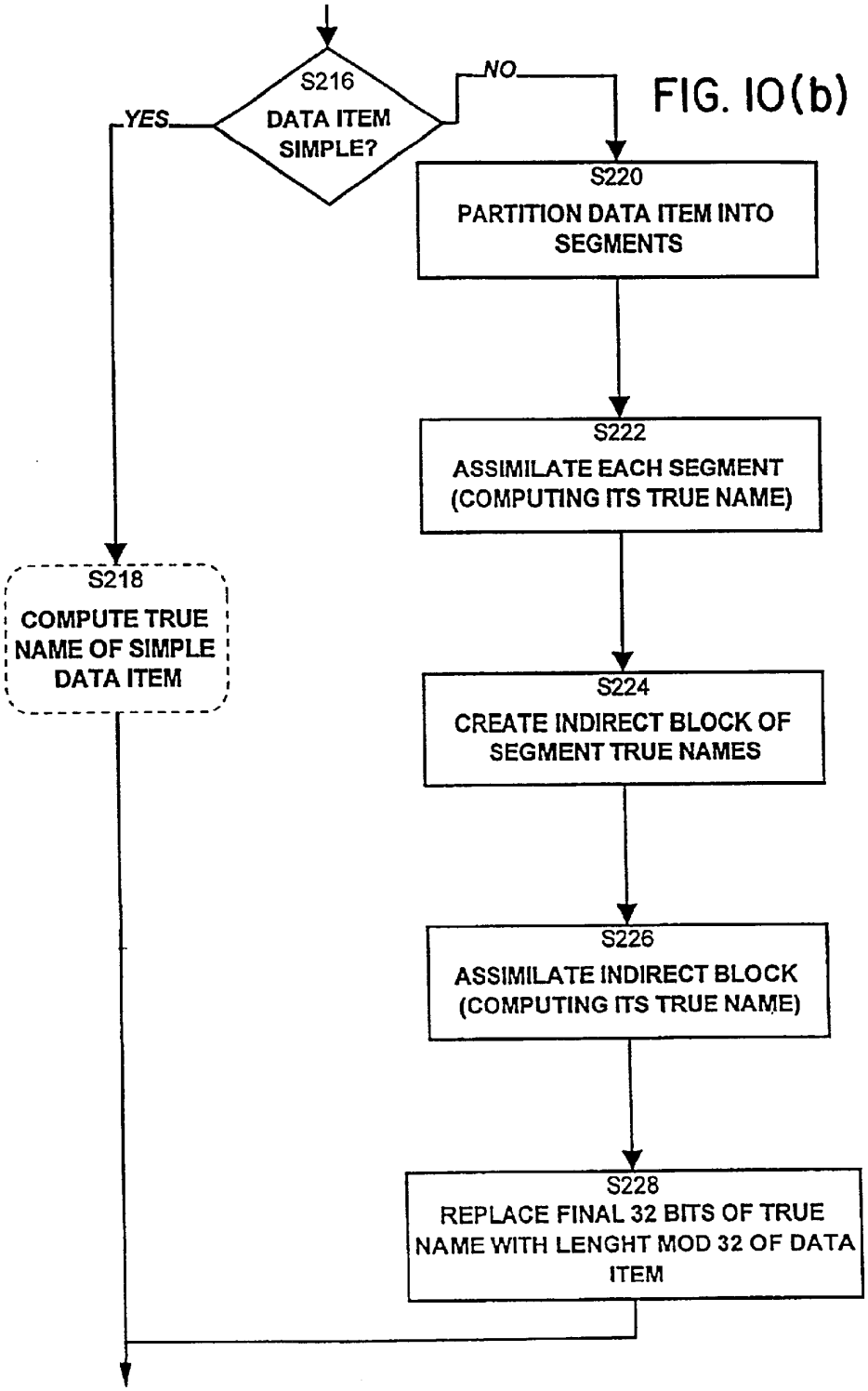

S216
DATA ITEM SIMPLE?
YES
NO
S218
COMPUTE TRUE NAME OF SIMPLE DATA ITEM
S220
PARTITION DATA ITEM INTO SEGMENTS
S222
ASSIMILATE EACH SEGMENT (COMPUTING ITS TRUE NAME)
S224
CREATE INDIRECT BLOCK OF SEGMENT TRUE NAMES
S226
ASSIMILATE INDIRECT BLOCK (COMPUTING ITS TRUE NAME)
S228
REPLACE FINAL 32 BITS OF TRUE NAME WITH LENGHT MOD 32 OF DATA ITEM

FIG. 10(b)

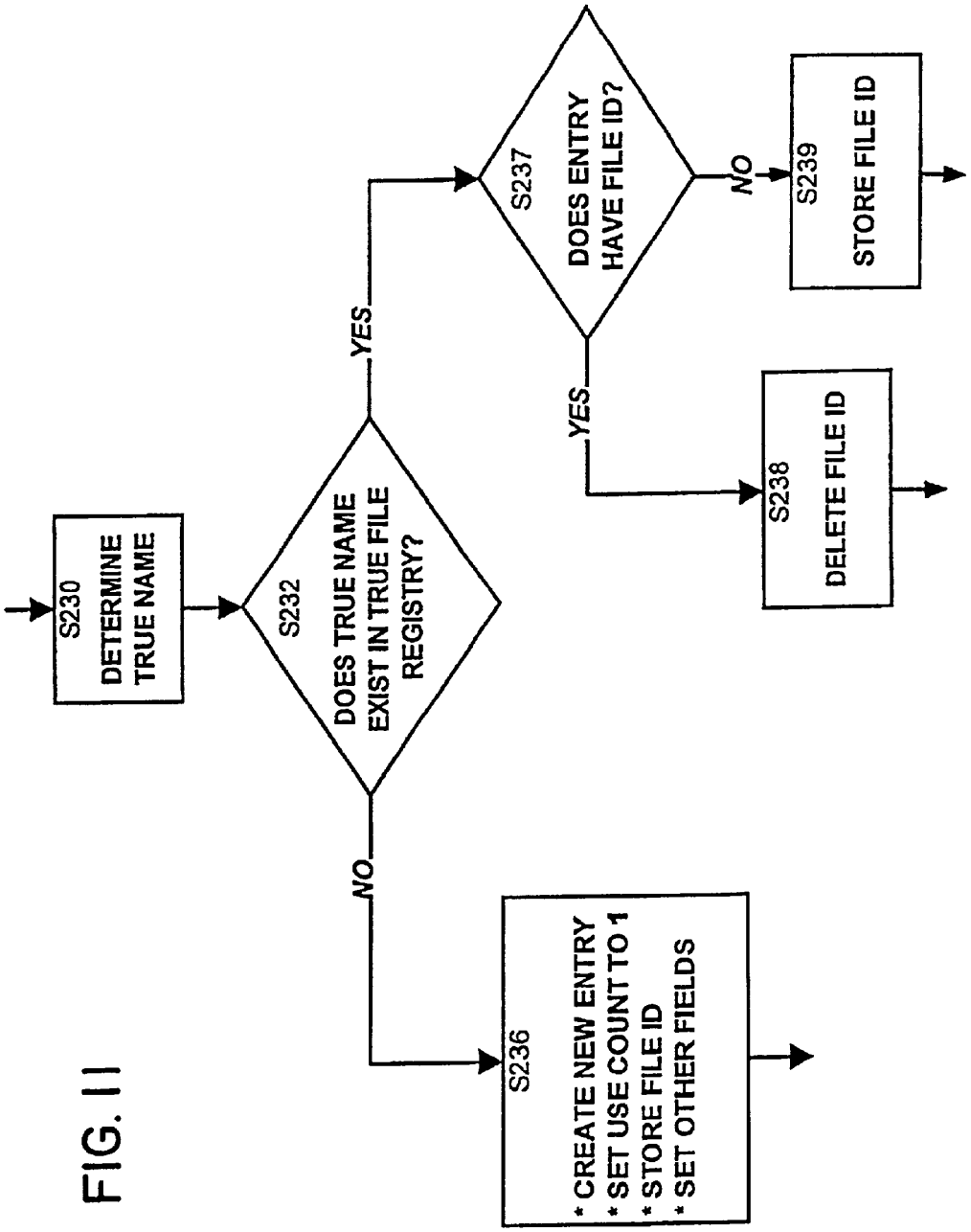

S230
DETERMINE TRUE NAME
S232
DOES TRUE NAME EXIST IN TRUE FILE REGISTRY?
YES
NO
S237
DOES ENTRY HAVE FILE ID?
YES
NO
S238
DELETE FILE ID
S239
STORE FILE ID
S236
* CREATE NEW ENTRY
* SET USE COUNT TO 1
* STORE FILE ID
* SET OTHER FIELDS

FIG. 11

FIG. 12

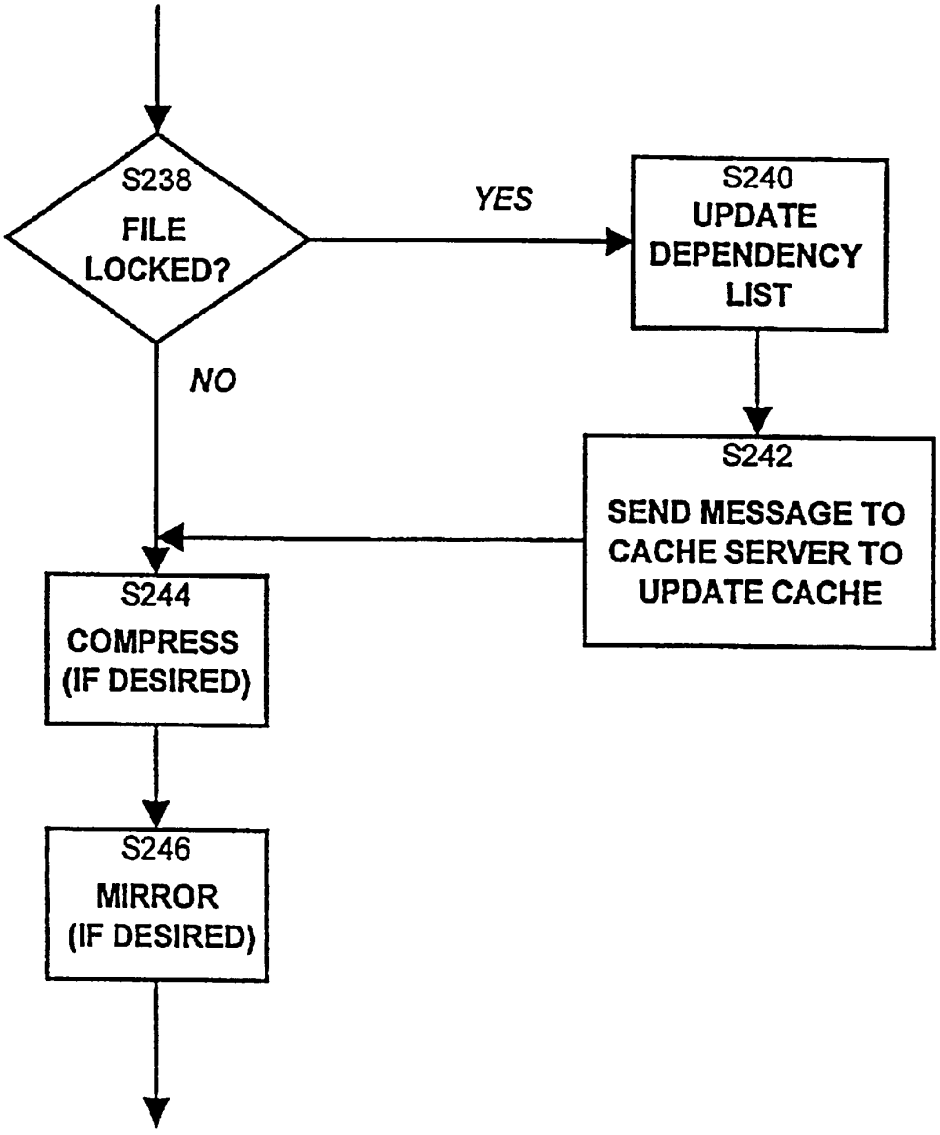
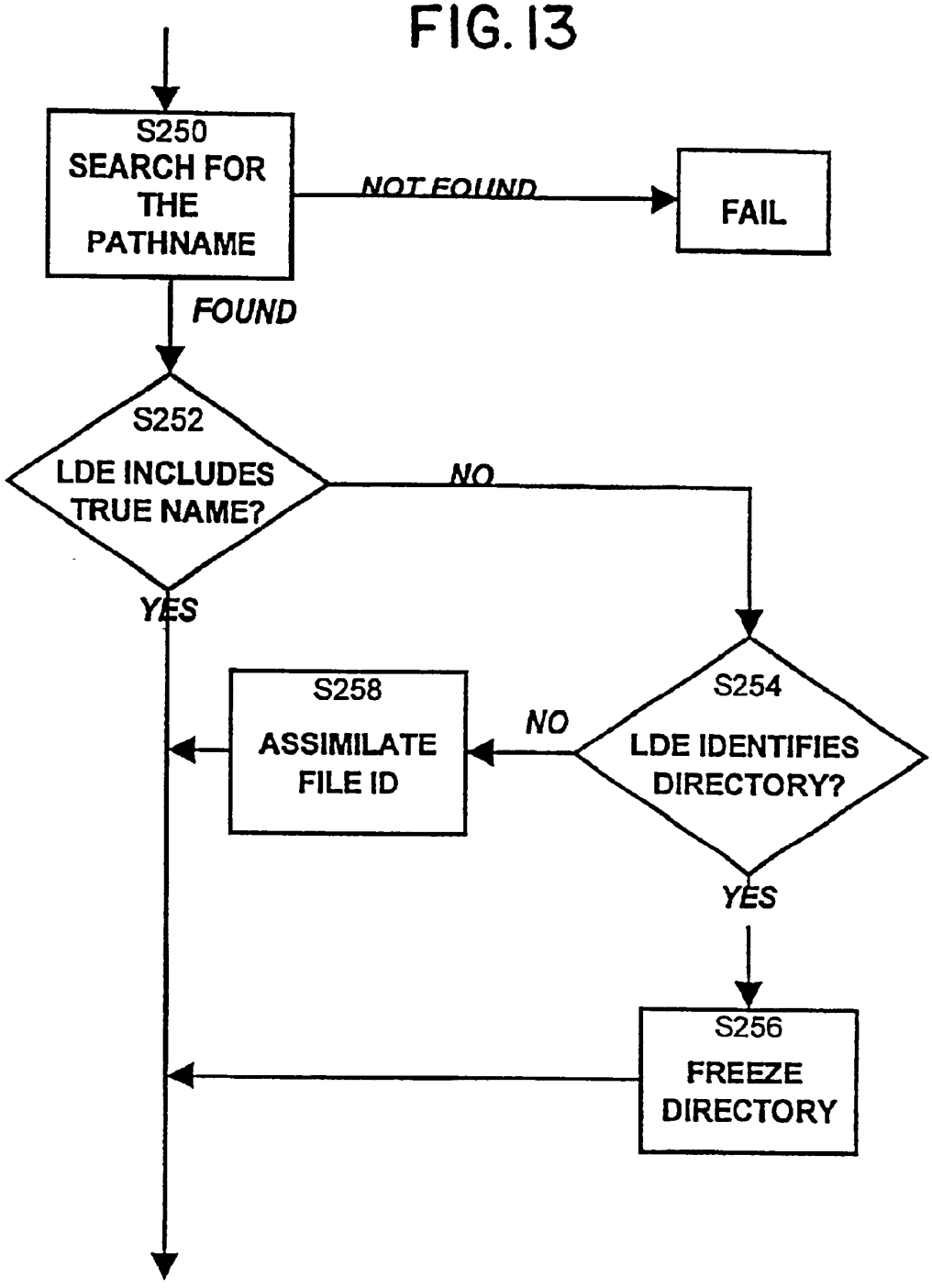

S238
FILE LOCKED?
YES
S240
UPDATE DEPENDENCY LIST
NO
S242
SEND MESSAGE TO CACHE SERVER TO UPDATE CACHE
S244
COMPRESS (IF DESIRED)
S246
MIRROR (IF DESIRED)

FIG. 13


S250
SEARCH FOR THE PATHNAME
NOT FOUND
FAIL
FOUND
S252
LDE INCLUDES TRUE NAME?
NO
YES
S254
LDE IDENTIFIES DIRECTORY?
NO
S258
ASSIMILATE FILE ID
YES
S256
FREEZE DIRECTORY

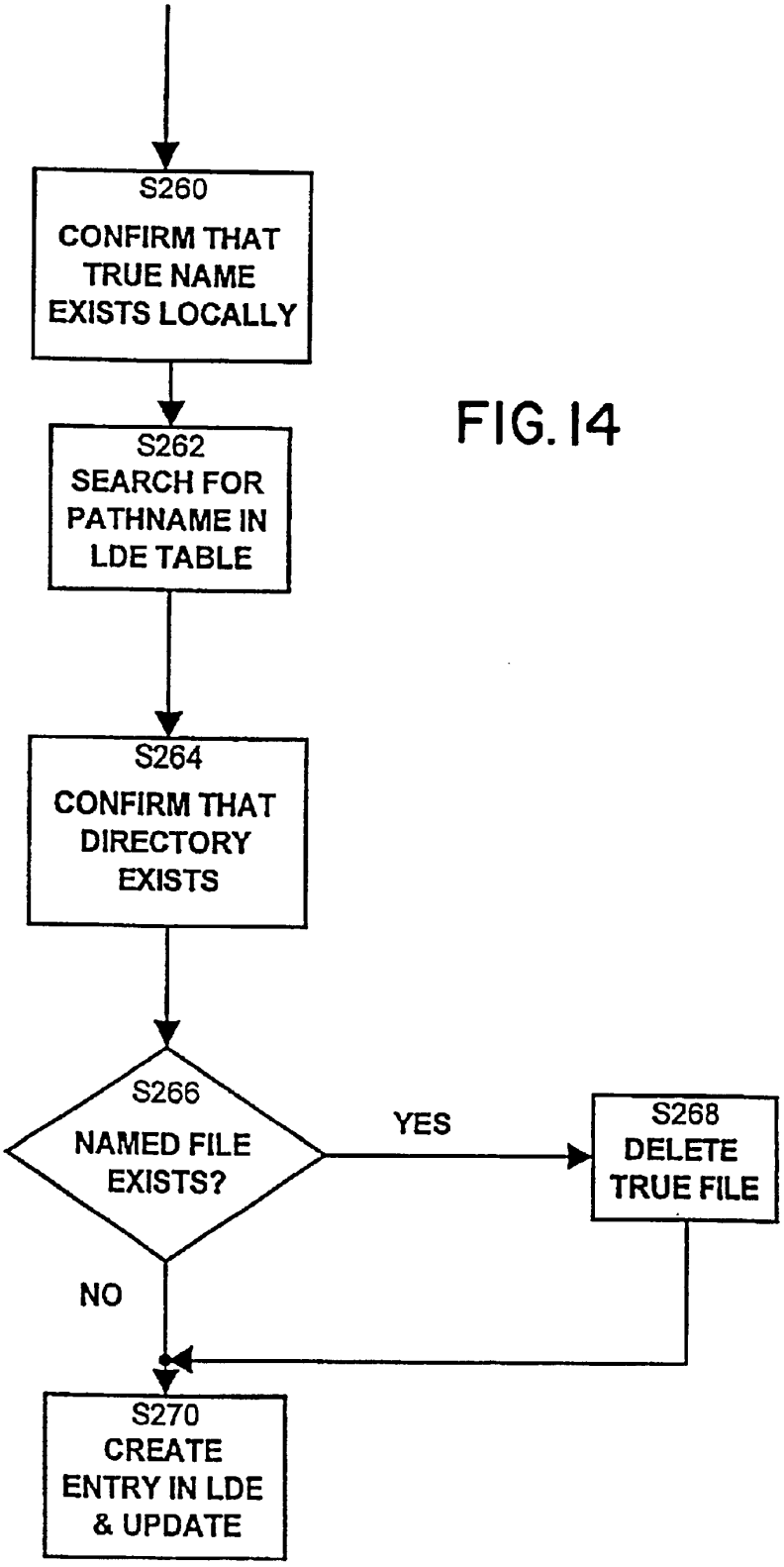

S260
CONFIRM THAT TRUE NAME EXISTS LOCALLY
S262
SEARCH FOR PATHNAME IN LDE TABLE
S264
CONFIRM THAT DIRECTORY EXISTS
S266
NAMED FILE EXISTS?
YES
S268
DELETE TRUE FILE
NO
S270
CREATE ENTRY IN LDE & UPDATE

FIG. 14

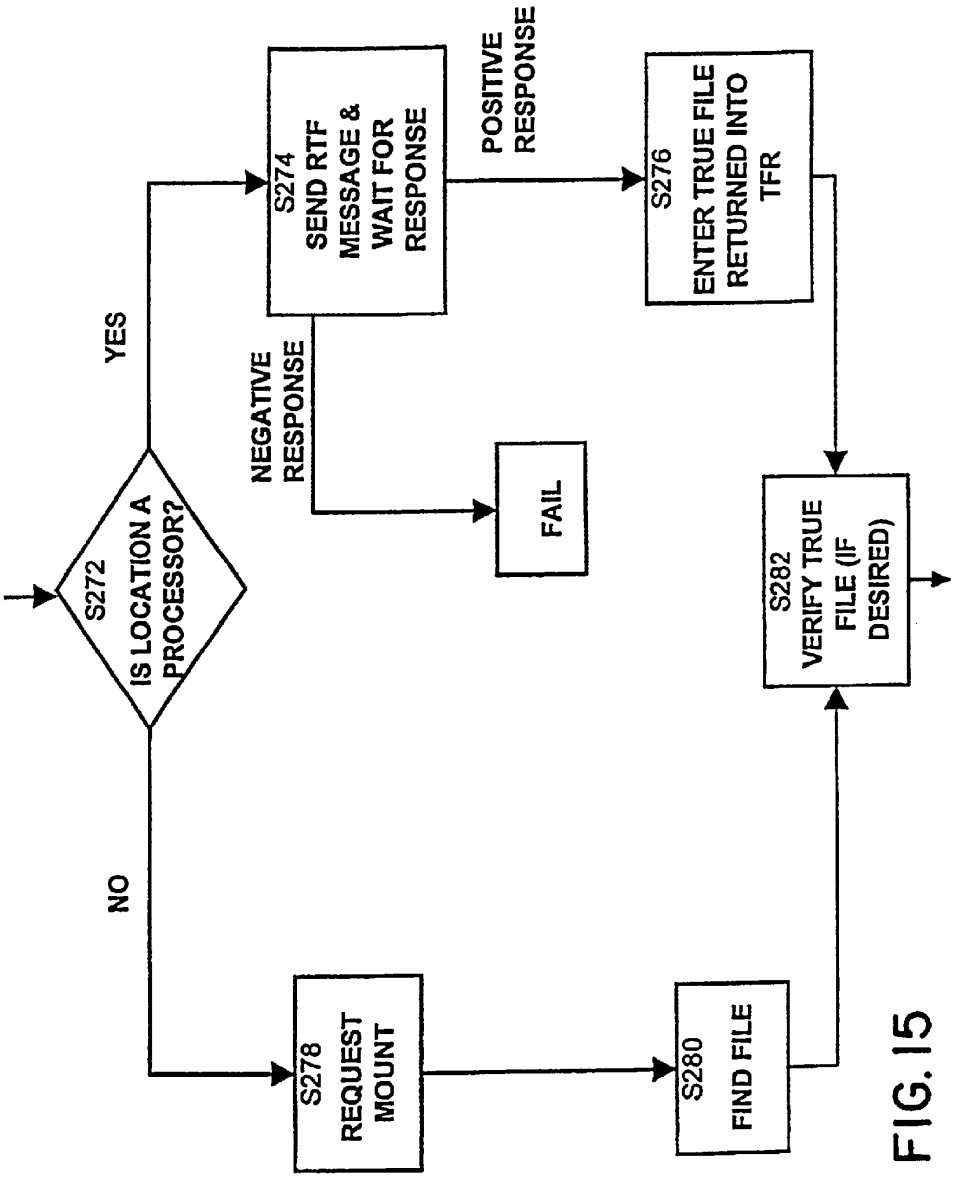
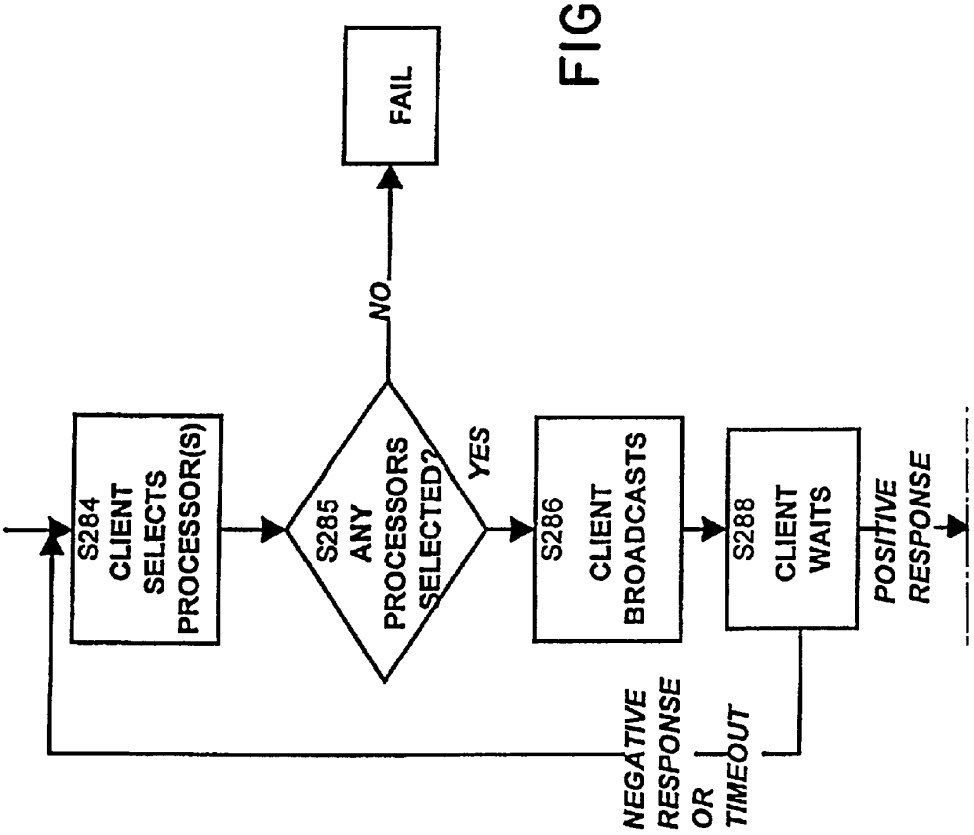

S272
IS LOCATION A PROCESSOR?
YES
NO
S274
SEND RTF MESSAGE & WAIT FOR RESPONSE
POSITIVE RESPONSE
NEGATIVE RESPONSE
FAIL
S276
ENTER TRUE FILE RETURNED INTO TFR
S278
REQUEST MOUNT
S280
FIND FILE
S282
VERIFY TRUE FILE (IF DESIRED)

FIG. 15

FIG. 16(a)


S284
CLIENT
SELECTS
PROCESSOR(S)
S285
ANY
PROCESSORS
SELECTED?
NO
FAIL
YES
S286
CLIENT
BROADCASTS
S288
CLIENT
WAITS
POSITIVE
RESPONSE
NEGATIVE
RESPONSE
OR
TIMEOUT

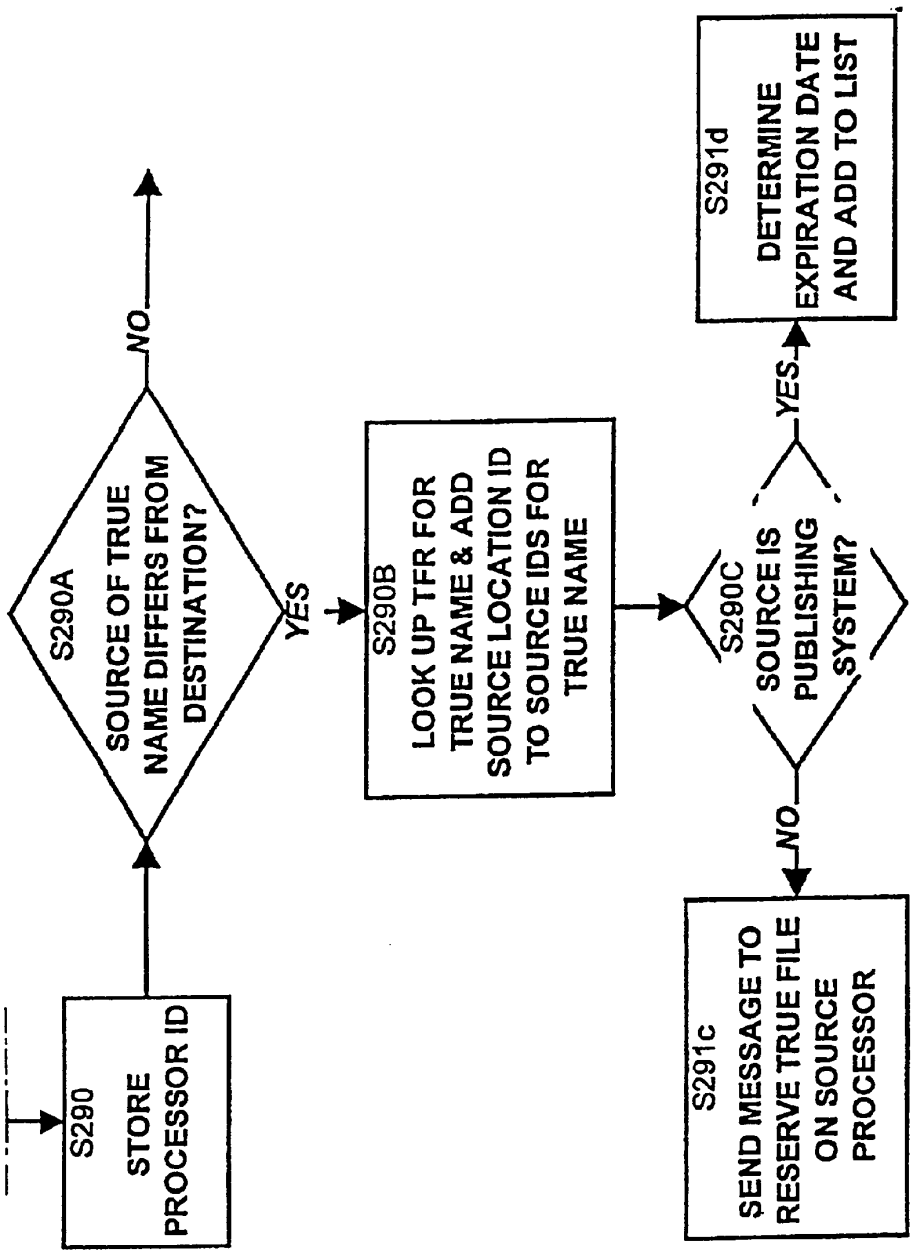

S290
STORE PROCESSOR ID
S290A
SOURCE OF TRUE NAME DIFFERS FROM DESTINATION?
NO
YES
S290B
LOOK UP TFR FOR TRUE NAME & ADD SOURCE LOCATION ID TO SOURCE IDS FOR TRUE NAME
S290C
SOURCE IS PUBLISHING SYSTEM?
NO
YES
S291c
SEND MESSAGE TO RESERVE TRUE FILE ON SOURCE PROCESSOR
S291d
DETERMINE EXPIRATION DATE AND ADD TO LIST

FIG. 16(b)

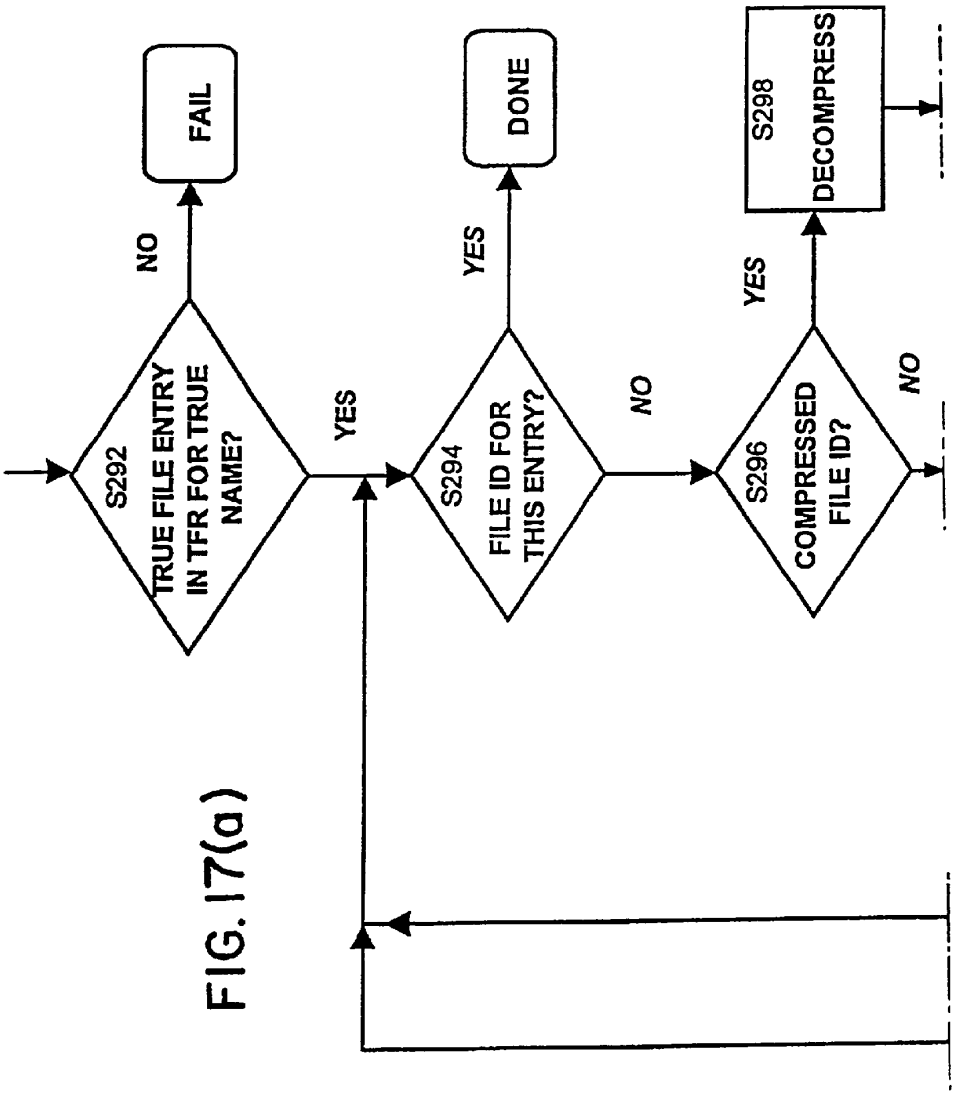

S292
TRUE FILE ENTRY IN TFR FOR TRUE NAME?
NO
FAIL
YES
S294
FILE ID FOR THIS ENTRY?
YES
DONE
NO
S296
COMPRESSED FILE ID?
YES
S298
DECOMPRESS
NO

FIG. 17(a)

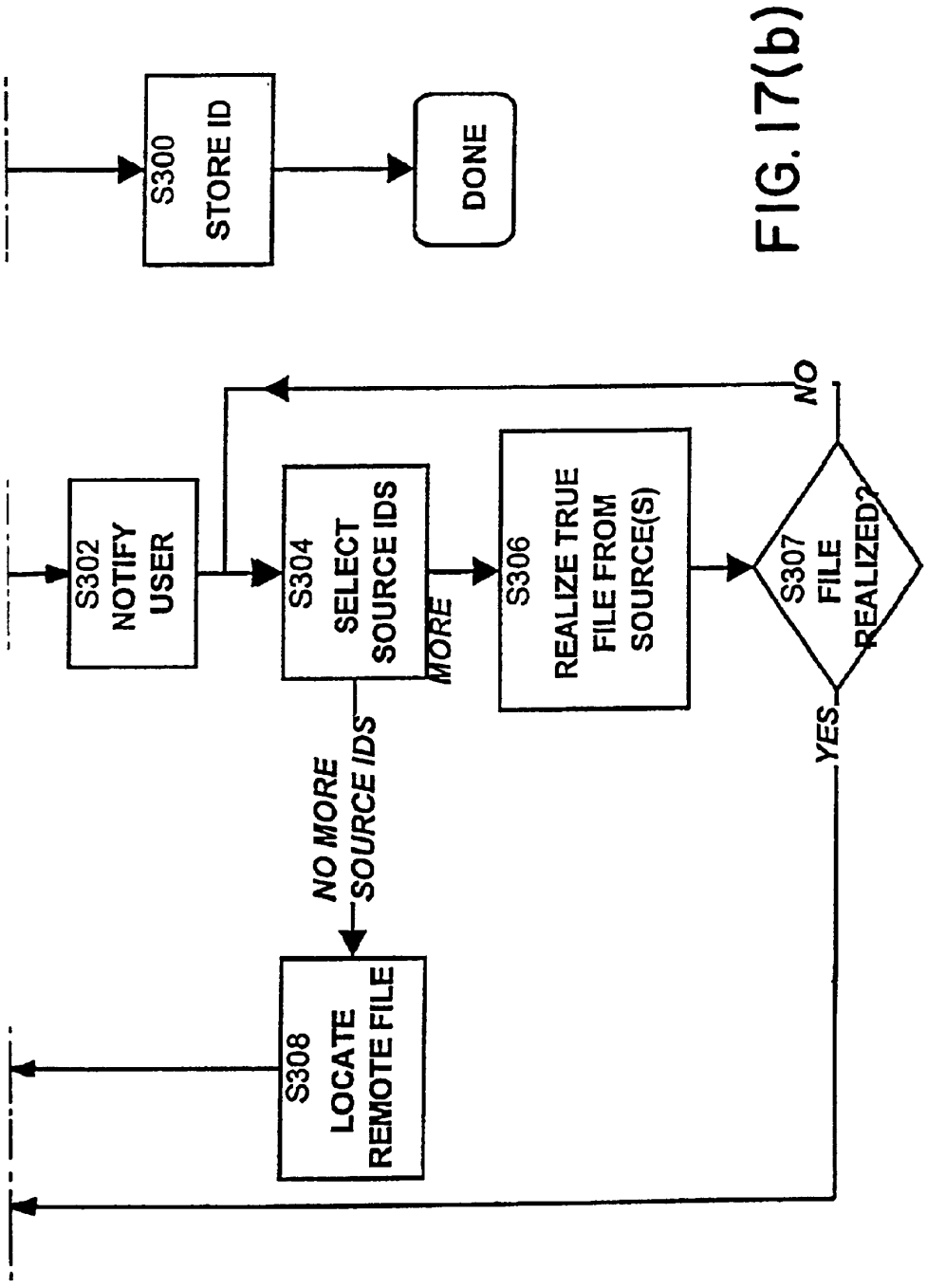
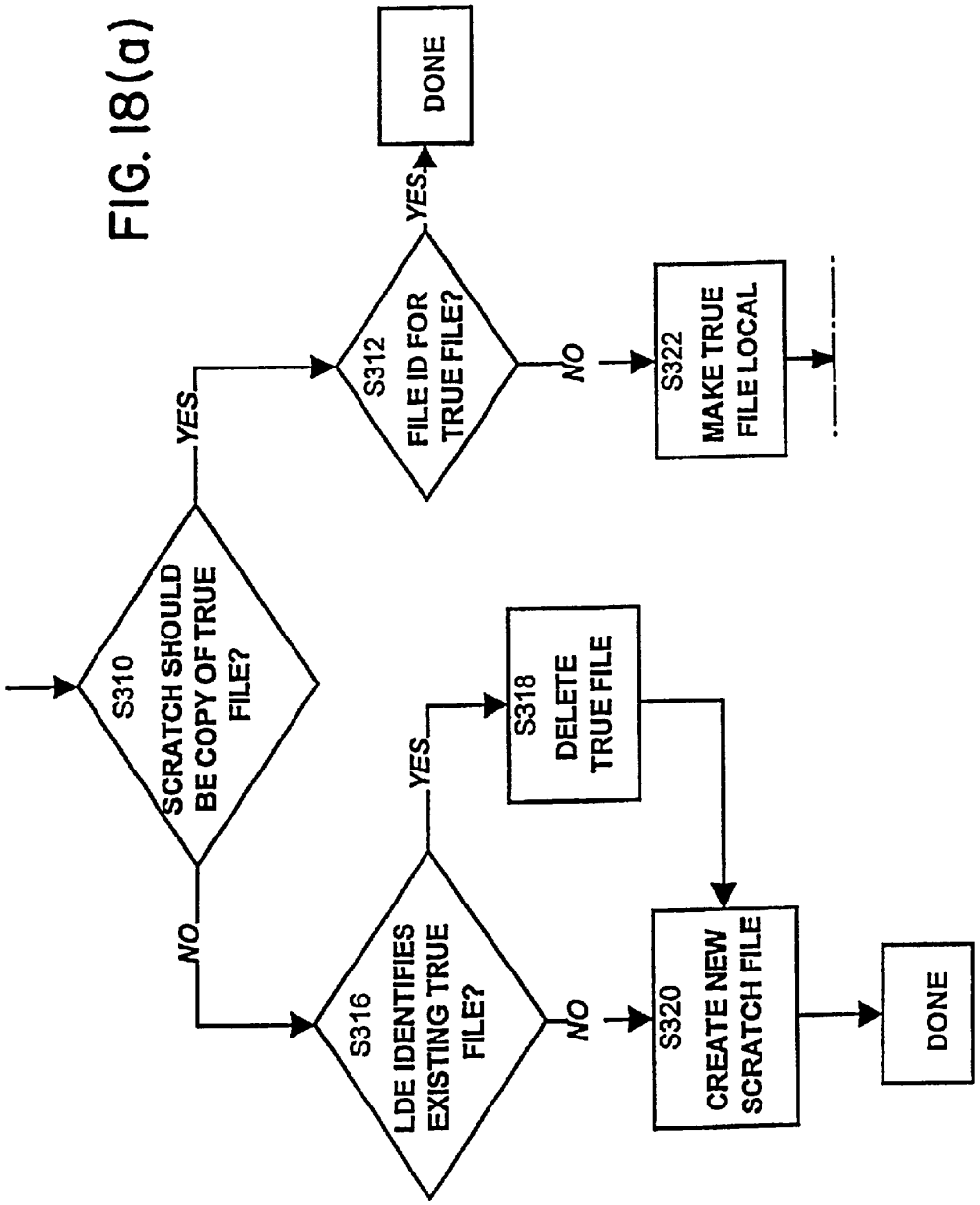

S300
STORE ID
DONE
S302
NOTIFY USER
S304
SELECT SOURCE IDS
MORE
NO MORE SOURCE IDS
S306
REALIZE TRUE FILE FROM SOURCE(S)
S307
FILE REALIZED?
NO
YES
S308
LOCATE REMOTE FILE

FIG. 17(b)

FIG. 18(a)


S310
SCRATCH SHOULD BE COPY OF TRUE FILE?
YES
NO
S312
FILE ID FOR TRUE FILE?
YES
NO
DONE
S322
MAKE TRUE FILE LOCAL
S316
LDE IDENTIFIES EXISTING TRUE FILE?
YES
NO
S318
DELETE TRUE FILE
S320
CREATE NEW SCRATCH FILE
DONE

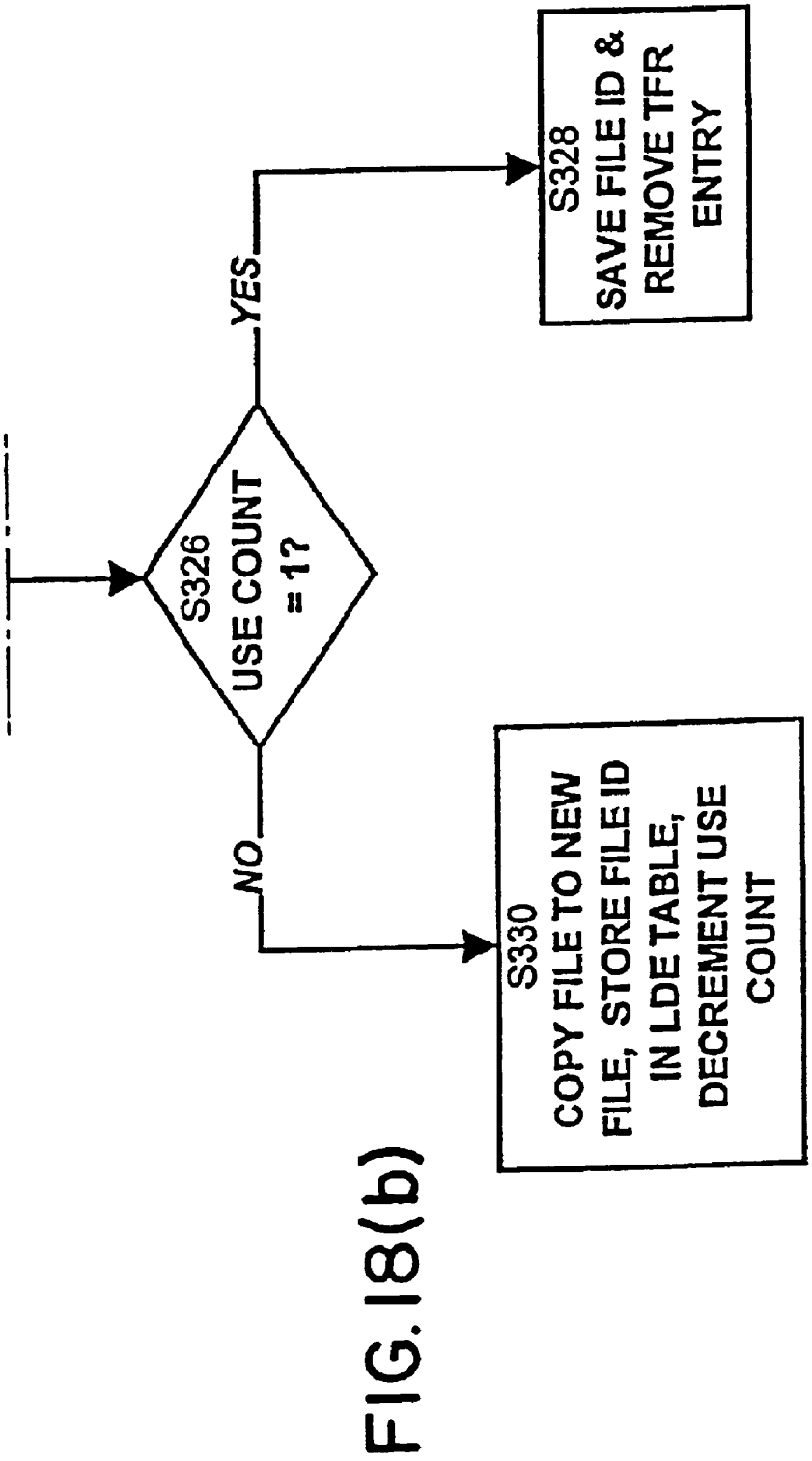

S326
USE COUNT
= 1?
YES
NO
S328
SAVE FILE ID &
REMOVE TFR
ENTRY
S330
COPY FILE TO NEW
FILE, STORE FILE ID
IN LDE TABLE,
DECREMENT USE
COUNT

FIG. 18(b)

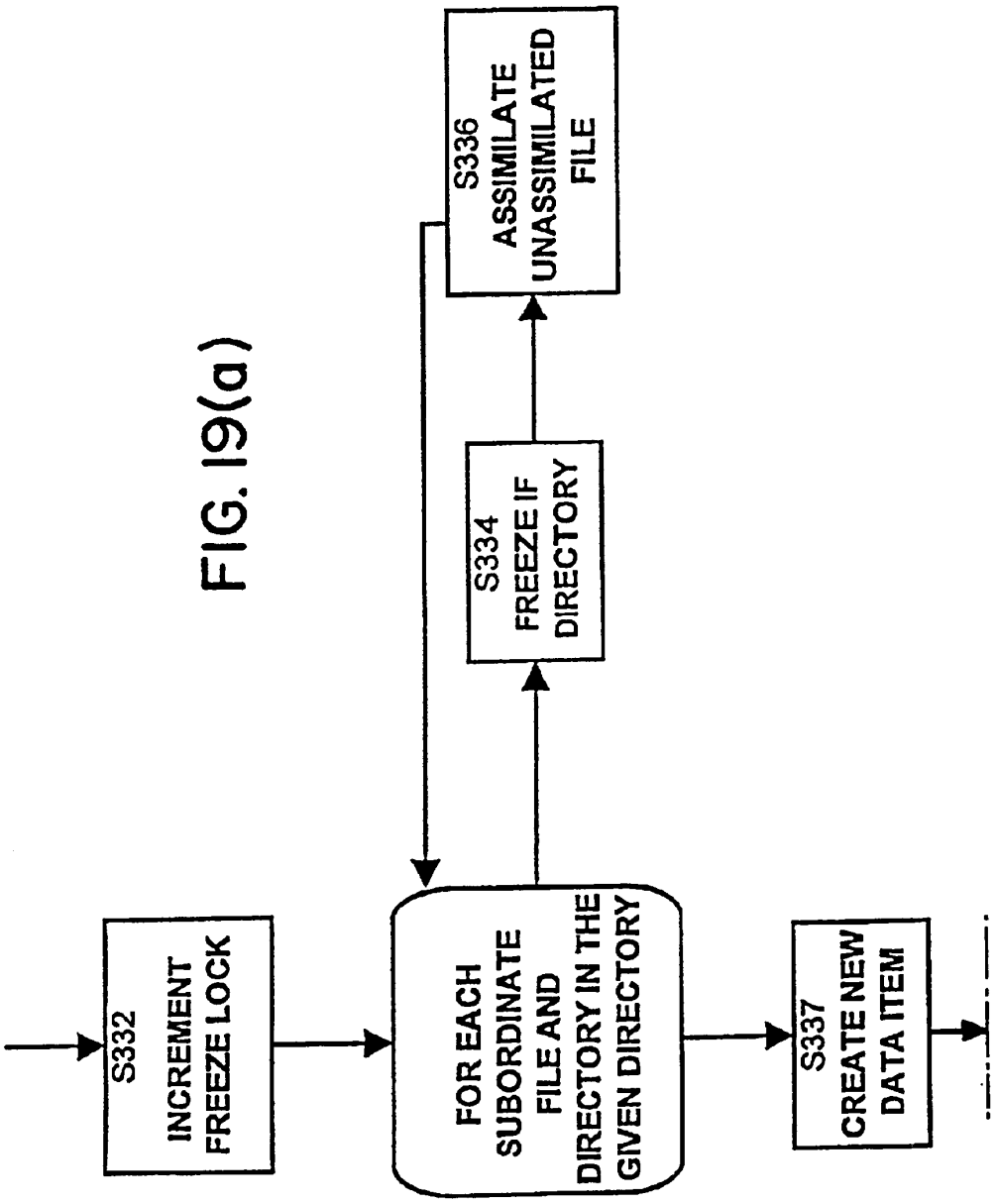

S332
INCREMENT FREEZE LOCK
FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S334
FREEZE IF DIRECTORY
S336
ASSIMILATE UNASSIMILATED FILE
S337
CREATE NEW DATA ITEM

FIG. 19(a)

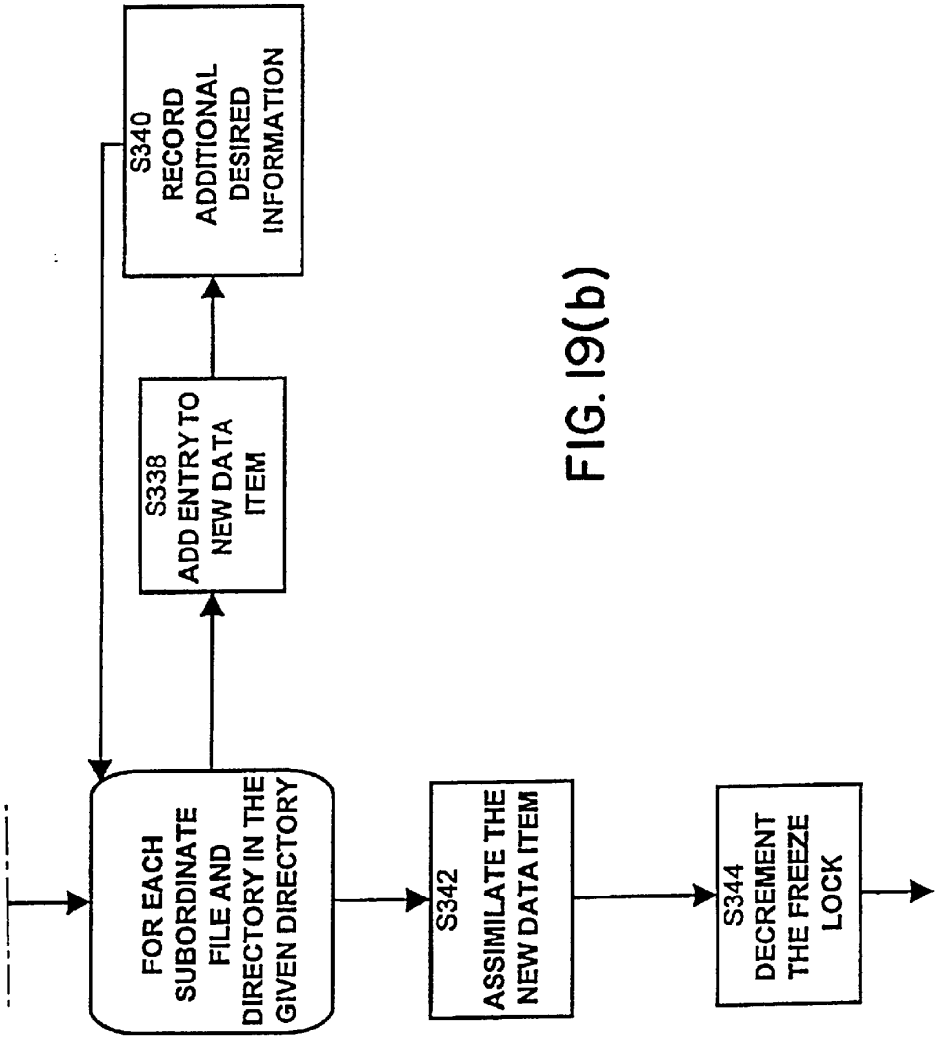

FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S338 ADD ENTRY TO NEW DATA ITEM
S340 RECORD ADDITIONAL DESIRED INFORMATION
S342 ASSIMILATE THE NEW DATA ITEM
S344 DECREMENT THE FREEZE LOCK

FIG. 19(b)

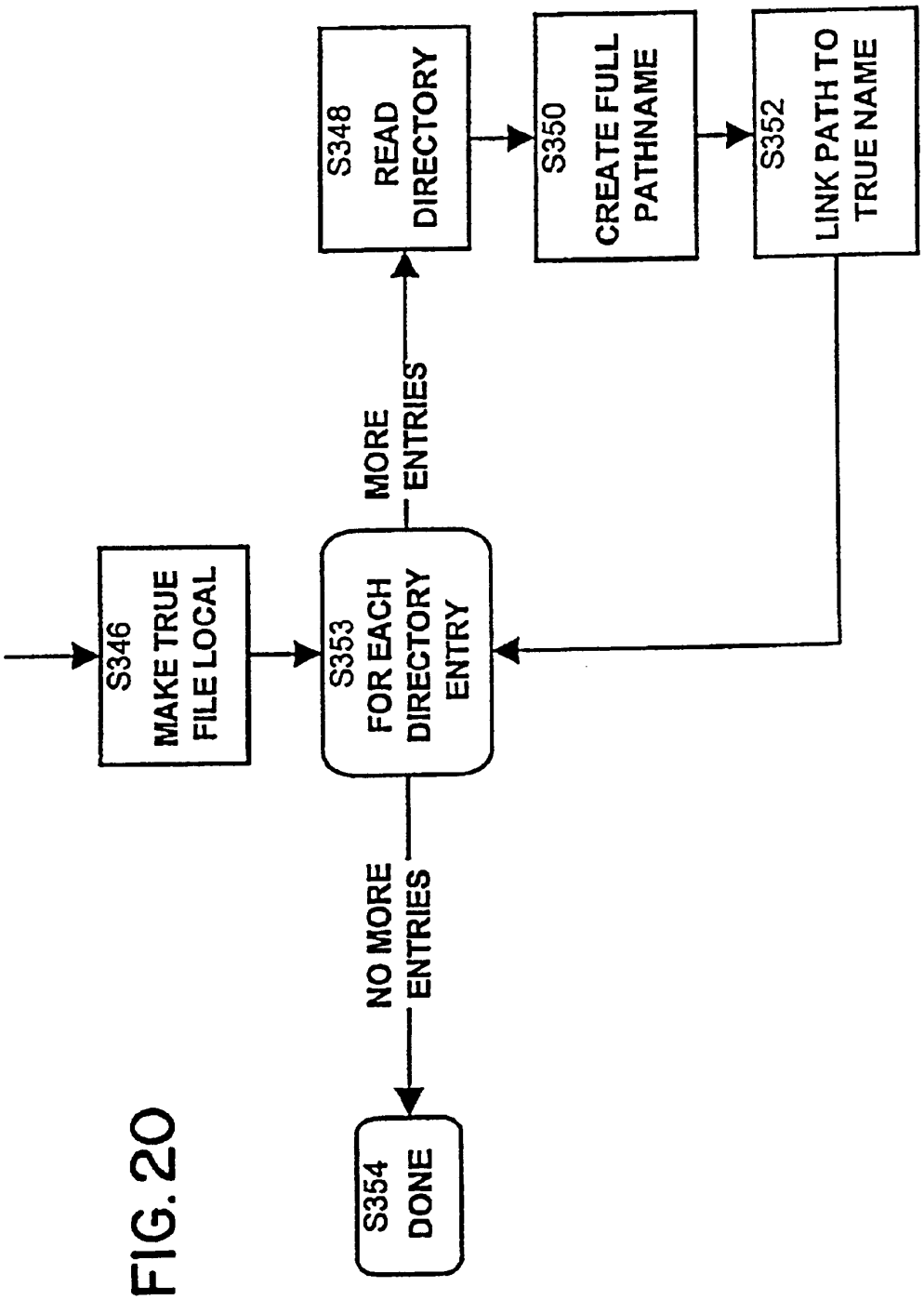

S346
MAKE TRUE FILE LOCAL
S353
FOR EACH DIRECTORY ENTRY
MORE ENTRIES
S348
READ DIRECTORY
S350
CREATE FULL PATHNAME
S352
LINK PATH TO TRUE NAME
NO MORE ENTRIES
S354
DONE

FIG. 20

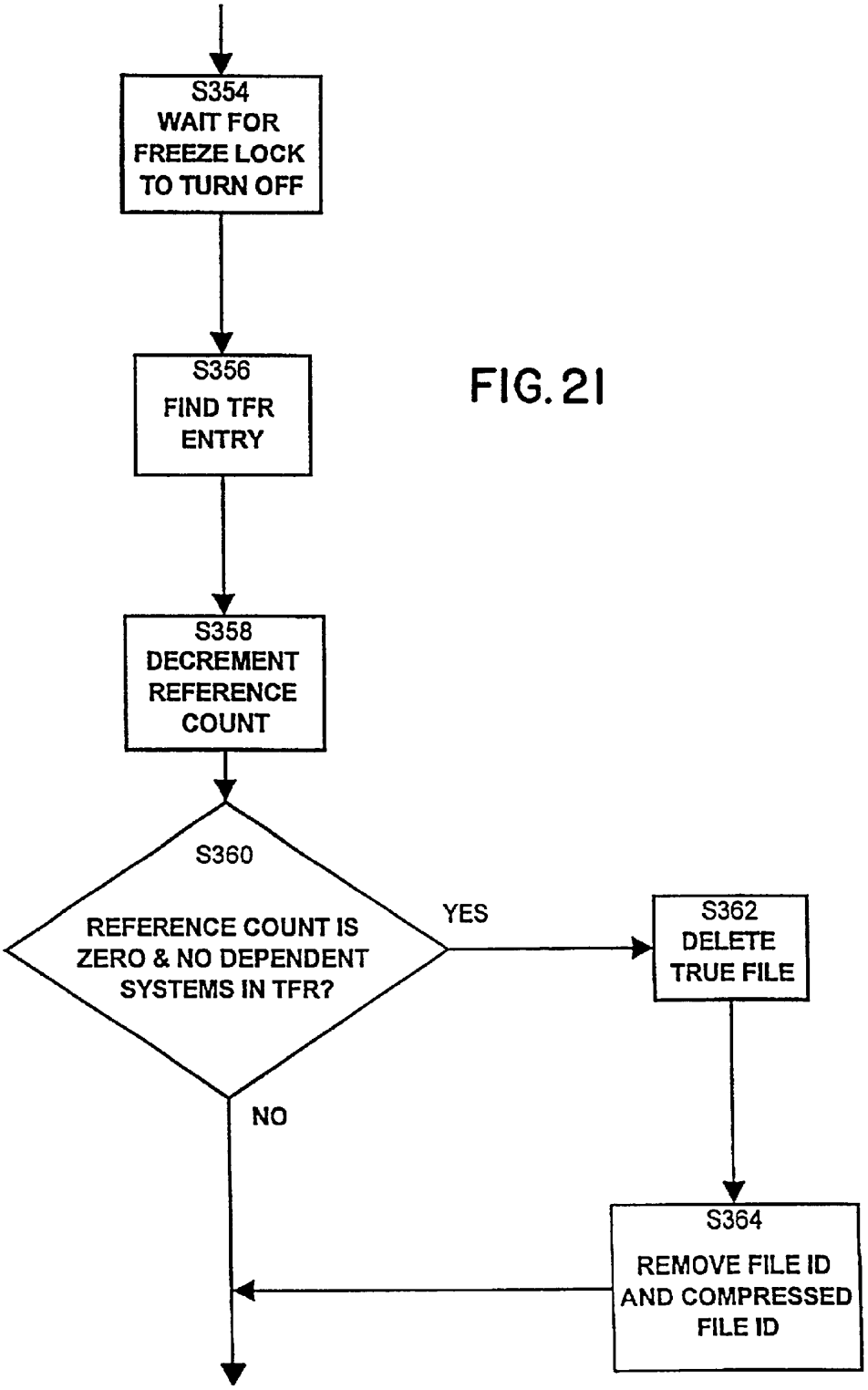

S354
WAIT FOR FREEZE LOCK TO TURN OFF
S356
FIND TFR ENTRY
S358
DECREMENT REFERENCE COUNT
S360
REFERENCE COUNT IS ZERO & NO DEPENDENT SYSTEMS IN TFR?
YES
NO
S362
DELETE TRUE FILE
S364
REMOVE FILE ID AND COMPRESSED FILE ID

FIG. 21

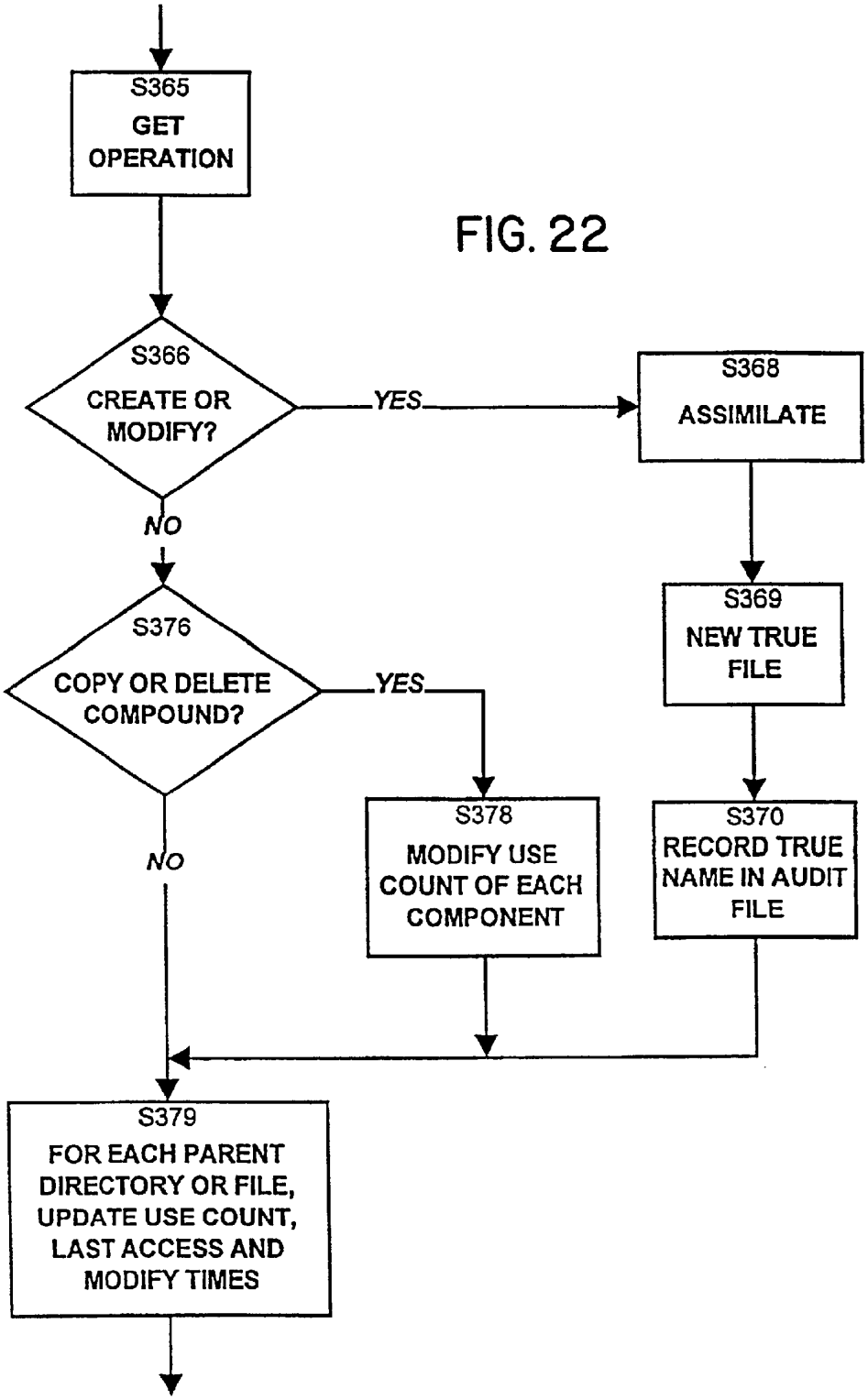
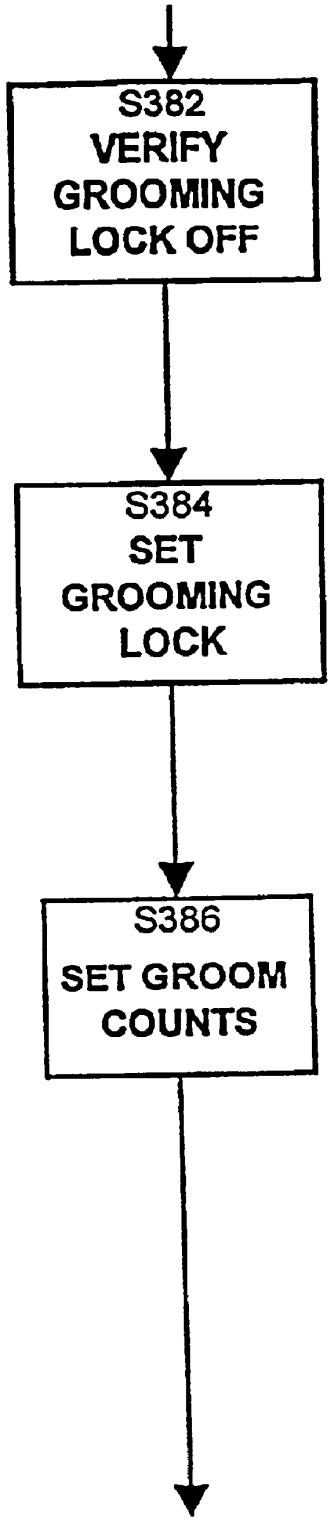

S365
GET OPERATION
S366
CREATE OR MODIFY?
YES
S368
ASSIMILATE
NO
S369
NEW TRUE FILE
S376
COPY OR DELETE COMPOUND?
YES
S378
MODIFY USE COUNT OF EACH COMPONENT
S370
RECORD TRUE NAME IN AUDIT FILE
NO
S379
FOR EACH PARENT DIRECTORY OR FILE, UPDATE USE COUNT, LAST ACCESS AND MODIFY TIMES

FIG. 22

FIG. 23


S382
VERIFY
GROOMING
LOCK OFF
S384
SET
GROOMING
LOCK
S386
SET GROOM
COUNTS

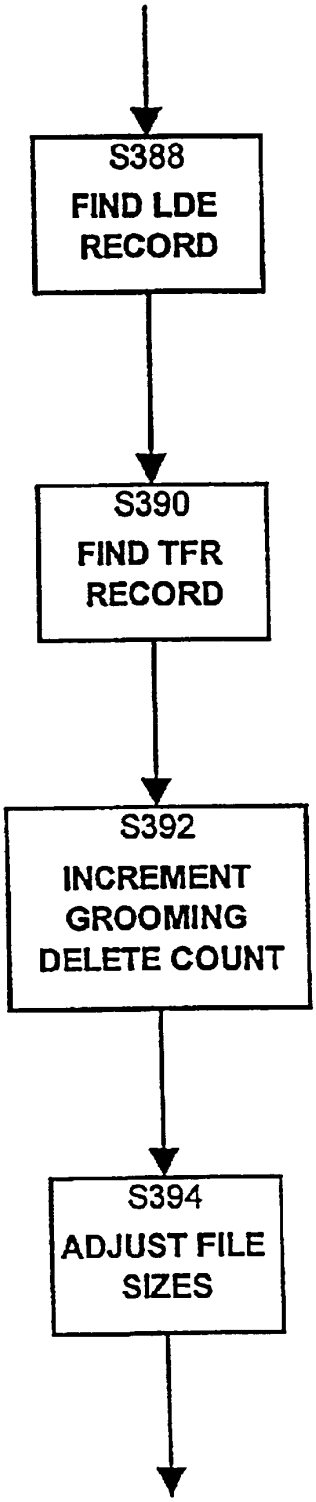

S388
FIND LDE RECORD
S390
FIND TFR RECORD
S392
INCREMENT GROOMING DELETE COUNT
S394
ADJUST FILE SIZES

FIG. 24

FIG. 25

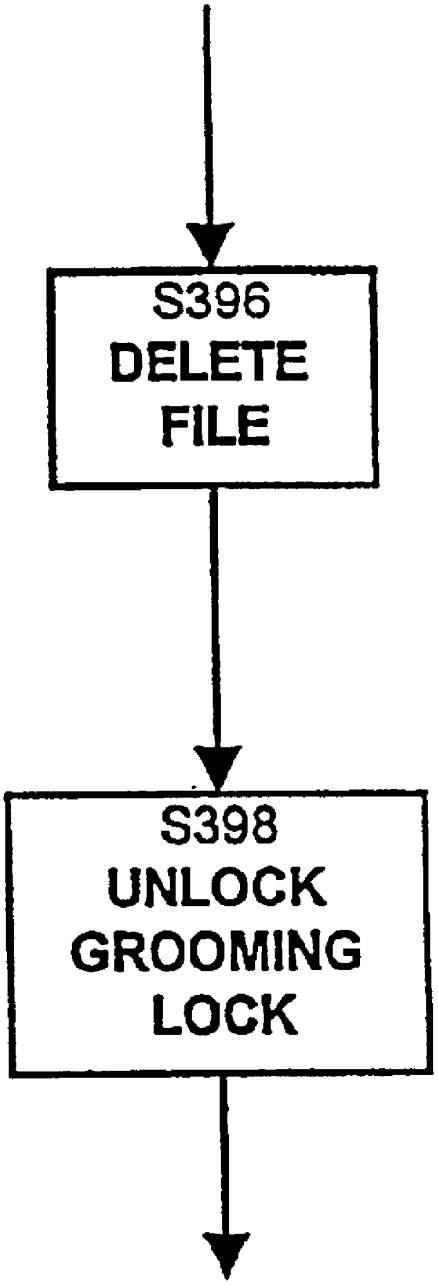
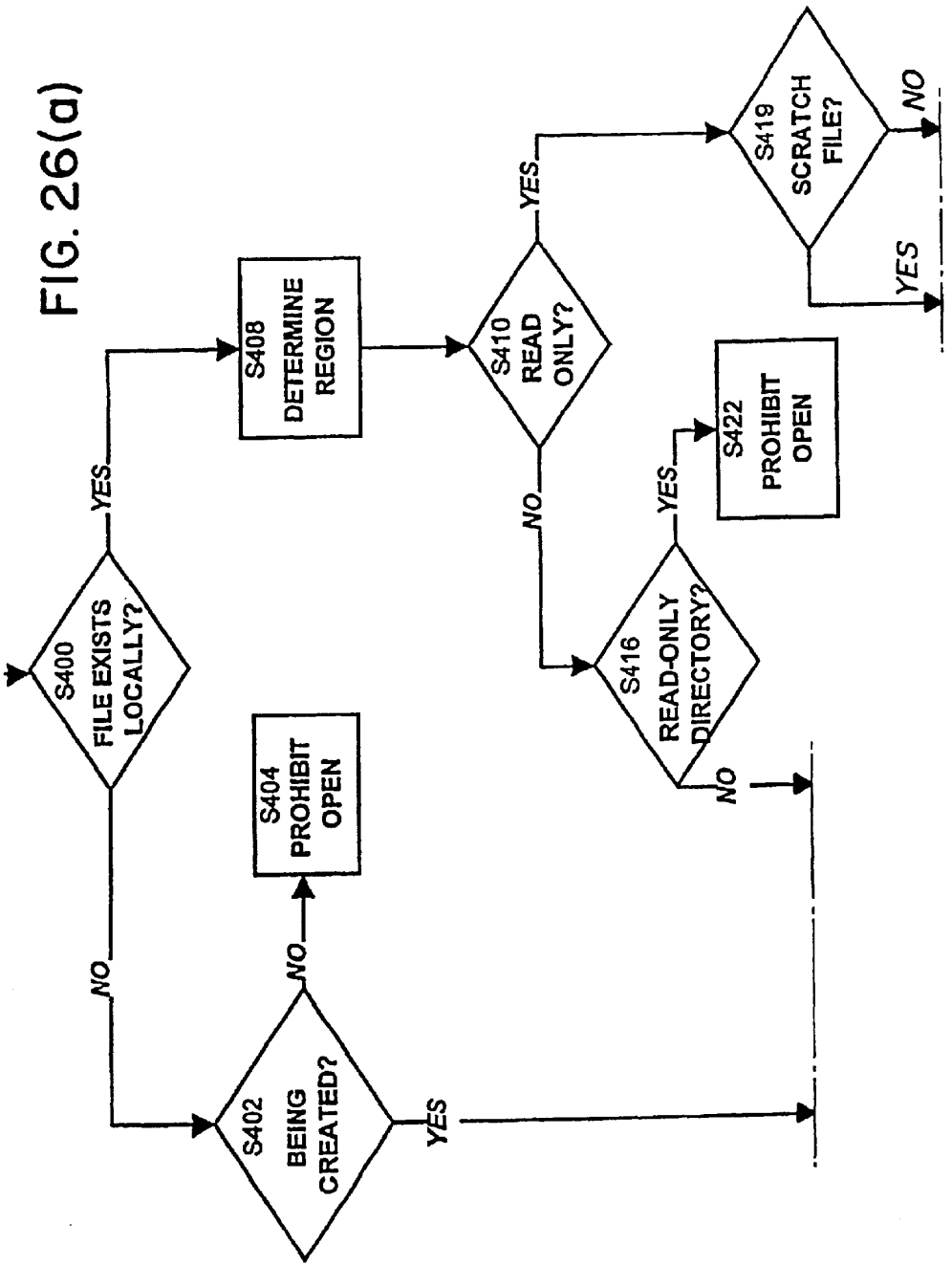


FIG. 26(a)


S400
FILE EXISTS LOCALLY?
YES
NO
S408
DETERMINE REGION
S410
READ ONLY?
YES
NO
S419
SCRATCH FILE?
NO
YES
S422
PROHIBIT OPEN
YES
S416
READ-ONLY DIRECTORY?
NO
S404
PROHIBIT OPEN
NO
S402
BEING CREATED?
YES

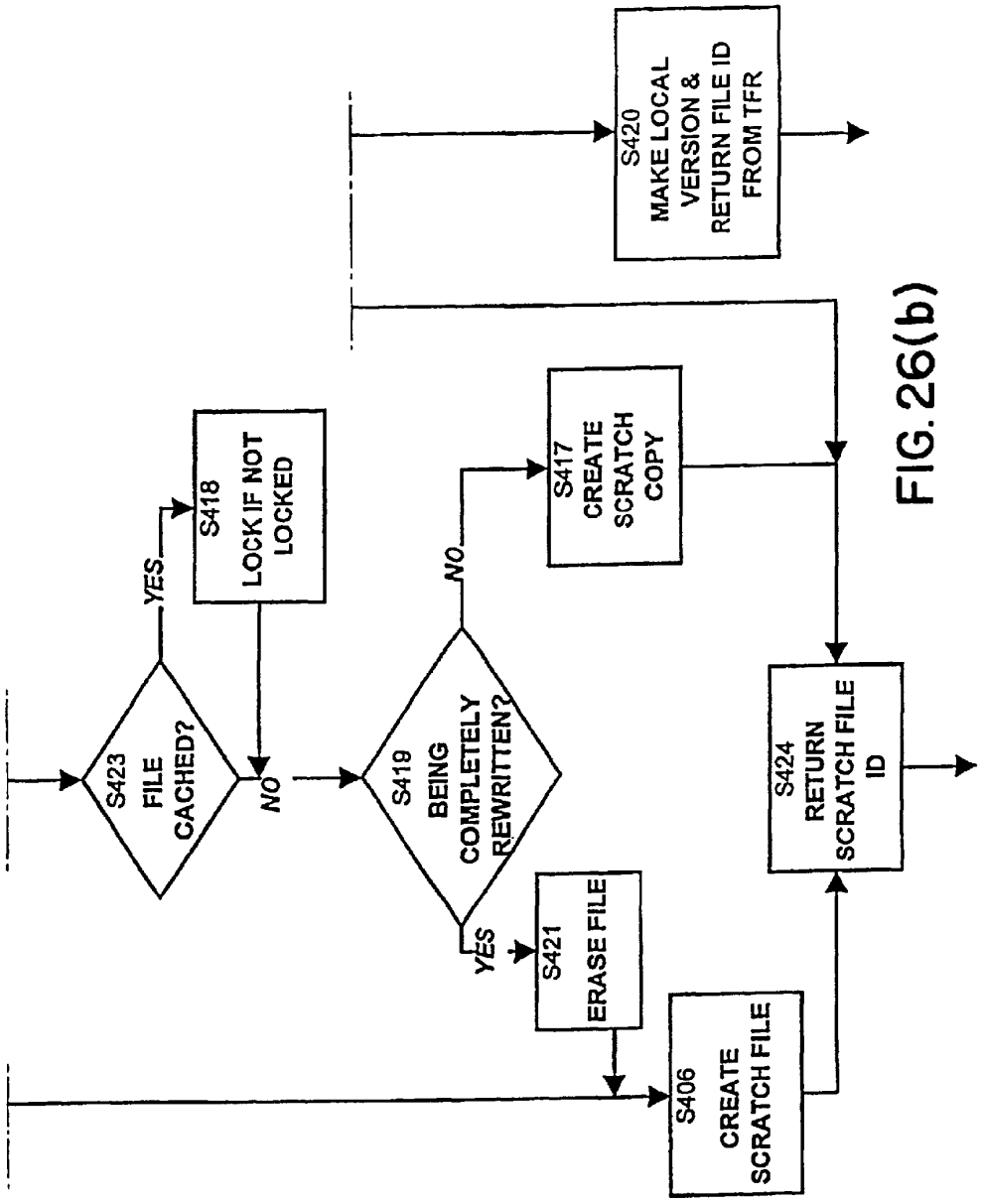

S420
MAKE LOCAL VERSION & RETURN FILE ID FROM TFR
S417
CREATE SCRATCH COPY
S418
LOCK IF NOT LOCKED
YES
NO
S423
FILE CACHED?
NO
S419
BEING COMPLETELY REWRITTEN?
YES
S421
ERASE FILE
S406
CREATE SCRATCH FILE
S424
RETURN SCRATCH FILE ID

FIG. 26(b)

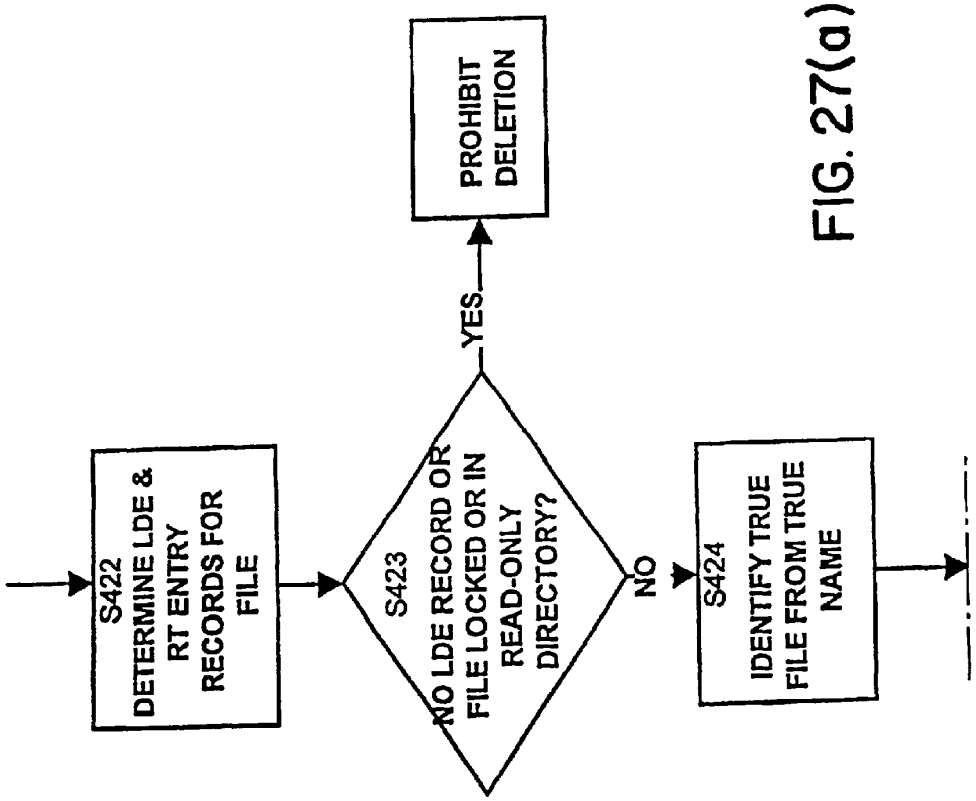

S422
DETERMINE LDE & RT ENTRY RECORDS FOR FILE
S423
NO LDE RECORD OR FILE LOCKED OR IN READ-ONLY DIRECTORY?
YES
PROHIBIT DELETION
NO
S424
IDENTIFY TRUE FILE FROM TRUE NAME

FIG. 27(a)

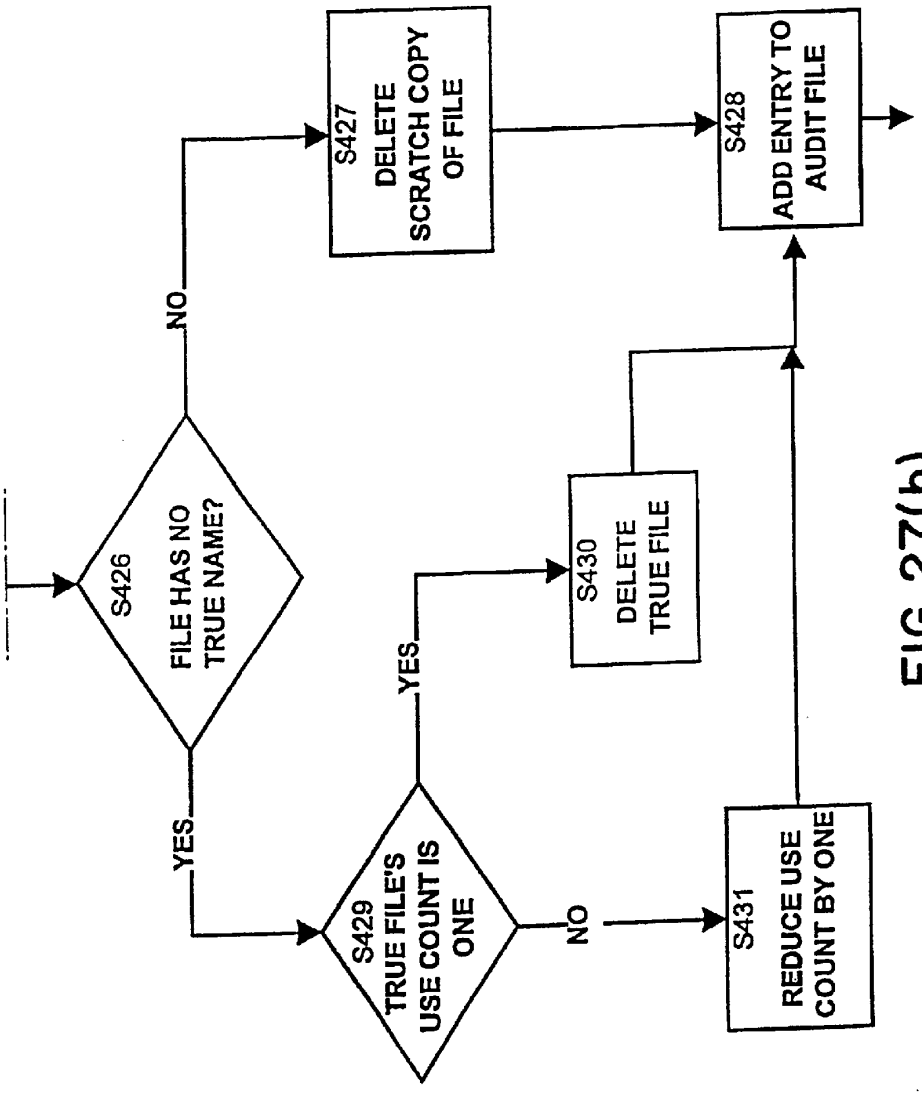
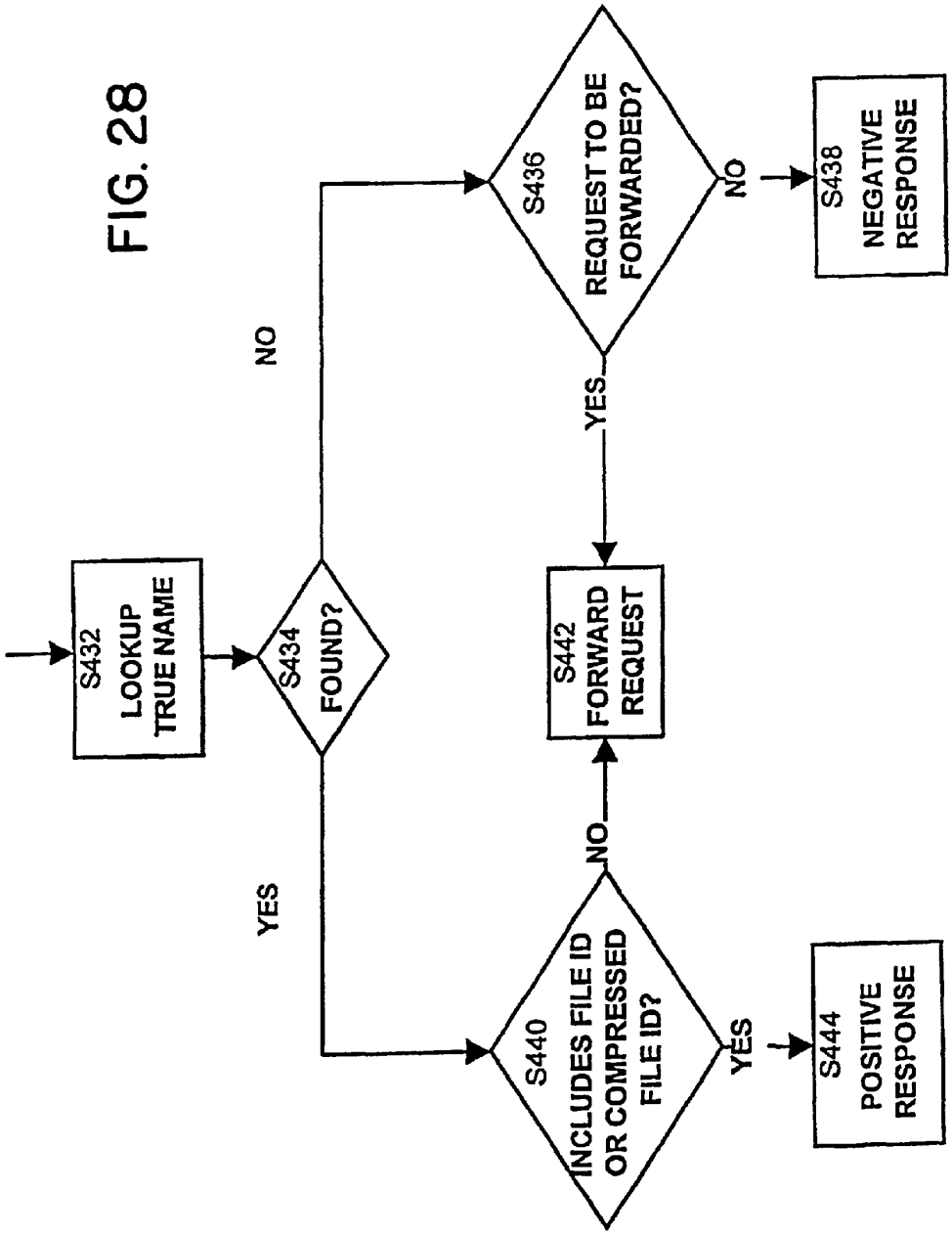

S426
FILE HAS NO TRUE NAME?
NO
YES
S427
DELETE SCRATCH COPY OF FILE
S428
ADD ENTRY TO AUDIT FILE
S429
TRUE FILE'S USE COUNT IS ONE
YES
NO
S430
DELETE TRUE FILE
S431
REDUCE USE COUNT BY ONE

FIG. 27(b)

FIG. 28


S432
LOOKUP TRUE NAME
S434
FOUND?
NO
YES
S436
REQUEST TO BE FORWARDED?
NO
S438
NEGATIVE RESPONSE
YES
S442
FORWARD REQUEST
YES
S440
INCLUDES FILE ID OR COMPRESSED FILE ID?
NO
YES
S444
POSITIVE RESPONSE

ENFORCEMENT AND POLICING OF LICENSED CONTENT USING CONTENT-BASED IDENTIFIERS

This is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791 filed Oct. 24, 2001 which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to data processing systems and, more particularly, to data processing systems wherein data items are identified by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items.

2. Background of the Invention

Data processing (DP) systems, computers, networks of computers, or the like, typically offer users and programs various ways to identify the data in the systems.

Users typically identify data in the data processing system by giving the data some form of name. For example, a typical operating system (OS) on a computer provides a file system in which data items are named by alphanumeric identifiers. Programs typically identify data in the data processing system using a location or address. For example, a program may identify a record in a file or database by using a record number which serves to locate that record.

In all but the most primitive operating systems, users and programs are able to create and use collections of named data items, these collections themselves being named by identifiers. These named collections can then, themselves, be made part of other named collections. For example, an OS may provide mechanisms to group files (data items) into directories (collections). These directories can then, themselves be made part of other directories. A data item may thus be identified relative to these nested directories using a sequence of names, or a so-called pathname, which defines a path through the directories to a particular data item (file or directory).

As another example, a database management system may group data records (data items) into tables and then group these tables into database files (collections). The complete address of any data record can then be specified using the database file name, the table name, and the record number of that data record.

Other examples of identifying data items include: identifying files in a network file system, identifying objects in an object-oriented database, identifying images in an image database, and identifying articles in a text database.

In general, the terms "data" and "data item" as used herein refer to sequences of bits. Thus a data item may be the contents of a file, a portion of a file, a page in memory, an object in an object-oriented program, a digital message, a digital scanned image, a part of a video or audio signal, or any other entity which can be represented by a sequence of bits. The term "data processing" herein refers to the processing of data items, and is sometimes dependent on the type of data item being processed. For example, a data processor for a digital image may differ from a data processor for an audio signal.

In all of the prior data processing systems the names or identifiers provided to identify data items (the data items

being files, directories, records in the database, objects in object-oriented programming, locations in memory or on a physical device, or the like) are always defined relative to a specific context. For instance, the file identified by a particular file name can only be determined when the directory containing the file (the context) is known. The file identified by a pathname can be determined only when the file system (context) is known. Similarly, the addresses in a process address space, the keys in a database table, or domain names on a global computer network such as the Internet are meaningful only because they are specified relative to a context.

In prior art systems for identifying data items there is no direct relationship between the data names and the data item. The same data name in two different contexts may refer to different data items, and two different data names in the same context may refer to the same data item.

In addition, because there is no correlation between a data name and the data it refers to, there is no a priori way to confirm that a given data item is in fact the one named by a data name. For instance, in a DP system, if one processor requests that another processor deliver a data item with a given data name, the requesting processor cannot, in general, verify that the data delivered is the correct data (given only the name). Therefore it may require further processing, typically on the part of the requester, to verify that the data item it has obtained is, in fact, the item it requested.

A common operation in a DP system is adding a new data item to the system. When a new data item is added to the system, a name can be assigned to it only by updating the context in which names are defined. Thus such systems require a centralized mechanism for the management of names. Such a mechanism is required even in a multiprocessing system when data items are created and identified at separate processors in distinct locations, and in which there is no other need for communication when data items are added.

In many data processing systems or environments, data items are transferred between different locations in the system. These locations may be processors in the data processing system, storage devices, memory, or the like. For example, one processor may obtain a data item from another processor or from an external storage device, such as a floppy disk, and may incorporate that data item into its system (using the name provided with that data item).

However, when a processor (or some location) obtains a data item from another location in the DP system, it is possible that this obtained data item is already present in the system (either at the location of the processor or at some other location accessible by the processor) and therefore a duplicate of the data item is created. This situation is common in a network data processing environment where proprietary software products are installed from floppy disks onto several processors sharing a common file server. In these systems, it is often the case that the same product will be installed on several systems, so that several copies of each file will reside on the common file server.

In some data processing systems in which several processors are connected in a network, one system is designated as a cache server to maintain master copies of data items, and other systems are designated as cache clients to copy local copies of the master data items into a local cache on an as-needed basis. Before using a cached item, a cache client must either reload the cached item, be informed of changes to the cached item, or confirm that the master item corre-

sponding to the cached item has not changed. In other words, a cache client must synchronize its data items with those on the cache server. This synchronization may involve reloading data items onto the cache client. The need to keep the cache synchronized or reload it adds significant overhead to existing caching mechanisms.

In view of the above and other problems with prior art systems, it is therefore desirable to have a mechanism which allows each processor in a multiprocessor system to determine a common and substantially unique identifier for a data item, using only the data in the data item and not relying on any sort of context.

It is further desirable to have a mechanism for reducing multiple copies of data items in a data processing system and to have a mechanism which enables the identification of identical data items so as to reduce multiple copies. It is further desirable to determine whether two instances of a data item are in fact the same data item, and to perform various other systems' functions and applications on data items without relying on any context information or properties of the data item.

It is also desirable to provide such a mechanism in such a way as to make it transparent to users of the data processing system, and it is desirable that a single mechanism be used to address each of the problems described above.

SUMMARY OF THE INVENTION

This invention provides, in a data processing system, a method and apparatus for identifying a data item in the system, where the identity of the data item depends on all of the data in the data item and only on the data in the data item. Thus the identity of a data item is independent of its name, origin, location, address, or other information not derivable directly from the data, and depends only on the data itself.

This invention further provides an apparatus and a method for determining whether a particular data item is present in the system or at a location in the system, by examining only the data identities of a plurality of data items.

Using the method or apparatus of the present invention, the efficiency and integrity of a data processing system can be improved. The present invention improves the design and operation of a data storage system, file system, relational database, object-oriented database, or the like that stores a plurality of data items, by making possible or improving the design and operation of at least some or all of the following features:

- the system stores at most one copy of any data item at a given location, even when multiple data names in the system refer to the same contents;
- the system avoids copying data from source to destination locations when the destination locations already have the data;
- the system provides transparent access to any data item by reference only to its identity and independent of its present location, whether it be local, remote, or offline;
- the system caches data items from a server, so that only the most recently accessed data items need be retained;
- when the system is being used to cache data items, problems of maintaining cache consistency are avoided;
- the system maintains a desired level of redundancy of data items in a network of servers, to protect against failure by ensuring that multiple copies of the data items are present at different locations in the system;
- the system automatically archives data items as they are created or modified;
- the system provides the-size, age, and location of groups of data items in order to decide whether they can be safely removed from a local file system;
- the system can efficiently record and preserve any collection of data items;
- the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items;
- the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of the data items;
- the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM, the inventory is independent of other properties of the data items such as their name, location, and date of creation;
- the system allows closely related sets of data items, such as matching or corresponding directories on disconnected computers, to be periodically resynchronized with one another;
- the system can verify that data retrieved from another location is the desired or requested data, using only the data identifier used to retrieve the data;
- the system can prove possession of specific data items by content without disclosing the content of the data items, for purposes of later legal verification and to provide anonymity;
- the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes.

Other objects, features, and characteristics of the present invention as well as the methods of operation and functions of the related elements of structure, and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1**(*a*) and **1**(*b*) depicts a typical data processing system in which a preferred embodiment of the present invention operates;

FIG. **2** depicts a hierarchy of data items stored at any location in such a data processing system;

FIGS. **3**–**9** depict data structures used to implement an embodiment of the present invention; and

FIGS. **10**(*a*)–**28** are flow charts depicting operation of various aspects of the present invention.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

An embodiment of the present invention is now described with reference to a typical data processing system **100**, which, with reference to FIGS. **1**(*a*) and **1**(*b*), includes one or more processors (or computers) **102** and various storage devices **104** connected in some way, for example by a bus **106**.

Each processor **102** includes a CPU **108**, a memory **110** and one or more local storage devices **112**. The CPU **108**,

memory **110**, and local storage device **112** may be internally connected, for example by a bus **114**. Each processor **102** may also include other devices (not shown), such as a keyboard, a display, a printer, and the like.

In a data processing system **100**, wherein more than one processor **102** is used, that is, in a multiprocessor system, the processors may be in one of various relationships. For example, two processors **102** may be in a client/server, client/client, or a server/server relationship. These inter-processor relationships may be dynamic, changing depending on particular situations and functions. Thus, a particular processor **102** may change its relationship to other processors as needed, essentially setting up a peer-to-peer relationship with other processors. In a peer-to-peer relationship, sometimes a particular processor **102** acts as a client processor, whereas at other times the same processor acts as a server processor. In other words, there is no hierarchy imposed on or required of processors **102**.

In a multiprocessor system, the processors **102** may be homogeneous or heterogeneous. Further, in a multiprocessor data processing system **100**, some or all of the processors **102** may be disconnected from the network of processors for periods of time. Such disconnection may be part of the normal operation of the system **100** or it may be because a particular processor **102** is in need of repair.

Within a data processing system **100**, the data may be organized to form a hierarchy of data storage elements, wherein lower level data storage elements are combined to form higher level elements. This hierarchy can consist of, for example, processors, file systems, regions, directories, data files, segments, and the like. For example, with reference to FIG. **2**, the data items on a particular processor **102** may be organized or structured as a file system **116** which comprises regions **117**, each of which comprises directories **118**, each of which can contain other directories **118** or files **120**. Each file **120** being made up of one or more data segments **122**.

In a typical data processing system, some or all of these elements can be named by users given certain implementation specific naming conventions, the name (or pathname) of an element being relative to a context. In the context of a data processing system **100**, a pathname is fully specified by a processor name, a filesystem name, a sequence of zero or more directory names identifying nested directories, and a final file name. (Usually the lowest level elements, in this case segments **122**, cannot be named by users.)

In other words, a file system **116** is a collection of directories **118**. A directory **118** is a collection of named files **120**—both data files **120** and other directory files **118**. A file **120** is a named data item which is either a data file (which may be simple or compound) or a directory file **118**. A simple file **120** consists of a single data segment **122**. A compound file **120** consists of a sequence of data segments **122**. A data segment **122** is a fixed sequence of bytes. An important property of any data segment is its size, the number of bytes in the sequence.

A single processor **102** may access one or more file systems **116**, and a single storage device **104** may contain one or more file systems **116**, or portions of a file system **116**. For instance, a file system **116** may span several storage devices **104**.

In order to implement controls in a file system, file system **116** may be divided into distinct regions, where each region is a unit of management and control. A region consists of a given directory **118** and is identified by the pathname (user defined) of the directory.

In the following, the term "location", with respect to a data processing system **100**, refers to any of a particular processor **102** in the system, a memory of a particular processor, a storage device, a removable storage medium (such as a floppy disk or compact disk), or any other physical location in the system. The term "local" with respect to a particular processor **102** refers to the memory and storage devices of that particular processor.

In the following, the terms "True Name", "data identity" and "data identifier" refer to the substantially unique data identifier for a particular data item. The term "True File" refers to the actual file, segment, or data item identified by a True Name.

A file system for a data processing system **100** is now described which is intended to work with an existing operating system by augmenting some of the operating system's file management system codes. The embodiment provided relies on the standard file management primitives for actually storing to and retrieving data items from disk, but uses the mechanisms of the present invention to reference and access those data items.

The processes and mechanisms (services) provided in this embodiment are grouped into the following categories: primitive mechanisms, operating system mechanisms, remote mechanisms, background mechanisms, and extended mechanisms.

Primitive mechanisms provide fundamental capabilities used to support other mechanisms. The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

Operating system mechanisms provide typical familiar file system mechanisms, while maintaining the data structures required to offer the mechanisms of the present invention. Operating system mechanisms are designed to augment existing operating systems, and in this way to make the present invention compatible with, and generally transparent to, existing applications. The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

Remote mechanisms are used by the operating system in responding to requests from other processors. These mecha-

nisms enable the capabilities of the present invention in a peer-to-peer network mode of operation. The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

Background mechanisms are intended to run occasionally and at a low priority. These provide automated management capabilities with respect to the present invention. The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links; and
4. Verify Region; and
5. Groom Source List.

Extended mechanisms run within application programs over the operating system. These mechanisms provide solutions to specific problems and applications. The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at location;
7. Verify True File;
8. Track for accounting purposes; and
9. Track for licensing purposes.

The file system herein described maintains sufficient information to provide a variety of mechanisms not ordinarily offered by an operating system, some of which are listed and described here. Various processing performed by this embodiment of the present invention will now be described in greater detail.

In some embodiments, some files **120** in a data processing system **100** do not have True Names because they have been recently received or created or modified, and thus their True Names have not yet been computed. A file that does not yet have a True Name is called a scratch file. The process of assigning a True Name to a file is referred to as assimilation, and is described later. Note that a scratch file may have a user provided name.

Some of the processing performed by the present invention can take place in a background mode or on a delayed or as-needed basis. This background processing is used to determine information that is not immediately required by the system or which may never be required. As an example, in some cases a scratch file is being changed at a rate greater than the rate at which it is useful to determine its True Name. In these cases, determining the True Name of the file can be postponed or performed in the background.

Data Structures

The following data structures, stored in memory **110** of one of more processors **102** are used to implement the mechanisms described herein. The data structures can be local to each processor **102** of the system **100**, or they can reside on only some of the processors **102**.

The data structures described are assumed to reside on individual peer processors **102** in the data processing system **100**. However, they can also be shared by placing them on a remote, shared file server (for instance, in a local area network of machines). In order to accommodate sharing data structures, it is necessary that the processors accessing the shared database use the appropriate locking techniques to ensure that changes to the shared database do not interfere with one another but are appropriately serialized. These locking techniques are well understood by ordinarily skilled programmers of distributed applications.

It is sometimes desirable to allow some regions to be local to a particular processor **102** and other regions to be shared among processors **102**. (Recall that a region is a unit of file system management and control consisting of a given directory identified by the pathname of the directory.) In the case of local and shared regions, there would be both local and shared versions of each data structure. Simple changes to the processes described below must be made to ensure that appropriate data structures are selected for a given operation.

The local directory extensions (LDE) table **124** is a data structure which provides information about files **120** and directories **118** in the data processing system **100**. The local directory extensions table **124** is indexed by a pathname or contextual name (that is, a user provided name) of a file and includes the True Name for most files. The information in local directory extension table **124** is in addition to that provided by the native file system of the operating system.

The True File registry (TFR) **126** is a data store for listing actual data items which have True Names, both files **120** and segments **122**. When such data items occur in the True File registry **126** they are known as True Files. True Files are identified in True File registry **126** by their True Names or identities. The table True File registry **126** also stores location, dependency, and migration information about True Files.

The region table (RT) **128** defines areas in the network storage which are to be managed separately. Region table **128** defines the rules for access to and migration of files **120** among various regions with the local file system **116** and remote peer file systems.

The source table (ST) **130** is a list of the sources of True Files other than the current True File registry **126**. The source table **130** includes removable volumes and remote processors.

The audit file (AF) **132** is a list of records indicating changes to be made in local or remote files, these changes to be processed in background.

The accounting log (AL) **134** is a log of file transactions used to create accounting information in a manner which preserves the identity of files being tracked independent of their name or location.

The license table (LT) **136** is a table identifying files, which may only be used by licensed users, in a manner independent of their name or location, and the users licensed to use them.

Detailed Descriptions of the Data Structures

The following table summarizes the fields of an local directory extensions table entry, as illustrated by record **138** in FIG. **3**.

Field	Description
Region ID	identifies the region in which this file is contained.

-continued

Field	Description
Pathname	the user provided name or contextual name of the file or directory, relative to the region in which it occurs.
True Name	the computed True Name or identity of the file or directory. This True Name is not always up to date, and it is set to a special value when a file is modified and is later recomputed in the background.
Type	indicates whether the file is a data file or a directory.
Scratch File ID	the physical location of the file in the file system, when no True Name has been calculated for the file. As noted above, such a file is called a scratch file.
Time of last access	the last access time to this file. If this file is a directory, this is the last access time to any file in the directory.
Time of last modification	the time of last change of this file. If this file is a directory, this is the last modification time of any file in the directory.
Safe flag	indicates that this file (and, if this file is a directory, all of its subordinate files) have been backed up on some other system, and it is therefore safe to remove them.
Lock flag	indicates whether a file is locked, that is, it is being modified by the local processor or a remote processor. Only one processor may modify a file at a time.
Size	the full size of this directory (including all subordinate files), if all files in it were fully expanded and duplicated. For a file that is not a directory this is the size of the actual True File.
Owner	the identity of the user who owns this file, for accounting and license tracking purposes.

Each record of the True File registry **126** has the fields shown in the True File registry record **140** in FIG. **4**. The True File registry **126** consists of the database described in the table below as well as the actual True Files identified by the True File IDs below.

Field	Description
True Name	computed True Name or identity of the file.
Compressed File ID	compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.
Grooming delete count	tentative count of how many references have been selected for deletion during a grooming operation.
Time of last access	most recent date and time the content of this file was accessed.
Expiration	date and time after which this file may be deleted by this server.
Dependent processors	processor IDs of other processors which contain references to this True File.
Source IDs	source ID(s) of zero or more sources from which this file or data item may be retrieved.
True File ID	identity or disk location of the actual physical representation of the file or file segment. It is sufficient to use a filename in the registration directory of the underlying operating system. The True File ID is absent if the actual file is not currently present at the current location.
Use count	number of other records on this processor which identify this True File.

A region table **128**, specified by a directory pathname, records storage policies which allow files in the file system to be stored, accessed and migrated in different ways. Storage policies are programmed in a configurable way using a set of rules described below.

Each region table record **142** of region table **128** includes the fields described in the following table (with reference to FIG. **5**):

Field	Description
Region ID	internally used identifier for this region.
Region file system	file system on the local processor of which this region is a part.
Region pathname	a pathname relative to the region file system which defines the location of this region. The region consists of all files and directories subordinate to this pathname, except those in a region subordinate to this region.
Mirror processor(s)	zero or more identifiers of processors which are to keep mirror or archival copies of all files in the current region. Multiple mirror processors can be defined to form a mirror group.
Mirror duplication count	number of copies of each file in this region that should be retained in a mirror group.
Region status	specifies whether this region is local to a single processor 102, shared by several processors 102 (if, for instance, it resides on a shared file server), or managed by a remote processor.
Policy	the migration policy to apply to this region. A single region might participate in several policies. The policies are as follows (parameters in brackets are specified as part of the policy):

region is a cached version from [processor ID];
region is a member of a mirror set defined by [processor ID].
region is to be archived on [processor ID].
region is to be backed up locally, by placing new copies in [region ID].
region is read only and may not be changed.
region is published and expires on [date].
Files in this region should be compressed.

A source table **130** identifies a source location for True Files. The source table **130** is also used to identify client processors making reservations on the current processor.

Each source record **144** of the source table **130** includes the fields summarized in the following table, with reference to FIG. **6**:

Field	Description
source ID	internal identifier used to identify a particular source.
source type	type of source location: Removable Storage Volume Local Region Cache Server Mirror Group Server Cooperative Server Publishing Server Client
source rights	includes information about the rights of this processor, such as whether it can ask the local processor to store data items for it.
source availability	measurement of the bandwidth, cost, and reliability of the connection to this source of True Files. The availability is used to select from among several possible sources.
source location	information on how the local processor is to access the source. This may be, for example, the name of a removable storage volume, or the processor ID and region path of a region on a remote processor.

The audit file **132** is a table of events ordered by timestamp, each record **146** in audit file **132** including the fields summarized in the following table (with reference to FIG. **7**):

Field	Description
Original Name	path of the file in question.
Operation	whether the file was created, read, written, copied or deleted. specifies whether the source is a file or a directory.
Processor ID	ID of the remote processor generating this event (if not local).
Timestamp	time and date file was closed (required only for accessed/modified files).
Pathname	Name of the file (required only for rename).
True Name	computed True Name of the file. This is used by remote systems to mirror changes to the directory and is filled in during background processing.

Each record **148** of the accounting log **134** records an event which may later be used to provide information for billing mechanisms. Each accounting log entry record **148** includes at least the information summarized in the following table, with reference to FIG. **8**:

Field	Description
date of entry	date and time of this log entry.
type of entry	Entry types include create file, delete file, and transmit file.
True Name	True Name of data item in question.
owner	identity of the user responsible for this action.

Each record **150** of the license table **136** records a relationship between a licensable data item and the user licensed to have access to it. Each license table record **150** includes the information summarized in the following table "with reference to FIG. **9**:

Field	Description
True Name	True Name of a data item subject to license validation.
licensee	identity of a user authorized to have access to this object.

Various other data structures are employed on some or all of the processors **102** in the data processing system **100**. Each processor **102** has a global freeze lock (GFL) **152** (FIG. **1**), which is used to prevent synchronization errors when a directory is frozen or copied. Any processor **102** may include a special archive directory (SAD) **154** into which directories may be copied for the purposes of archival. Any processor **102** may include a special media directory (SMD) **156**, into which the directories of removable volumes are stored to form a media inventory. Each processor has a grooming lock **158**, which is set during a grooming operation. During this period the grooming delete count of True File registry entries **140** is active, and no True Files should be deleted until grooming is complete. While grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

Primitive Mechanisms

The first of the mechanisms provided by the present invention, primitive mechanisms, are now described. The mechanisms described here depend on underlying data management mechanisms to create, copy, read, and delete data items in the True File registry **126**, as identified by a True File ID. This support may be provided by an underlying operating system or disk storage manager.

The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link Path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

1. Calculate True Name

A True Name is computed using a function, MD, which reduces a data block B of arbitrary length to a relatively small, fixed size identifier, the True Name of the data block, such that the True Name of the data block is virtually guaranteed to represent the data block B and only data block B.

The function MD must have the following properties:

1. The domain of the function MD is the set of all data items. The range of the function MD is the set of True Names.

2. The function MD must take a data item of arbitrary length and reduce it to an integer value in the range 0 to N−1, where N is the cardinality of the set of True Names. That is, for an arbitrary length data block B, 0<MD(B)<N.
3. The results of MD(B) must be evenly and randomly distributed over the range of N, in such a way that simple or regular changes to B are virtually guaranteed to produce a different value of MD(B).
4. It must be computationally difficult to find a different value B' such that MD(B)=MD(B').
5. The function MD(B) must be efficiently computed.

A family of functions with the above properties are the so-called message digest functions, which are used in digital security systems as techniques for authentification of data. These functions (or algorithms) include MD4, MD5, and SHA.

In the presently preferred embodiments, either MD5 or SHA is employed as the basis for the computation of True Names. Whichever of these two message digest functions is employed, that same function must be employed on a system-wide basis.

It is impossible to define a function having a unique output for each possible input when the number of elements in the range of the function is smaller than the number of elements in its domain. However, a crucial observation is that the actual data items that will be encountered in the operation of any system embodying this invention form a very sparse subset of all the possible inputs.

A colliding set of data items is defined as a set wherein, for one or more pairs x and y in the set, MD(x)=MD(y). Since a function conforming to the requirements for MD must evenly and randomly distribute its outputs, it is possible, by making the range of the function large enough, to make the probability arbitrarily small that actual inputs encountered in the operation of an embodiment of this invention will form a colliding set.

To roughly quantify the probability of a collision, assume that there are no more than 2^{30} storage devices in the world, and that each storage device has an average of at most 2^{20} different data items. Then there are at most 2^{50} data items in the world. If the outputs of MD range between 0 and 2^{128}, it can be demonstrated that the probability of a collision is approximately 1 in 2^{29}. Details on the derivation of these probability values are found, for example, in P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," *Lecture Notes in Computer Science* 434: *Advances in Cryptology—Eurocrypt '89 Proceedings*, Springer-Verlag, pp. 329–354.

Note that for some less preferred embodiments of the present invention, lower probabilities of uniqueness may be acceptable, depending on the types of applications and mechanisms used. In some embodiments it may also be useful to have more than one level of True Names, with some of the True Names having different degrees of uniqueness. If such a scheme is implemented, it is necessary to ensure that less unique True Names are not propagated in the system.

While the invention is described herein using only the True Name of a data item as the identifier for the data item, other preferred embodiments use tagged, typed, categorized or classified data items and use a combination of both the True Name and the tag, type, category or class of the data item as an identifier. Examples of such categorizations are files, directories, and segments; executable files and data files, and the like. Examples of classes are classes of objects in an object-oriented system. In such a system, a lower degree of True Name uniqueness is acceptable over the entire universe of data items, as long as sufficient uniqueness is provided per category of data items. This is because the tags provide an additional level of uniqueness.

A mechanism for calculating a True Name given a data item is now described, with reference to FIGS. **10**(*a*) and **10**(*b*).

A simple data item is a data item whose size is less than a particular given size (which must be defined in each particular implementation of the invention). To determine the True Name of a simple data item, with reference to FIG. **10**(*a*), first compute the MD function (described above) on the given simple data item (step S**212**). Then append to the resulting 128 bits, the byte length modulo **32** of the data item (Step S**214**). The resulting 160-bit value is the True Name of the simple data item.

A compound data item is one whose size is greater than the particular given size of a simple data item. To determine the True Name of an arbitrary (simple or compound) data item, with reference to FIG. **10**(*b*), first determine if the data item is a simple or a compound data item (Step S**216**). If the data item is a simple data item, then compute its True Name in step S**218** (using steps S**212** and S**214** described above), otherwise partition the data item into segments (Step S**220**) and assimilate each segment (Step S**222**) (the primitive mechanism, Assimilate a Data Item, is described below), computing the True Name of the segment. Then create an indirect block consisting of the computed segment True Names (Step S**224**). An indirect block is a data item which consists of the sequence of True Names of the segments. Then, in step S**226**, assimilate the indirect block and compute its True Name. Finally, replace the final thirty-two (32) bits of the resulting True Name (that is, the length of the indirect block) by the length modulo **32** of the compound data item (Step S**228**). The result is the True Name of the compound data item.

Note that the compound data item may be so large that the indirect block of segment True Names is itself a compound data item. In this case the mechanism is invoked recursively until only simple data items are being processed.

Both the use of segments and the attachment of a length to the True Name are not strictly required in a system using the present invention, but are currently considered desirable features in the preferred embodiment.

2. Assimilate Data Item

A mechanism for assimilating a data item (scratch file or segment) into a file system, given the scratch file ID of the data item, is now described with reference to FIG. **11**. The purpose of this mechanism is to add a given data item to the True File registry **126**. If the data item already exists in the True File registry **126**, this will be discovered and used during this process, and the duplicate will be eliminated.

Thereby the system stores at most one copy of any data item or file by content, even when multiple names refer to the same content.

First, determine the True Name of the data item corresponding to the given scratch File ID using the Calculate True Name primitive mechanism (Step S**230**). Next, look for an entry for the True Name in the True File registry **126** (Step S**232**) and determine whether a True Name entry, record **140**, exists in the True File registry **126**. If the entry record includes a corresponding True File ID or compressed File ID (Step S**237**), delete the file with the scratch File ID (Step S**238**). Otherwise store the given True File ID in the entry record (step S**239**).

If it is determined (in step S**232**) that no True Name entry exists in the True File registry **126**, then, in Step S**236**, create a new entry in the True File registry **126** for this True Name.

Set the True Name of the entry to the calculated True Name, set the use count for the new entry to one, store the given True File ID in the entry and set the other fields of the entry as appropriate.

Because this procedure may take some time to compute, it is intended to run in background after a file has ceased to change. In the meantime, the file is considered an unassimilated scratch file.

3. New True File

The New True File process is invoked when processing the audit file **132**, some time after a True File has been assimilated (using the Assimilate Data Item primitive mechanism). Given a local directory extensions table entry record **138** in the local directory extensions table **124**, the New True File process can provide the following steps (with reference to FIG. **12**), depending on how the local processor is configured:

First, in step S**238**, examine the local directory extensions table entry record **138** to determine whether the file is locked by a cache server. If the file is locked, then add the ID of the cache server to the dependent processor list of the True File registry table **126**, and then send a message to the cache server to update the cache of the current processor using the Update Cache remote mechanism (Step **242**).

If desired, compress the True File (Step S**246**), and, if desired, mirror the True File using the Mirror True File background mechanism (Step S**248**).

4. Get True Name from Path

The True Name of a file can be used to identify a file by contents, to confirm that a file matches its original contents, or to compare two files. The mechanism to get a True Name given the pathname of a file is now described with reference to FIG. **13**.

First, search the local directory extensions table **124** for the entry record **138** with the given pathname (Step S**250**). If the pathname is not found, this process fails and no True Name corresponding to the given pathname exists. Next, determine whether the local directory extensions table entry record **138** includes a True Name (Step S**252**), and if so, the mechanism's task is complete. Otherwise, determine whether the local directory extensions table entry record **138** identifies a directory (Step **5254**), and if so, freeze the directory (Step S**256**) (the primitive mechanism Freeze Directory is described below).

Otherwise, in step S**258**, assimilate the file (using the Assimilate Data Item primitive mechanism) defined by the File ID field to generate its True Name and store its True Name in the local directory extensions entry record. Then return the True Name identified by the local directory extensions table **124**.

5. Link Path to True Name

The mechanism to link a path to a True Name provides a way of creating a new directory entry record identifying an existing, assimilated file. This basic process may be used to copy, move, and rename files without a need to copy their contents. The mechanism to link a path to a True Name is now described with reference to FIG. **14**.

First, if desired, confirm that the True Name exists locally by searching for it in the True Name registry or local directory extensions table **135** (Step S**260**). Most uses of this mechanism will require this form of validation. Next, search for the path in the local directory extensions table **135** (Step S**262**). Confirm that the directory containing the file named in the path already exists (Step S**264**). If the named file itself exists, delete the File using the Delete True File operating system mechanism (see below) (Step S**268**).

Then, create an entry record in the local directory extensions with the specified path (Step S**270**) and update the entry record and other data structures as follows: fill in the True Name field of the entry with the specified True Name; increment the use count for the True File registry entry record **140** of the corresponding True Name; note whether the entry is a directory by reading the True File to see if it contains a tag (magic number) indicating that it represents a frozen directory (see also the description of the Freeze Directory primitive mechanism regarding the tag); and compute and set the other fields of the local directory extensions appropriately. For instance, search the region table **128** to identify the region of the path, and set the time of last access and time of last modification to the current time.

6. Realize True File from Location

This mechanism is used to try to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. This mechanism is now described with reference to FIG. **15**.

First, in step S**272**, determine whether the location specified is a processor. If it is determined that the location specified is a processor, then send a Request True File message (using the Request True File remote mechanism) to the remote processor and wait for a response (Step S**274**). If a negative response is received or no response is received after a timeout period, this mechanism fails. If a positive response is received, enter the True File returned in the True File registry **126** (Step S**276**). (If the file received was compressed, enter the True File ID in the compressed File ID field.)

If, on the other hand, it is determined in step S**272** that the location specified is not a processor, then, if necessary, request the user or operator to mount the indicated volume (Step S**278**). Then (Step S**280**) find the indicated file on the given volume and assimilate the file using the Assimilate Data Item primitive mechanism. If the volume does not contain a True File registry **126**, search the media inventory to find the path of the file on the volume. If no such file can be found, this mechanism fails.

At this point, whether or not the location is determined (in step S**272**) to be a processor, if desired, verify the True File (in step S**282**).

7. Locate Remote File

This mechanism allows a processor to locate a file or data item from a remote source of True Files, when a specific source is unknown or unavailable. A client processor system may ask one of several or many sources whether it can supply a data object with a given True Name. The steps to perform this mechanism are as follows (with reference to FIGS. **16**(*a*) and **16**(*b*)).

The client processor **102** uses the source table **145** to select one or more source processors (Step S**284**). If no source processor can be found, the mechanism fails. Next, the client processor **102** broadcasts to the selected sources a request to locate the file with the given True Name using the Locate True File remote mechanism (Step S**286**). The request to locate may be augmented by asking to propagate this request to distant servers. The client processor then waits for one or more servers to respond positively (Step S**288**). After all servers respond negatively, or after a timeout period with no positive response, the mechanism repeats selection (Step S**284**) to attempt to identify alternative sources. If any selected source processor responds, its processor ID is the result of this mechanism. Store the processor ID in the source field of the True File registry entry record **140** of the given True Name (Step S**290**).

If the source location of the True Name is a different processor or medium than the destination (Step S**290***a*), perform the following steps:

(i) Look up the True File registry entry record **140** for the corresponding True Name, and add the source location ID to the list of sources for the True Name (Step S**290***b*); and

(ii) If the source is a publish ing system, determine the expiration date on the publishing system for th e True Name and add that to the list of sources. If the source is not a publishing system, send a message to reserve the True File on the source processor (Step S**290***c*).

Source selection in step S**284** may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost, and ignoring previously selected processors which did not respond in step S**288**.

8. Make True File Local

This mechanism is used when a True Name is known and a locally accessible copy of the corresponding file or data item is required. This mechanism makes it possible to actually read the data in a True File. The mechanism takes a True Name and returns when there is a local, accessible copy of the True File in the True File registry **126**. This mechanism is described here with reference to the flow chart of FIGS. **17**(*a*) and **17**(*b*).

First, look in the True file registry **126** for a True File entry record **140** for the corresponding True Name (Step S**292**). If no such entry is found this mechanism fails. If there is already a True File ID for the entry (Step S**294**), this mechanism's task is complete. If there is a compressed file ID for the entry (Step S**296**), decompress the file corresponding to the file ID (Step S**296**) and store the decompressed file ID in the entry (Step S**300**). This mechanism is then complete.

If there is no True File ID for the entry (Step S**294**) and there is no compressed file ID for the entry (Step S**296**), then continue searching for the requested file. At this time it may be necessary to notify the user that the system is searching for the requested file.

If there are one or more source IDs, then select an order in which to attempt to realize the source ID (Step S**304**). The order may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost. For each source in the order chosen, realize the True File from the source location (using the Realize True File from Location primitive mechanism), until the True File is realized (Step S**306**). If it is realized, continue with step S**294**. If no known source can realize the True File, use the Locate Remote File primitive mechanism to attempt to find the True File (Step S**308**). If this succeeds, realize the True File from the identified source location and continue with step S**296**.

9. Create Scratch File

A scratch copy of a file is required when a file is being created or is about to be modified. The scratch copy is stored in the file system of the underlying operating system. The scratch copy is eventually assimilated when the audit file record entry **146** is processed by the Process Audit File Entry primitive mechanism. This Create Scratch File mechanism requires a local directory extensions table entry record **138**. When it succeeds, the local directory extensions table entry record **138** contains the scratch file ID of a scratch file that is not contained in the True File registry **126** and that may be modified. This mechanism is now described with reference to FIGS. **18**(*a*) and **18**(*b*).

First determine whether the scratch file should be a copy of the existing True File (Step S**310**). If so, continue with step S**312**. Otherwise, determine whether the local directory extensions table entry record **138** identifies an existing True File (Step S**316**), and if so, delete the True File using the Delete True File primitive mechanism (Step S**318**). Then create a new, empty scratch file and store its scratch file ID in the local directory extensions table entry record **138** (Step S**320**). This mechanism is then complete.

If the local directory extensions table entry record **138** identifies a scratch file ID (Step S**312**), then the entry already has a scratch file, so this mechanism succeeds.

If the local directory extensions table entry record **138** identifies a True File (S**316**), and there is no True File ID for the True File (S**312**), then make the True File local using the Make True File Local primitive mechanism (Step S**322**). If there is still no True File ID, this mechanism fails.

There is now a local True File for this file. If the use count in the corresponding True File registry entry record **140** is one (Step S**326**), save the True File ID in the scratch file ID of the local directory extensions table entry record **138**, and remove the True File registry entry record **140** (Step S**328**). (This step makes the True File into a scratch file.) This mechanism's task is complete.

Otherwise, if the use count in the corresponding True File registry entry record **140** is not one (in step S**326**), copy the file with the given True File ID to a new scratch file, using the Read File OS mechanism and store its file ID in the local directory extensions table entry record **138** (step S**330**), and reduce the use count for the True File by one. If there is insufficient space to make a copy, this mechanism fails.

10. Freeze Directory

This mechanism freezes a directory in order to calculate its True Name. Since the True Name of a directory is a function of the files within the directory, they must not change during the computation of the True Name of the directory. This mechanism requires the pathname of a directory to freeze. This mechanism is described with reference to FIGS. **19**(*a*) and **19**(*b*).

In step S**332**, add one to the global freeze lock. Then search the local directory extensions table **124** to find each subordinate data file and directory of the given directory, and freeze each subordinate directory found using the Freeze Directory primitive mechanism (Step S**334**). Assimilate each unassimilated data file in the directory using the Assimilate Data Item primitive mechanism (Step S**336**). Then create a data item which begins with a tag or marker (a "magic number") being a unique data item indicating that this data item is a frozen directory (Step S**337**). Then list the file name and True Name for each file in the current directory (Step s**338**). Record any additional information required, such as the type, time of last access and modification, and size (Step S**340**). Next, in step S**342**, using the Assimilate Data Item primitive mechanism, assimilate the data item created in step S**338**. The resulting True Name is the True Name of the frozen directory. Finally, subtract one from the global freeze lock (Step S**344**).

11. Expand Frozen Directory

This mechanism expands a frozen directory in a given location. It requires a given pathname into which to expand the directory, and the True Name of the directory and is described with reference to FIG. **20**.

First, in step S**346**, make the True File with the given True Name local using the Make True File Local primitive mechanism. Then read each directory entry in the local file created in step S**346** (Step S**348**). For each such directory entry, do the following:

Create a full pathname using the given pathname and the file name of the entry (Step S**350**); and

link the created path to the True Name (Step S**352**) using the Link Path to True Name primitive mechanism.

12. Delete True File

This mechanism deletes a reference to a True Name. The underlying True Pile is not removed from the True File registry **126** unless there are no additional references to the file. With reference to FIG. **21**, this mechanism is performed as follows:

If the global freeze lock is on, wait until the global freeze lock is turned off (Step S**354**). This prevents deleting a True File while a directory which might refer to it is being frozen. Next, find the True File registry entry record **140** given the True Name (Step S**356**). If the reference count field of the True File registry **126** is greater than zero, subtract one from the reference count field (Step S**358**). If it is determined (in step S**360**) that the reference count field of the True File registry entry record **140** is zero, and if there are no dependent systems listed in the True File registry entry record **140**, then perform the following steps:

(i) If the True File is a simple data item, then delete the True File, otherwise,

(ii) (the True File is a compound data item) for each True Name in the data item, recursively delete the True File corresponding to the True Name (Step S**362**).

(iii) Remove the file indicated by the True File ID and compressed file ID from the True File registry **126**, and remove the True File registry entry record **140** (Step S**364**).

13. Process Audit File Entry

This mechanism performs tasks which are required to maintain information in the local directory extensions table **124** and True File registry **126**, but which can be delayed while the processor is busy doing more time-critical tasks. Entries **142** in the audit file **132** should be processed at a background priority as long as there are entries to be processed. With reference to FIG. **22**, the steps for processing an entry are as follows:

Determine the operation in the entry **142** currently being processed (Step S**365**). If the operation indicates that a file was created or written (Step S**366**), then assimilate the file using the Assimilate Data Item primitive mechanism (Step S**368**), use the New True File primitive mechanism to do additional desired processing (such as cache update, compression, and mirroring) (Step S**369**), and record the newly computed True Name for the file in the audit file record entry (Step S**370**).

Otherwise, if the entry being processed indicates that a compound data item or directory was copied (or deleted) (Step S**376**), then for each component True Name in the compound data item or directory, add (or subtract) one to the use count of the True File registry entry record **140** corresponding to the component True Name (Step S**378**).

In all cases, for each parent directory of the given file, update the size, time of last access, and time of last modification, according to the operation in the audit record (Step S**379**).

Note th at the audit record is not removed after processing, but is retained for some reasonable period so that it may be used by the Synchronize Directory extended mechanism to allow a disconnected remote processor to update its representation of the local system.

14. Begin Grooming

This mechanism makes it possible to select a set of files for removal and determine the overall amount of space to be recovered. With reference to FIG. **23**, first verify that the global grooming lock is currently unlocked (Step S**382**). Then set the global grooming lock, set the total amount of space freed during grooming to zero a nd empty the list of files selected for deletion (Step S**384**). For each True File in the True File registry **126**, set the delete count to zero (Step S**386**).

15. Select For Removal

This grooming mechanism tentatively selects a pathname to allow its corresponding True File to be removed. With reference to FIG. **24**, first find the local directory extensions table entry record **138** corresponding to the given pathname (Step S**388**). Then find the True File registry entry record **140** corresponding to the True File name in the local directory extensions table entry record **138** (Step S**390**). Add one to the grooming delete count in the True File registry entry record **140** and add the pathname to a list of files selected for deletion (Step S**392**). If the grooming delete count of the True File registry entry record **140** is equal to the use count of the True File registry entry record **140**, and if the there are no entries in the dependency list of the True File registry entry record **140**, then add the size of the file indicated by the True File ID and or compressed file ID to the total amount of space freed during grooming (Step S**394**).

16. End Grooming

This grooming mechanism ends the grooming phase and removes all files selected for removal. With reference to FIG. **25**, for each file in the list of files selected for deletion, delete the file (Step S**396**) and then unlock the global grooming lock (Step S**398**).

Operating System Mechanisms

The next of the mechanisms provided by the present invention, operating system mechanisms, are now described.

The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

1. Open File

A mechanism to open a file is described with reference to FIGS. **26**(*a*) and **26**(*b*). This mechanism is given as input a pathname and the type of access required for the file (for example, read, write, read/write, create, etc.) and produces either the File ID of the file to be opened or an indication that no file should be opened. The local directory extensions table record **138** and region table record **142** associated with the opened file are associated with the open file for later use in other processing functions which refer to the file, such as read, write, and close.

First, determine whether or not the named file exists locally by examining the local directory extensions table **124** to determine whether there is an entry corresponding to the given pathname (Step S**400**). If it is determined that the file name does not exist locally, then, using the access type, determine whether or not the file is being created by this opening process (Step S**402**). If the file is not being created, prohibit the open (Step S**404**). If the file is being created, create a zero-length scratch file using an entry in local directory extensions table **124** and produce the scratch file ID of this scratch file as the result (Step S**406**).

If, on the other hand, it is determined in step S**400** that the file name does exist locally, then determine the region in which the file is located by searching the region table **128** to

find the record **142** with the longest region path which is a prefix of the file pathname (Step S**408**). This record identifies the region of the specified file.

Next, determine using the access type, whether the file is being opened for writing or whether it is being opened only for reading (Step S**410**). If the file is being opened for reading only, then, if the file is a scratch file (Step S**419**), return the scratch File ID of the file (Step S**424**). Otherwise get the True Name from the local directory extensions table **124** and make a local version of the True File associated with the True Name using the Make True File Local primitive mechanism, and then return the True File ID associated with the True Name (Step S**420**).

If the file is not being opened for reading only (Step S**410**), then, if it is determined by inspecting the region table entry record **142** that the file is in a read-only directory (Step S**416**), then prohibit the opening (Step S**422**).

If it is determined by inspecting the region table **128** that the file is in a cached region (Step S**423**), then send a Lock Cache message to the corresponding cache server, and wait for a return message (Step S**418**). If the return message says the file is already locked, prohibit the opening.

If the access type indicates that the file being modified is being rewritten completely (Step S**419**), so that the original data will not be required, then Delete the File using the Delete File OS mechanism (Step S**421**) and perform step S**406**. Otherwise, make a scratch copy of the file (Stop S**417**) and produce the scratch file ID of the scratch file as the result (Step S**424**).

2. Close File

This mechanism takes as input the local directory extensions table entry record **138** of an open file and the data maintained for the open file. To close a file, add an entry to the audit file indicating the time and operation (create, read or write). The audit file processing (using the Process Audit File Entry primitive mechanism) will take care of assimilating the file and thereby updating the other records.

3. Read File

To read a file, a program must provide the offset and length of the data to be read, and the location of a buffer into which to copy the data read.

The file to be read from is identified by an open file descriptor which includes a File ID as computed by the Open File operating system mechanism defined above. The File ID may identify either a scratch file or a True File (or True File segment). If the File ID identifies a True File, it may be either a simple or a compound True File. Reading a file is accomplished by the following steps:

In the case where the File ID identifies a scratch file or a simple True File, use the read capabilities of the underlying operating system.

In the case where the File ID identifies a compound file, break the read operation into one or more read operations on component segments as follows:

A. Identify the segment(s) to be read by dividing the specified file offset and length each by the fixed size of a segment (a system dependent parameter), to determine the segment number and number of segments that must be read.

B. For each segment number computed above, do the following:

i. Read the compound True File index block to determine the True Name of the segment to be read.

ii. Use the Realize True File from Location primitive mechanism to make the True File segment available locally. (If that mechanism fails, the Read File mechanism fails).

iii. Determine the File ID of the True File specified by the True Name corresponding to this segment.

iv. Use the Read File mechanism (recursively) to read from this segment into the corresponding location in the specified buffer.

4. Write File

File writing uses the file ID and data management capabilities of the underlying operating system. File access (Make File Local described above) can be deferred until the first read or write.

5. Delete File or Directory

The process of deleting a file, for a given pathname, is described here with reference to FIGS. **27**(*a*) and **27**(*b*).

First, determine the local directory extensions table entry record **138** and region table entry record **142** for the file (Step S**422**). If the file has no local directory extensions table entry record **138** or is locked or is in a read-only region, prohibit the deletion.

Identify the corresponding True File given the True Name of the file being deleted using the True File registry **126** (Step S**424**). If the file has no True Name, (Step S**426**) then delete the scratch copy of the file based on its scratch file ID in the local directory extensions table **124** (Step S**427**), and continue with step S**428**.

If the file has a True Name and the True File's use count is one (Step S**429**), then delete the True File (Step S**430**), and continue with step S**428**.

If the file has a True Name and the True File's use count is greater than one, reduce its use count by one (Step S**431**). Then proceed with step S**428**.

In Step S**428**, delete the local directory extensions table entry record, and add an entry to the audit file **132** indicating the time and the operation performed (delete).

6. Copy File or Directory

A mechanism is provided to copy a file or directory given a source and destination processor and pathname. The Copy File mechanism does not actually copy the data in the file, only the True Name of the file. This mechanism is performed as follows:

(A) Given the source path, get the True Name from the path. If this step fails, the mechanism fails.

(B) Given the True Name and the destination path, link the destination path to the True Name.

(C) If the source and destination processors have different True File registries, find (or, if necessary, create) an entry for the True Name in the True File registry table **126** of the destination processor. Enter into the source ID field of this new entry the source processor identity.

(D) Add an entry to the audit file **132** indicating the time and operation performed (copy).

This mechanism addresses capability of the system to avoid copying data from a source location to a destination location when the destination already has the data. In addition, because of the ability to freeze a directory, this mechanism also addresses capability of the system immediately to make a copy of any collection of files, thereby to support an efficient version control mechanisms for groups of files.

7. Move File or Directory

A mechanism is described which moves (or renames) a file from a source path to a destination path. The move operation, like the copy operation, requires no actual transfer of data, and is performed as follows:

(A) Copy the file from the source path to the destination path.

(B) If the source path is different from the destination path, delete the source path.

8. Get File Status

This mechanism takes a file pathname and provides information about the pathname. First the local directory extensions table entry record **138** corresponding to the pathname given is found. If no such entry exists, **10** then this mechanism fails, otherwise, gather information about the file and its corresponding True File from the local directory extensions table **124**. The information can include any information shown in the data structures, including the size, type, owner, True Name, sources, time of last access, time of last modification, state (local or not, assimilated or not, compressed or not), use count, expiration date, and reservations.

9. Get Files in Directory

This mechanism enumerates the files in a directory. It is used (implicitly) whenever it is necessary to determine whether a file exists (is present) in a directory. For instance, it is implicitly used in the Open File, Delete File, Copy File or Directory, and Move File operating system mechanisms, because the files operated on are referred to by pathnames containing directory names. The mechanism works as follows:

The local directory extensions table **124** is searched for an entry **138** with the given directory pathname. If no such entry is found, or if the entry found is not a directory, then this mechanism fails.

If there is a corresponding True File field in the local directory extensions table record, then it is assumed that the True File represents a frozen directory.

The Expand Frozen Directory primitive mechanism is used to expand the existing True File into directory entries in the local directory extensions table.

Finally, the local directory extensions table **124** is again searched, this time to find each directory subordinate to the given directory. The names found are provided as the result.

Remote Mechanisms

The remote mechanisms provided by the present invention are now described. Recall that remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation.

In a presently preferred embodiment, processors communicate with each other using a remote procedure call (RPC) style interface, running over one of any number of communication protocols such as IPX/SPX or TCP/IP. Each peer processor which provides access to its True File registry **126** or file regions, or which depends on another peer processor, provides a number of mechanisms which can be used by its peers.

The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

1. Locate True File

This mechanism allows a remote processor to determine whether the local processor contains a copy of a specific True File. The mechanism begins with True Name and a flag indicating whether to forward requests for this file to other servers. This mechanism is now described with reference to FIG. **28**.

First determine if the True File is available locally or if there is some indication of where the True File is located (for example, in the Source IDs field). Look up the requested True Name in the True File registry **126** (Step S**432**).

If a True File registry entry record **140** is not found for this True Name (Step S**434**), and the flag indicates that the request is not to be forwarded (Step S**436**), respond negatively (Step S**438**). That is, respond to the effect that the True File is not available.

One the other hand, if a True File registry entry record **140** is not found (Step S**434**), and the flag indicates that the request for this True File is to be forwarded (Step S**436**), then forward a request for this True File to some other processors in the system (Step S**442**). If the source table for the current processor identifies one or more publishing servers which should have a copy of this True File, then forward the request to each of those publishing servers (Step S**436**).

If a True File registry entry record **140** is found for the required True File (Step S**434**), and if the entry includes a True File ID or Compressed File ID (Step S**440**), respond positively (Step S**444**). If the entry includes a True File ID then this provides the identity or disk location of the actual physical representation of the file or file segment required. If the entry include a Compressed File ID, then a compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.

If the True File registry entry record **140** is found (Step S**434**) but does not include a True File ID (the File ID is absent if the actual file is not currently present at the current location) (Step S**440**), and if the True File registry entry record **140** includes one or more source processors, and if the request can be forwarded, then forward the request for this True File to one or more of the source processors (Step S**444**).

2. Reserve True File

This mechanism allows a remote processor to indicate that it depends on the local processor for access to a specific True File. It takes a True Name as input. This mechanism is described here.

(A) Find the True File registry entry record **140** associated with the given True File. If no entry exists, reply negatively.

(B) If the True File registry entry record **140** does not include a True File ID or compressed File ID, and if the True File registry entry record **140** includes no source IDs for removable storage volumes, then this processor does not have access to a copy of the given file. Reply negatively.

(C) Add the ID of the sending processor to the list of dependent processors for the True File registry entry record **140**. Reply positively, with an indication of whether the reserved True File is on line or off line.

3. Request True File

This mechanism allows a remote processor to request a copy of a True File from the local processor. It requires a True Name and responds positively by sending a True File back to the requesting processor. The mechanism operates as follows:

(A) Find the True File registry entry record **140** associated with the given True Name. If there is no such True File registry entry record **140**, reply negatively.

(B) Make the True File local using the Make True File Local primitive mechanism. If this mechanism fails, the Request True File mechanism also fails.

(C) Send the local True File in either it is uncompressed or compressed form to the requesting remote processor. Note that if the True File is a compound file, the components are not sent.

(D) If the remote file is listed in the dependent process list of the True File registry entry record **140**, remove it.

4. Retire True File

This mechanism allows a remote processor to indicate that it no longer plans to maintain a copy of a given True File. An alternate source of the True File can be specified, if, for instance, the True File is being moved from one server to another. It begins with a True Name, a requesting processor ID, and an optional alternate source. This mechanism operates as follows:

(A) Find a True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Find the requesting processor on the source list and, if it is there, remove it.

(C) If an alternate source is provided, add it to the source list for the True File registry entry record **140**.

(D) If the source list of the True File registry entry record **140** has no items in it, use the Locate Remote File primitive mechanism to search for another copy of the file. If it fails, raise a serious error.

5. Cancel Reservation

This mechanism allows a remote processor to indicate that it no longer requires access to a True File stored on the local processor. It begins with a True Name and a requesting processor ID and proceeds as follows:

(A) Find the True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Remove the identity of the requesting processor from the list of dependent processors, if it appears.

(C) If the list of dependent processors becomes zero and the use count is also zero, delete the True File.

6. Acquire True File

This mechanism allows a remote processor to insist that a local processor make a copy of a specified True File. It is used, for example, when a cache client wants to write through a new version of a file. The Acquire True File mechanism begins with a data item and an optional True Name for the data item and proceeds as follows:

(A) Confirm that the requesting processor has the right to require the local processor to acquire data items. If not, send a negative reply.

(B) Make a local copy of the data item transmitted by the remote processor.

(C) Assimilate the data item into the True File registry of the local processor.

(D) If a True Name was provided with the file, the True Name calculation can be avoided, or the mechanism can verify that the file received matches the True Name sent.

(E) Add an entry in the dependent processor list of the true file registry record indicating that the requesting processor depends an this copy of the given True File.

(F) Send a positive reply.

7. Lock Cache

This mechanism allows a remote cache client to lock a local file so that local users or other cache clients cannot change it while the remote processor is using it. The mechanism begins with a pathname and proceeds as follows:

(A) Find the local directory extensions table entry record **138** of the specified pathname. If no such entry exists, reply negatively.

(B) If an local directory extensions table entry record **138** exists and is already locked, reply negatively that the file is already locked.

(C) If an local directory extensions table entry record **138** exists and is not locked, lock the entry. Reply positively.

8. Update Cache

This mechanism allows a remote cache client to unlock a local file and update it with new contents. It begins with a pathname and a True Name. The file corresponding to the True Name must be accessible from the remote processor. This mechanism operates as follows:

Find the local directory extensions table entry record **138** corresponding to the given pathname. Reply negatively if no such entry exists or if the entry is not locked.

Link the given pathname to the given True Name using the Link Path to True Name primitive mechanism.

Unlock the local directory extensions table entry record **138** and return positively.

9. Check Expiration Date

Return current or new expiration date and possible alternative source to caller.

Background Processes and Mechanisms

The background processes and mechanisms provided by the present invention are now described. Recall that background mechanisms are intended to run occasionally and at a low priority to provide automated management capabilities with respect to the present invention.

The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links;
4. Verify Region; and
5. Groom Source List.

1. Mirror True File

This mechanism is used to ensure that files are available in alternate locations in mirror groups or archived on archival servers. The mechanism depends on application-specific migration/archival criteria (size, time since last access, number of copies required, number of existing alternative sources) which determine under what conditions a file should be moved. The Mirror True File mechanism operates as follows, using the True File specified, perform the following steps:

(A) Count the number of available locations of the True File by inspecting the source list of the True File registry entry record **140** for the True File. This step determines how many copies of the True File are available in the system.

(B) If the True File meets the specified migration criteria, select a mirror group server to which a copy of the file should be sent. Use the Acquire True File remote mechanism to copy the True File to the selected mirror group server. Add the identity of the selected system to the source list for the True File.

2. Groom Region

This mechanism is used to automatically free up space in a processor by deleting data items that may be available elsewhere. The mechanism depends on application-specific grooming criteria (for instance, a file may be removed if there is an alternate online source for it, it has not been accessed in a given number of days, and it is larger than a given size). This mechanism operates as follows:

Repeat the following steps (i) to (iii) with more aggressive grooming criteria until sufficient space is freed or until

all grooming criteria have been exercised. Use grooming information to determine how much space has been freed. Recall that, while grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

(i) Begin Grooming (using the primitive mechanism).

(ii) For each pathname in the specified region, for the True File corresponding to the pathname, if the True File is present, has at least one alternative source, and meets application specific grooming criteria for the region, select the file for removal (using the primitive mechanism).

(iii) End Grooming (using the primitive mechanism).

If the region is used as a cache, no other processors are dependent an True Files to which it refers, and all such True Files are mirrored elsewhere. In this case, True Files can be removed with impunity. For a cache region, the grooming criteria would ordinarily eliminate the least recently accessed True Files first. This is best done by sorting the True Files in the region by the most recent access time before performing step (ii) above. The application specific criteria would thus be to select for removal every True File encountered (beginning with the least recently used) until the required amount of free space is reached.

3. Check for Expired Links

This mechanism is used to determine whether dependencies on published files should be refreshed. The following steps describe the operation of this mechanism:

For each pathname in the specified region, for each True File corresponding to the pathname, perform the following step:

If the True File registry entry record **140** corresponding to the True File contains at least one source which is a publishing server, and if the expiration date on the dependency is past or close, then perform the following steps:

(A) Determine whether the True File registry entry record contains other sources which have not expired.

(B) Check the True Name expiration of the server. If the expiration date has been extended, or an alternate source is suggested, add the source to the True File registry entry record **140**.

(C) If no acceptable alternate source was found in steps (A) or (B) above, make a local copy of the True File.

(D) Remove the expired source.

4. Verify Region

This mechanism can be used to ensure that the data items in the True File registry **126** have not been damaged accidentally or maliciously. The operation of this mechanism is described by the following steps:

(A) Search the local directory extensions table **124** for each pathname in the specified region and then perform the following steps:

(i) Get the True File name corresponding to the pathname;

(ii) If the True File registry entry **140** for the True File does not have a True File ID or compressed file ID, ignore it.

(iii) Use the Verify True File mechanism (see extended mechanisms below) to confirm that the True File specified is correct.

5. Groom Source List

The source list in a True File entry should be groomed sometimes to ensure there are not too many mirror or archive copies. When a file is deleted or when a region definition or its mirror criteria are changed, it may be necessary to inspect the affected True Files to determine whether there are too many mirror copies. This can be done with the following steps:

For each affected True File,

(A) Search the local directory extensions table to find each region that refers to the True File.

(B) Create a set of "required sources", initially empty.

(C) For each region found,

(a) determine the mirroring criteria for that region,

(b) determine which sources for the True File satisfy the mirroring criteria, and

(c) add these sources to the set of required sources.

(D) For each source in the True File registry entry, if the source identifies a remote processor (as opposed to removable media), and if the source is not a publisher, and if the source is not in the set of required sources, then eliminate the source, and use the Cancel Reservation remote mechanism to eliminate the given processor from the list of dependent processors recorded at the remote processor identified by the source.

Extended Mechanisms

The extended mechanisms provided by the present invention are now described. Recall that extended mechanisms run within application programs over the operating system to provide solutions to specific problems and applications.

The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize Directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at Location;
7. Verify True File;
8. Track for Accounting Purposes; and
9. Track for Licensing Purposes.

1. Inventory Existing Directory

This mechanism determines the True Names of files in an existing on-line directory in the underlying operating system. One purpose of this mechanism is to install True Name mechanisms in an existing file system.

An effect of such an installation is to eliminate immediately all duplicate files from the file system being traversed. If several file systems are inventoried in a single True File registry, duplicates across the volumes are also eliminated.

(A) Traverse the underlying file system in the operating system. For each file encountered, excluding directories, perform the following:

(i) Assimilate the file encountered (using the Assimilate File primitive mechanism). This process computes its True Name and moves its data into the True File registry **126**.

(ii) Create a pathname consisting of the path to the volume directory and the relative path of the file on the media. Link this path to the computed True Name using the Link Path to True Name primitive mechanism.

2. Inventory Removable, Read-only Files

A system with access to removable, read-only media volumes (such as WORM disks and CD-ROMS) can create a usable inventory of the files on these disks without having to make online copies. These objects can then be used for archival purposes, directory overlays, or other needs. An operator must request that an inventory be created for such a volume.

This mechanism allows for maintaining inventories of the contents of files and data items on removable media, such as

diskettes and CD-ROMs, independent of other properties of the files such as name, location, and date of creation.

The mechanism creates an online inventory of the files on one or more removable volumes, such as a floppy disk or CD-ROM, when the data on the volume is represented as a directory. The inventory service uses a True Name to identify each file, providing a way to locate the data independent of its name, date of creation, or location.

The inventory can be used for archival of data (making it possible to avoid archiving data when that data is already on a separate volume), for grooming (making it possible to delete infrequently accessed files if they can be retrieved from removable volumes), for version control (making it possible to generate a new version of a CD-ROM without having to copy the old version), and for other purposes.

The inventory is made by creating a volume directory in the media inventory in which each file named identifies the data item on the volume being inventoried. Data items are not copied from the removable volume during the inventory process.

An operator must request that an inventory be created for a specific volume. Once created, the volume directory can be frozen or copied like any other directory. Data items from either the physical volume or the volume directory can be accessed using the Open File operating system mechanism which will cause them to be read from the physical volume using the Realize True File from Location primitive mechanism.

To create an inventory the following steps are taken:

(A) A volume directory in the media inventory is created to correspond to the volume being inventoried. Its contextual name identifies the specific volume.

(B) A source table entry **144** for the volume is created in the source table **130**. This entry **144** identifies the physical source volume and the volume directory created in step (A).

(C) The filesystem on the volume is traversed. For each file encountered, excluding directories, the following steps are taken:

(i) The True Name of the file is computed. An entry is created in the True Name registry **124**, including the True Name of the file using the primitive mechanism. The source field of the True Name registry entry **140** identifies the source table entry **144**.

(ii) A pathname is created consisting of the path to the volume directory and the relative path of the file on the media. This path is linked to the computed True Name using Link Path to True Name primitive mechanism.

(D) After all files have been inventoried, the volume directory is frozen. The volume directory serves as a table of contents for the volume. It can be copied using the Copy File or Directory primitive mechanism to create an "overlay" directory which can then be modified, making it possible to edit a virtual copy of a read-only medium.

3. Synchronize Directories

Given two versions of a directory derived from the same starting point, this mechanism creates a new, synchronized version which includes the changes from each. Where a file is changed in both versions, this mechanism provides a user exit for handling the discrepancy. By using True Names, comparisons are instantaneous, and no copies of files are necessary.

This mechanism lets a local processor synchronize a directory to account for changes made at a remote processor. Its purpose is to bring a local copy of a directory up to date after a period of no communication between the local and remote processor. Such a period might occur if the local processor were a mobile processor detached from its server, or if two distant processors were run independently and updated nightly.

An advantage of the described synchronization process is that it does not depend on synchronizing the clocks of the local and remote processors. However, it does require that the local processor track its position in the remote processor's audit file.

This mechanism does not resolve changes made simultaneously to the same file at several sites. If that occurs, an external resolution mechanism such as, for example, operator intervention, is required.

The mechanism takes as input a start time, a local directory pathname, a remote processor name, and a remote directory pathname name, and it operates by the following steps:

(A) Request a copy of the audit file **132** from the remote processor using the Request True File remote mechanism.

(B) For each entry **146** in the audit file **132** after the start time, if the entry indicates a change to a file in the remote directory, perform the following steps:

(i) Compute the pathname of the corresponding file in the local directory. Determine the True Name of the corresponding file.

(ii) If the True Name of the local file is the same as the old True Name in the audit file, or if there is no local file and the audit entry indicates a new file is being created, link the new True Name in the audit file to the local pathname using the Link Path to True Name primitive mechanism.

(iii) Otherwise, note that there is a problem with the synchronization by sending a message to the operator or to a problem resolution program, indicating the local pathname, remote pathname, remote processor, and time of change.

(C) After synchronization is complete, record the time of the final change. This time is to be used as the new start time the next time this directory is synchronized with the same remote processor.

4. Publish Region

The publish region mechanism allows a processor to offer the files in a region to any client processors for a limited period of time.

The purpose of the service is to eliminate any need for client processors to make reservations with the publishing processor. This in turn makes it possible for the publishing processor to service a much larger number of clients.

When a region is published, an expiration date is defined for all files in the region, and is propagated into the publishing system's True File registry entry record **140** for each file.

When a remote file is copied, for instance using the Copy File operating system mechanism, the expiration date is copied into the source field of the client's True File registry entry record **140**. When the source is a publishing system, no dependency need be created.

The client processor must occasionally and in background, check for expired links, to make sure it still has access to these files. This is described in the background mechanism Check for Expired Links.

5. Retire Directory

This mechanism makes it possible to eliminate safely the True Files in a directory, or at least dependencies on them, after ensuring that any client processors depending on those

files remove their dependencies. The files in the directory are not actually deleted by this process. The directory can be deleted with the Delete File operating system mechanism.

The mechanism takes the pathname of a given directory, and optionally, the identification of a preferred alternate source processor for clients to use. The mechanism performs the following steps:

(A) Traverse the directory. For each file in the directory, perform the following steps:

(i) Get the True Name of the file from its path and find the True File registry entry **140** associated with the True Name.

(ii) Determine an alternate source for the True File. If the source IDs field of the TFR entry includes the preferred alternate source, that is the alternate source. If it does not, but includes some other source, that is the alternate source. If it contains no alternate sources, there is no alternate source.

(iii) For each dependent processor in the True File registry entry **140**, ask that processor to retire the True File, specifying an alternate source if one was determined, using the remote mechanism.

6. Realize Directory at Location

This mechanism allows the user or operating system to force copies of files from some source location to the True File registry **126** at a given location. The purpose of the mechanism is to ensure that files are accessible in the event the source location becomes inaccessible. This can happen for instance if the source or given location are on mobile computers, or are on removable media, or if the network connection to the source is expected to become unavailable, or if the source is being retired.

This mechanism is provided in the following steps for each file in the given directory, with the exception of subdirectories:

(A) Get the-local directory extensions table entry record **138** given the pathname of the file. Get the True Name of the local directory extensions table entry record **138**. This service assimilates the file if it has not already been assimilated.

(B) Realize the corresponding True File at the given location. This service causes it to be copied to the given location from a remote system or removable media.

7. Verify True File

This mechanism is used to verify that the data item in a True File registry **126** is indeed the correct data item given its True Name. Its purpose is to guard against device errors, malicious changes, or other problems.

If an error is found, the system has the ability to "heal" itself by finding another source for the True File with the given name. It may also be desirable to verify that the error has not propagated to other systems, and to log the problem or indicate it to the computer operator. These details are not described here.

To verify a data item that is not in a True File registry **126**, use the Calculate True Name primitive mechanism described above.

The basic mechanism begins with a True Name, and operates in the following steps:

(A) Find the True File registry entry record **140** corresponding to the given True Name.

(B) If there is a True File ID for the True File registry entry record **140** then use it. Otherwise, indicate that no file exists to verify.

(C) Calculate the True Name of the data item given the file ID of the data item.

(D) Confirm that the calculated True Name is equal to the given True Name.

(E) If the True Names are not equal, there is an error in the True File registry **126**. Remove the True File ID from the True File registry entry record **140** and place it somewhere else. Indicate that the True File registry entry record **140** contained an error.

8. Track for Accounting Purposes

This mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another. The mechanism can be used as a basis for a value-based accounting system in which charges are based on the identity of the data stored or transmitted, rather than simply on the number of bits.

This mechanism allows the system to track possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. True names make it possible to identify each file briefly yet uniquely for this purpose.

Tracking the identities of files requires maintaining an accounting log **134** and processing it for accounting or billing purposes. The mechanism operates in the following steps:

(A) Note every time a file is created or deleted, for instance by monitoring audit entries in the Process Audit File Entry primitive mechanism. When such an event is encountered, create an entry **148** in the accounting log **134** that shows the responsible party and the identity of the file created or deleted.

(B) Every time a file is transmitted, for instance when a file is copied with a Request True File remote mechanism or an Acquire True File remote mechanism, create an entry in the accounting log **134** that shows the responsible party, the identity of the file, and the source and destination processors.

(C) Occasionally run an accounting program to process the accounting log **134**, distributing the events to the account records of each responsible party. The account records can eventually be summarized for billing purposes.

9. Track for Licensing Purposes

This mechanism ensures that licensed files are not used by unauthorized parties. The True Name provides a safe way to identify licensed material. This service allows proof of possession of specific files according to their contents without disclosing their contents.

Enforcing use of valid licenses can be active (for example, by refusing to provide access to a file without authorization) or passive (for example, by creating a report of users who do not have proper authorization).

One possible way to perform license validation is to perform occasional audits of employee systems. The service described herein relies on True Names to support such an audit, as in the following steps:

(A) For each licensed product, record in the license table **136** the True Name of key files in the product (that is, files which are required in order to use the product, and which do not occur in other products) Typically, for a software product, this would include the main executable image and perhaps other major files such as clip-art, scripts, or online help. Also record the identity of each system which is authorized to have a copy of the file.

(B) Occasionally, compare the contents of each user processor against the license table **136**. For each True Name in the license table do the following:

(i) Unless the user processor is authorized to have a copy of the file, confirm that the user processor does not have a copy of the file using the Locate True File mechanism.

(ii) If the user processor is found to have a file that it is not authorized to have, record the user processor and True Name in a license violation table.

The System in Operation

Given the mechanisms described above, the operation of a typical DP system employing these mechanisms is now described in order to demonstrate how the present invention meets its requirements and capabilities.

In operation, data items (for example, files, database records, messages, data segments, data blocks, directories, instances of object classes, and the like) in a DP system employing the present invention are identified by substantially unique identifiers (True Names), the identifiers depending on all of the data in the data items and only on the data in the data items. The primitive mechanisms Calculate True Name and Assimilate Data Item support this property. For any given data item, using the Calculate True Name primitive mechanism, a substantially unique identifier or True Name for that data item can be determined.

Further, in operation of a DP system incorporating the present invention, multiple copies of data items are avoided (unless they are required for some reason such as backups or mirror copies in a fault-tolerant system). Multiple copies of data items are avoided even when multiple names refer to the same data item. The primitive mechanisms Assimilate Data Items and New True File support this property. Using the Assimilate Data Item primitive mechanism, if a data item already exists in the system, as indicated by an entry in the True File registry **126**, this existence will be discovered by this mechanism, and the duplicate data item (the new data item) will be eliminated (or not added). Thus, for example, if a data file is being copied onto a system from a floppy disk, if, based on the True Name of the data file, it is determined that the data file already exists in the system (by the same or some other name), then the duplicate copy will not be installed. If the data item was being installed on the system by some name other than its current name, then, using the Link Path to True Name primitive mechanism, the other (or new) name can be linked to the already existing data item.

In general, the mechanisms of the present invention operate in such a way as to avoid recreating an actual data item at a location when a copy of that data item is already present at that location. In the case of a copy from a floppy disk, the data item (file) may have to be copied (into a scratch file) before it can be determined that it is a duplicate. This is because only one processor is involved. On the other hand, in a multiprocessor environment or DP system, each processor has a record of the True Names of the data items on that processor. When a data item is to be copied to another location (another processor) in the DP system, all that is necessary is to examine the True Name of the data item prior to the copying. If a data item with the same True Name already exists at the destination location (processor), then there is no need to copy the data item. Note that if a data item which already exists locally at a destination location is still copied to the destination location (for example, because the remote system did not have a True Name for the data item or because it arrives as a stream of un-named data), the Assimilate Data Item primitive mechanism will prevent multiple copies of the data item from being created.

Since the True Name of a large data item (a compound data item) is derived from and based on the True Names of components of the data item, copying of an entire data item can be avoided. Since some (or all) of the components of a large data item may already be present at a destination location, only those components which are not present there need be copied. This property derives from the manner in which True Names are determined.

When a file is copied by the Copy File or Directory operating system mechanism, only the True Name of the file is actually replicated.

When a file is opened (using the Open File operating system mechanism), it uses the Make True File Local primitive mechanism (either directly or indirectly through the Create Scratch File primitive mechanism) to create a local copy of the file. The Open File operating system mechanism uses the Make True File Local primitive mechanism, which uses the Realize True File from Location primitive mechanism, which, in turn uses the Request True File remote mechanism.

The Request True File remote mechanism copies only a single data item from one processor to another. If the data item is a compound file, its component segments are not copied, only the indirect block is copied. The segments are copied only when they are read (or otherwise needed).

The Read File operating system mechanism actually reads data. The Read File mechanism is aware of compound files and indirect blocks, and it uses the Realize True File from Location primitive mechanism to make sure that component segments are locally available, and then uses the operating system file mechanisms to read data from the local file.

Thus, when a compound file is copied from a remote system, only its True Name is copied. When it is opened, only its indirect block is copied. When the corresponding file is read, the required component segments are realized and therefore copied.

In operation data items can be accessed by reference to their identities (True Names) independent of their present location. The actual data item or True File corresponding to a given data identifier or True Name may reside anywhere in the system (that is, locally, remotely, offline, etc). If a required True File is present locally, then the data in the file can be accessed. If the data item is not present locally, there are a number of ways in which it can be obtained from wherever it is present. Using the source IDs field of the True File registry table, the location(s) of copies of the True File corresponding to a given True Name can be determined. The Realize True File from Location primitive mechanism tries to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. If, on the other hand, for some reason it is not known where there is a copy of the True File, or if the processors identified in the source IDs field do not respond with the required True File, the processor requiring the data item can make a general request for the data item using the Request True File remote mechanism from all processors in the system that it can contact.

As a result, the system provides transparent access to any data item by reference to its data identity, and independent of its present location.

In operation, data items in the system can be verified and have their integrity checked. This is from the manner in which True Names are determined. This can be used for security purposes, for instance, to check for viruses and to verify that data retrieved from another location is the desired and requested data. For example, the system might store the True Names of all executable applications on the system and then periodically redetermine the True Names of each of these applications to ensure that they match the stored True

Names. Any change in a True Name potentially signals corruption in the system and can be further investigated. The Verify Region background mechanism and the Verify True File extended mechanisms provide direct support for this mode of operation. The Verify Region mechanism is used to ensure that the data items in the True File registry have not been damaged accidentally or maliciously. The Verify True File mechanism verifies that a data item in a True File registry is indeed the correct data item given its True Name.

Once a processor has determined where (that is, at which other processor or location) a copy of a data item is in the DP system, that processor might need that other processor or location to keep a copy of that data item. For example, a processor might want to delete local copies of data items to make space available locally while knowing that it can rely on retrieving the data from somewhere else when needed. To this end the system allows a processor to Reserve (and cancel the reservation of) True Files at remote locations (using the remote mechanism). In this way the remote locations are put on notice that another location is relying on the presence of the True File at their location.

A DP system employing the present invention can be made into a fault-tolerant system by providing a certain amount of redundancy of data items at multiple locations in the system. Using the Acquire True File and Reserve True File remote mechanisms, a particular processor can implement its own form of fault-tolerance by copying data items to other processors and then reserving them there. However, the system also provides the Mirror True File background mechanism to mirror (make copies) of the True File available elsewhere in the system. Any degree of redundancy (limited by the number of processors or locations in the system) can be implemented. As a result, this invention maintains a desired degree or level of redundancy in a network of processors, to protect against failure of any particular processor by ensuring that multiple copies of data items exist at different locations.

The data structures used to implement various features and mechanisms of this invention store a variety of useful information which can be used, in conjunction with the various mechanisms, to implement storage schemes and policies in a DP system employing the invention. For example, the size, age and location of a data item (or of groups of data items) is provided. This information can be used to decide how the data items should be treated. For example, a processor may implement a policy of deleting local copies of all data items over a certain age if other copies of those data items are present elsewhere in the system. The age (or variations on the age) can be determined using the time of last access or modification in the local directory extensions table, and the presence of other copies of the data item can be determined either from the Safe Flag or the source IDs, or by checking which other processors in the system have copies of the data item and then reserving at least one of those copies.

In operation, the system can keep track of data items regardless of how those items are named by users (or regardless of whether the data items even have names). The system can also track data items that have different names (in different or the same location) as well as different data items that have the same name. Since a data item is identified by the data in the item, without regard for the context of the data, the problems of inconsistent naming in a DP system are overcome.

In operation, the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of these data items. True Names are globally unique identifiers which can be published simply by copying them. For example, a user might create a textual representation of a file on system A with True Name N (for instance as a hexadecimal string), and post it on a computer bulletin board. Another user on system B could create a directory entry F for this True Name N by using the Link Path to True Name primitive mechanism. (Alternatively, an application could be developed which hides the True Name from the users, but provides the same public transfer service.)

When a program on system B attempts to open pathname P linked to True Name N, the Locate Remote File primitive mechanism would be used, and would use the Locate True File remote mechanism to search for True Name N on one or more remote processors, such as system A. If system B has access to system A, it would be able to realize the True File (using the Realize True File from Location primitive mechanism) and use it locally. Alternatively, system B could find True Name N by accessing any publicly available True Name server, if the server could eventually forward the request to system A.

Clients of a local server can indicate that they depend on a given True File (using the Reserve True File remote mechanism) so that the True File is not deleted from the server registry as long as some client requires access to it. (The Retire True File remote mechanism is used to indicate that a client no longer needs a given True File.)

A publishing server, on the other hand, may want to provide access to many clients, and possibly anonymous ones, without incurring the overhead of tracking dependencies for each client. Therefore, a public server can provide expiration dates for True Files in its registry. This allows client systems to safely maintain references to a True File on the public server. The Check For Expired Links background mechanism allows the client of a publishing server to occasionally confirm that its dependencies on the publishing server are safe.

In a variation of this aspect of the invention, a processor that is newly connected (or reconnected after some absence) to the system can obtain a current version of all (or of needed) data in the system by requesting it from a server processor. Any such processor can send a request to update or resynchronize all of its directories (starting at a root directory), simply by using the Synchronize Directories extended mechanism on the needed directories.

Using the accounting log or some other user provided mechanism, a user can prove the existence of certain data items at certain times. By publishing (in a public place) a list of all True Names in the system on a given day (or at some given time), a user can later refer back to that list to show that a particular data item was present in the system at the time that list was published. Such a mechanism is useful in tracking, for example, laboratory notebooks or the like to prove dates of conception of inventions. Such a mechanism also permits proof of possession of a data item at a particular date and time.

The accounting log file can also track the use of specific data items and files by content for accounting purposes. For instance, an information utility company can determine the data identities of data items that are stored and transmitted through its computer systems, and use these identities to provide bills to its customers based on the identities of the data items being transmitted (as defined by the substantially unique identifier). The assignment of prices for storing and transmitting specific True Files would be made by the information utility and/or its data suppliers; this information would be joined periodically with the information in the accounting log file to produce customer statements.

Backing up data items in a DP system employing the present invention can be done based on the True Names of the data items. By tracking backups using True Names, duplication in the backups is prevented. In operation, the system maintains a backup record of data identifiers of data items already backed up, and invokes the Copy File or Directory operating system mechanism to copy only those data items whose data identifiers are not recorded in the backup record. Once a data item has been backed up, it can be restored by retrieving it from its backup location, based on the identifier of the data item. Using the backup record produced by the backup to identify the data item, the data item can be obtained using, for example, the Make True File Local primitive mechanism.

In operation, the system can be used to cache data items from a server, so that only the most recently accessed data items need be retained. To operate in this way, a cache client is configured to have a local registry (its cache) with a remote Local Directory Extensions table (from the cache server). Whenever a file is opened (or read), the Local Directory Extensions table is Used to identify the True Name, and the Make True File Local primitive mechanism inspects the local registry. When the local registry already has a copy, the file is already cached. Otherwise, the Locate True File remote mechanism is used to get a copy of the file. This mechanism consults the cache server and uses the Request True File remote mechanism to make a local copy, effectively loading the cache.

The Groom Cache background mechanism flushes the cache, removing the least-recently-used files from the cache client's True File registry. While a file is being modified on a cache client, the Lock Cache and Update Cache remote mechanisms prevent other clients from trying to modify the same file.

In operation, when the system is being used to cache data items, the problems of maintaining cache consistency are avoided.

To access a cache and to fill it from its server, a key is required to identify the data item desired. Ordinarily, the key is a name or address (in this case, it would be the pathname of a file). If the data associated with such a key is changed, the client's cache becomes inconsistent; when the cache client refers to that name, it will retrieve the wrong data. In order to maintain cache consistency it is necessary to notify every client immediately whenever a change occurs on the server.

By using an embodiment of the present invention, the cache key uniquely identifies the data it represents. When the data associated with a name changes, the key itself changes. Thus, when a cache client wishes to access the modified data associated with a given file name, it will use a new key (the True Name of the new file) rather than the key to the old file contents in its cache. The client will always request the correct data, and the old data in its cache will be eventually aged and flushed by the Groom Cache background mechanism.

Because it is not necessary to immediately notify clients when changes on the cache server occur, the present invention makes it possible for a single server to support a much larger number of clients than is otherwise possible.

In operation, the system automatically archives data items as they are created or modified. After a file is created or modified, the Close File operating system mechanism creates an audit file record, which is eventually processed by the Process Audit File Entry primitive mechanism. This mechanism uses the New True File primitive mechanism for any file which is newly created, which in turn uses the Mirror True File background mechanism if the True File is in a mirrored or archived region. This mechanism causes one or more copies of the new file to be made on remote processors.

In operation, the system can efficiently record and preserve any collection of data items. The Freeze Directory primitive mechanism creates a True File which identifies all of the files in the directory and its subordinates. Because this True File includes the True Names of its constituents, it represents the exact contents of the directory tree at the time it was frozen. The frozen directory can be copied with its components preserved.

The Acquire True File remote mechanism (used in mirroring and archiving) preserves the directory tree structure by ensuring that all of the component segments and True Files in a compound data item are actually copied to a remote system. Of course, no transfer is necessary for data items already in the registry of the remote system.

In operation, the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items.

The Freeze Directory primitive mechanism is used to create a collection of data items. The constituent files and segments referred to by the frozen directory are maintained in the registry, without any need to make copies of the constituents each time the directory is frozen.

Whenever a pathname is traversed, the Get Files in Directory operating system mechanism is used, and when it encounters a frozen directory, it uses the Expand Frozen Directory primitive mechanism.

A frozen directory can be copied from one pathname to another efficiently, merely by copying its True Name. The Copy File operating system mechanism is used to copy a frozen directory.

Thus it is possible to efficiently create copies of different versions of a directory, thereby creating a record of its history (hence a version control system).

In operation, the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM. The inventory is independent of other properties of the data items such as their name, location, and date of creation.

The Inventory Existing Directory extended mechanism provides a way to create True File Registry entries for all of the files in a directory. One use of this inventory is as a way to pre-load a True File registry with backup record information. Those files in the registry (such as previously installed software) which are on the volumes inventoried need not be backed up onto other volumes.

The Inventory Removable, Read-only Files extended mechanism not only determines the True Names for the files on the medium, but also records directory entries for each file in a frozen directory structure. By copying and modifying this directory, it is possible to create an on line patch, or small modification of an existing read-only file. For example, it is possible to create an online representation of a modified CD-ROM, such that the unmodified files are actually on the CD-ROM, and only the modified files are online.

In operation, the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. Using the Track for Accounting Purposes extended mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another.

True Names in Relational and Object-Oriented Databases

Although the preferred embodiment of this invention has been presented in the context of a file system, the invention of True Names would be equally valuable in a relational or object-oriented database. A relational or object-oriented database system using True Names would have similar benefits to those of the file system employing the invention. For instance, such a database would permit efficient elimination of duplicate records, support a cache for records, simplify the process of maintaining cache consistency, provide location-independent access to records, maintain archives and histories of records, and synchronize with distant or disconnected systems or databases.

The mechanisms described above can be easily modified to serve in such a database environment. The True Name registry would be used as a repository of database records. All references to records would be via the True Name of the record. (The Local Directory Extensions table is an example of a primary index that uses the True Name as the unique identifier of the desired records.)

In such a database, the operations of inserting, updating, and deleting records would be implemented by first assimilating records into the registry, and then updating a primary key index to map the key of the record to its contents by using the True Name as a pointer to the contents.

The mechanisms described in the preferred embodiment, or similar mechanisms, would be employed in such a system. These mechanisms could include, for example, the mechanisms for calculating true names, assimilating, locating, realizing, deleting, copying, and moving True Files, for mirroring True Files, for maintaining a cache of True Files, for grooming True Files, and other mechanisms based on the use of substantially unique identifiers.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiments, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

What is claimed is:

1. In a system in which a plurality of files are distributed across a plurality of computers, a method comprising:

obtaining a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the given function to determine the name comprises the contents of the data file; and

in response to a request for the a data file, the request including at least the name of the particular file, causing a copy of the file to be provided from a given one of the plurality of computers, wherein a copy of the requested file is only provided to licensed parties.

2. A method as in claim **1** further comprising:

determining, using at least the name, whether a copy of the data file is present on a particular one of said computers.

3. A method as in claim **1** further comprising:

determining, using at least the name, whether an unauthorized or unlicensed copy of the data file is present on a particular one of said computers.

4. A method as in claim **1**, further comprising:

maintaining accounting information relating to the data files.

5. A method as in claim **4**, wherein the maintaining of accounting information includes at least some of activities selected from:

(a) tracking which files have been stored on a computer; and

(b) tracking which files have been transmitted from a computer.

6. A method, in a system in which a plurality of files are distributed across a plurality of computers, wherein data in a file in the system may represent a digital message, a digital image, a video signal or an audio signal, the method comprising:

obtaining a name for a data file, the name having been determined using an MD5 function of the data, wherein the data used by the MD5 function comprises the contents of the data file; and

in response to a request for the data file, the request including at least the name of the data file, providing a copy of the data file from a given one of the plurality of computers, said providing being based at least in part on the obtained name, and wherein a copy of the requested file is only provided to licensed parties.

7. A method, in a system in which a plurality of files are distributed across a plurality of computers, wherein some of the computers communicate with each other using a TCP/IP communication protocol, the method comprising:

obtaining a name for a data file, the contents of said data file representing a digital image, the name having been determined using at least a given function of the data in the data file, wherein the data used by the given function to determine the name comprises the contents of the data file; and

in response to a request for the data file, the request including at least the name of the data file, providing a copy of the file from a given one of the plurality of computers, wherein a copy of the requested file is not provided to unlicensed parties or to unauthorized parties.

8. A method, in a network comprising a plurality of computers, some of the computers functioning as servers and some of the computers functioning as clients, wherein some computers in the network communicate with each other using a TCP/IP communication protocol, wherein a key is required to identify a file on the network, the method comprising:

storing some files on a first computer in the network and storing copies of some of the files from the first computer on a set of computers distinct from the first computer;

for a particular file, determining a different cache key from an ordinarily used key for the file, the different key being determined at least in part using a message function MD5 of the data, wherein the data used by the function to determine the name comprises the contents of the particular file; and

responsive to a request for the particular file, the request including the different key for the file, causing a copy of the particular file to be provided to the requester,

wherein the requested file is not provided to unlicensed parties, and

wherein the contents of the file may represent: a page in memory, a digital message, a digital image, a video signal or an audio signal.

9. A content delivery method, comprising:

distributing files across a network of servers;

for a particular file having a contextual name specifying locations in the network at which the file may be located, determining another name for the particular

file, the other name including at least a data identifier determined using a given function of the data, where said data used by the given function to determine the other name comprises the contents of the particular file;

obtaining a request for the particular file, the request including the contextual name and the other name of the particular file; and

responsive to the request, providing a copy of the particular file from one of the servers of the network of servers, said providing being based, at least in part, on the other name of the particular item, wherein the requested file is not provided to unlicensed parties.

10. A method, in a system in which a plurality of files are distributed across a plurality of computers, the method comprising:

obtaining a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the function comprises the contents of the particular file;

determining, using at least the name, whether a copy of the data file is present on at least one of said computers; and

determining whether a copy of the data file that is present on a at least one of said computers is an unauthorized copy or an unlicensed copy of the data file.

11. A method as in claim **10** further comprising:

allowing the file to be provided from one of the computers having an authorized or licensed copy of the file.

12. A method as in claim **10** wherein at least some of the plurality of computers comprise a peer-to-peer network.

13. A method, in a system in which a plurality of files are distributed across a plurality of computers which form a peer-to-peer network, the method comprising:

obtaining a True Name for a data file, the True Name being based at least in part on a given function of the data, wherein the data used by the given function comprises the contents of the particular file; and

determining, using at least the name, whether an unlicensed or unauthorized copy of the data file is present on a particular computer.

14. A method comprising:

obtaining a name for a data file, the name being based at least in part on a function of the data, wherein the data used by the function comprise at least the contents of the file; and

in response to a request for the data file, the request including at least the obtained name of the data file, causing the contents of the data file to be provided from a computer having a licensed copy of the data file.

15. A method as in claim **14** wherein the function is a message digest function or a hash function.

16. A method as in claim **14** wherein the function is selected from the functions: MD4, MD5, and SHA.

17. A method as in claim **14** wherein the given function randomly distributes its outputs.

18. A method as in claim **14** wherein the function produces a substantially unique value based on the data comprising the data file.

19. A method as in claim **14** wherein a data file may comprise a file, a portion of a file, a page in memory, a digital message, a digital image, a video signal or an audio signal.

20. A method as in claim **14** wherein certain processors in the network communicate with each other using a TCP/IP communication protocol.

21. A method as in claim **14** wherein said name for said data file, as determined using said function, will change when the data file is modified.

22. A method, in a system in which a plurality of files are distributed across a plurality of computers, the method comprising:

obtaining a name for a data file, the name being based at least in part on an MD5 function of the data which comprises the contents of the particular file; and

determining, using at least the obtained name, whether an unauthorized or unlicensed copy of the data file is present on a at least one of said computers.

23. A method comprising:

obtaining a list of file names, at least one file name for each of a plurality of files, each of said file names having been determined, at least in part, by applying a function to the contents of the corresponding file; and

using at least said list to determine whether unauthorized or unlicensed copies of some of the plurality of data files are present on a particular computer.

24. A method as in claim **23** further comprising:

in response to a request for a particular data file, allowing the contents of the data file to be provided from a computer determined to have a licensed or authorized copy of the data file.

25. A method as in claim **23** wherein the particular computer is part of a peer-to-peer network of computers.

26. A method as in claim **23** further comprising:

if the computer is found to have a file that it is not authorized or licensed to have, recording information about the computer and about the file.

27. A method as in claim **23** wherein the function is a message digest function or a hash function.

28. A method as in claim **23** wherein the function is selected from the functions: MD4, MD5, and SHA.

29. A method as in claim **23** wherein the given function randomly distributes its outputs.

30. A method as in claim **23** wherein the function produces a substantially unique value based on the data comprising the data file.

31. A method comprising:

obtaining a list of True Names, one for each of a plurality of files, wherein, for each of the files, the True Name for that file is determined using a function of the contents of the file;

for at least some computers that make up part of a peer-to-peer network of computers, comparing at least some of the contents of the computers to the list of True Names to determine whether unauthorized or unlicensed copies of some of the plurality of data files are present on those computers; and

based at least in part on said comparing, if a computer is found to have content that it is not authorized or licensed to have, recording information about the computer and about the unauthorized or unlicensed content.

32. A method as in claim **31** wherein the True Names are determined using a message digest function or a hash function.

33. A method as in claim **31** wherein the function is selected from the functions: MD4, MD5, and SHA.

34. A method as in claim **31**, further comprising:

in response to a request for the data file, allowing a copy of the file to be provided from a given one of the plurality of computers having an authorized or licensed copy of the file.

35. A method comprising:

obtaining a list of True Names, one for each of a plurality of files, wherein, for each of the files, the True Name for

that file is determined using an MD5 function of the contents of the file;

comparing the True Names of at least some of the contents of a computer to the list of True Names to determine whether unauthorized or unlicensed copies of some of the plurality of data files are present on that computer; and

based at least in part on said comparing, if a computer is found to have content that it is not authorized or licensed to have, recording information about the computer and about the unauthorized or unlicensed content.

36. In a system in which a data file is distributed across a plurality of computers, a method comprising:

obtaining a name for the data file, the name being based at least in part on a given function of the data, wherein the data used by the function which comprise the contents of the data file; and

determining, using at least the name, whether an unauthorized or unlicensed copy of the data file is present on a particular one of said computers.

37. Computer-readable media tangibly embodying a program of instructions executable by at least one computer, the program comprising code to:

obtain a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the function comprises the contents of the data file; and

in response to a request for the a data file, the request including at least the name of the particular file, cause a copy of the file to be provided from a given one of the plurality of computers, wherein the copy is only provided to licensed parties.

38. Computer-readable media tangibly embodying a program of instructions executable by at least one computer, the program comprising code to:

obtain a True Name for a data file, the True Name being based at least in part on a given function of the data, wherein the data used by the function comprises the contents of the particular file; and

determine, using at least the name, whether an unauthorized or unlicensed copy of the data file is present on a particular computer.

39. Computer-readable media tangibly embodying a program of instructions executable by at least one computer, the program comprising code to:

obtain a list of file names, one for each of a plurality of files, each of said file names having been determined, at least in part, by applying a function to the contents of the corresponding file; and

determine, using at least said list, whether unauthorized or unlicensed copies of some of the plurality of data files are present on a particular computer.

40. Computer-readable media tangibly embodying a program of instructions executable by at least one computer, the program comprising code to:

obtain a name for the data file, the name being based at least in part on a given function of the data which comprise the contents of the data file; and

determine, using at least the name, whether an unauthorized or unlicensed copy of the data file is present on a particular one of said computers.

41. Media as in claim **40** wherein the given function is a message digest function or a hash function.

42. A computer system programmed to:

obtain a list of file names for a plurality of files, each of said file names having been determined, at least in part, by applying a function to the contents of the corresponding file; and

determine, using at least said list, whether unauthorized or unlicensed copies of some of the plurality of data files are present on a particular computer.

43. A computer system as in claim **42** wherein the function is a message digest function or a hash function.

44. A computer system as in claim **42** wherein the function is an MD5 function.

45. In a system in which a plurality of files are distributed across a plurality of computers, at least some of the computers forming a peer-to-peer network, a method comprising:

obtaining a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the given function comprises the contents of the data file; and

in response to a request for the a data file, the request including at least the name of the particular file, attempting to cause a copy of the file to be provided from a given one of the plurality of computers, wherein the requested file is only provided to authorized or licensed parties.

46. A method, in a system in which a plurality of files are distributed across a plurality of computers, at least some of the computers forming a peer-to-peer network, the method comprising:

obtaining a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the function comprises the contents of the data file;

determining whether a copy of the data file is present on a at least one of said computers; and

determining, using at least the name, whether a copy of the data file that is present on a at least one of said computers is an unlicensed copy of the file.

47. A method, in a system in which a plurality of files are distributed across a plurality of computers, the method comprising:

determining whether a copy of a data file is present on a at least one of said computers;

obtaining a name for a data file, the name being based at least in part on a given function of the data which comprises the contents of the data file; and,

using at least the name, attempting to determine whether a copy of the data file that is present on the at least one of said computers is an unlicensed copy of the file.

48. A method as in claim **47** wherein the given function comprises a message digest or a hash function.

49. A method as in claim **48** wherein the given function is selected from the functions: MD4, MD5 and SHA.

50. A method as in claim **47** further comprising:

if a computer is found to have a file that it is not licensed to have, recording information about the computer.

51. A method as in claim **47**, further comprising:

maintaining accounting information relating to data files in the system.

52. A method as in any one of claims **4** and **51**, further comprising:

using the accounting information as a basis for a system in which charges are based on an identity of the data files.

53. A method as in claim **47**, further comprising, for at least one computer in the system:

(a) tracking which data files have been stored on a computer; and

(b) tracking which data files have been transmitted from a computer.

54. A method, in network in which a plurality of files are distributed across a plurality of computers of the network, the method comprising:

determining whether a copy of a data file is present on a at least one of said computers, said data file representing one or more of a digital image, a video signal or an audio signal;

obtaining a name for the data file, the name being based at least in part on an MD5 function of the data which comprises the contents of the data file; and,

when a copy of the data file is found to be present on one of the computers, determining, using at least the name, whether the copy of the data file is an authorized or licensed copy of the data file; and

if a computer is found to have a file that it is not authorized or licensed to have, recording information about the computer or the data file.

55. In network in which a plurality of files arc distributed across a plurality of computers of the network, a method comprising:

for each file of a plurality of data files,

(a) determining whether a copy of the file data file is present on a at least one of said computers;

(b) obtaining a name for the data file, the name being based at least in part on an MD5 function of the data which comprises the contents of the data file;

(c) when a copy of the data file is found to be present on one of the computers, determining, using at least the name, whether the copy of the data file-is an authorized or licensed copy of the data file; and,

(d) if a computer is found to have a file that it is not authorized or licensed to have, recording information about the computer or the data file.

56. A method, in a system in which a plurality of files are distributed across a plurality of computers, the method comprising:

obtaining a name for a data file, the name being based at least in part on a given function of the data, wherein the data used by the given function comprises the contents of the data file; and

determining, based at least in part on the obtained name, whether a copy of the data file that is present on a at least one of said computers is an unauthorized or unlicensed copy of the file.

* * * * *

US006928442C1

(12) **EX PARTE REEXAMINATION CERTIFICATE** (7609th)

United States Patent
Farber et al.

(10) **Number: US 6,928,442 C1**
(45) **Certificate Issued: Jul. 13, 2010**

(54) **ENFORCEMENT AND POLICING OF LICENSE CONTENT USING CONTENT-BASED IDENTIFIERS**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignee: **Level 3 Communications, LLC**, Broomfield, CO (US)

Reexamination Request:
No. 90/010,260, Aug. 29, 2008

Reexamination Certificate for:
Patent No.: **6,928,442**
Issued: **Aug. 9, 2005**
Appl. No.: **09/987,723**
Filed: **Nov. 15, 2001**

Related U.S. Application Data

(63) Continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/10**; 707/999.01; 707/E17.01; 709/203; 709/219; 709/229

(58) **Field of Classification Search** 705/52, 705/59; 707/2; 713/168, 180
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,835,260	A		9/1974	Prescher et al.
4,096,568	A		6/1978	Bennett et al.
4,221,003	A		9/1980	Chang et al.
4,558,413	A		12/1985	Schmidt et al.
4,658,093	A	*	4/1987	Hellman 705/52
4,821,184	A		4/1989	Clancy et al.
4,914,586	A		4/1990	Swinehart et al.
4,949,302	A		8/1990	Arnold et al.
5,007,658	A		4/1991	Blechschmidt et al.
5,014,192	A		5/1991	Mansfield et al.
5,047,918	A		9/1991	Schwartz et al.
5,084,815	A		1/1992	Mazzario
5,117,351	A		5/1992	Miller
5,163,147	A		11/1992	Orita
5,182,799	A		1/1993	Tamura et al.
5,199,073	A		3/1993	Scott
5,202,982	A	*	4/1993	Gramlich et al. 707/2
5,204,897	A		4/1993	Wyman

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0 268 069	A2	5/1988
EP	0 315 425		5/1989
EP	0 558 945	A2	9/1993

(Continued)

OTHER PUBLICATIONS

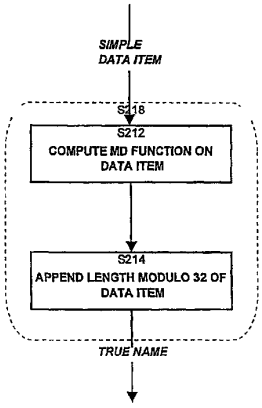
David R. Cheriton and Timothy P. Mann "Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance," published by ACM Transactions on Computer Systems, vol. 7, No. 2 (1989).*

(Continued)

Primary Examiner—Christopher E Lee

(57) **ABSTRACT**

Data files are distributed across a plurality of computers. The computers may form a network such as a content delivery network (CDN) or a peer-to-peer network. The network may operate as a TCP/IP network such as the Internet. Data files may represent may represent digital messages, images, videos or audio signals. For content—data items or files in the system—a name is obtained (or determined), where the name is based, at least in part, on a given function of the data in a data item or file. The given function may be a message digest or hash function, and it may be MD4, MD5, and SHA. A cony of a requested file is only provided to licensed (or authorized) parties. The system may check one or more computers for unauthorized or unlicensed content. Content is served based on a measure of availability of servers.



U.S. PATENT DOCUMENTS

5,204,958	A		4/1993	Cheng et al.
5,204,966	A		4/1993	Wittenberg et al.
5,247,620	A		9/1993	Fukuzawa et al.
5,260,999	A		11/1993	Wyman
5,276,869	A		1/1994	Forrest et al.
5,287,514	A		2/1994	Gram
5,297,279	A		3/1994	Bannon et al.
5,317,693	A		5/1994	Cuenod et al.
5,347,653	A		9/1994	Flynn et al.
5,351,302	A		9/1994	Leighton et al.
5,357,440	A		10/1994	Talbott et al.
5,359,523	A		10/1994	Talbott et al.
5,361,356	A		11/1994	Clark et al.
5,371,897	A		12/1994	Brown et al.
5,394,555	A		2/1995	Hunter et al.
5,403,639	A		4/1995	Belsan et al.
5,438,508	A		8/1995	Wyman
5,442,343	A		8/1995	Cato et al.
5,448,718	A		9/1995	Cohn et al.
5,454,039	A		9/1995	Coppersmith et al.
5,465,365	A		11/1995	Winterbottom
5,467,471	A		11/1995	Bader
5,475,826	A		12/1995	Fischer
5,491,817	A		2/1996	Gopal et al.
5,499,294	A		3/1996	Friedman
5,504,879	A		4/1996	Eisenberg et al.
5,537,585	A		7/1996	Blickenstaff et al.
5,553,143	A	*	9/1996	Ross et al. 705/59
5,581,615	A	*	12/1996	Stern 713/180
5,581,764	A		12/1996	Fitzgerald et al.
5,588,147	A		12/1996	Neeman et al.
5,600,834	A		2/1997	Howard
5,604,803	A		2/1997	Aziz
5,604,892	A		2/1997	Nuttall et al.
5,630,067	A		5/1997	Kindell et al.
5,632,031	A		5/1997	Velissaropoulos et al.
5,649,196	A		7/1997	Woodhill et al.
5,677,952	A		10/1997	Blakley, III et al.
5,678,038	A		10/1997	Dockter et al.
5,694,596	A		12/1997	Campbell
5,701,316	A		12/1997	Alferness et al.
5,710,922	A		1/1998	Alley et al.
5,724,425	A		3/1998	Chang et al.
5,724,552	A		3/1998	Taoda
5,742,807	A		4/1998	Masinter
5,745,879	A		4/1998	Wyman
5,757,913	A	*	5/1998	Bellare et al. 713/168
5,757,915	A		5/1998	Aucsmith et al.
5,826,049	A		10/1998	Ogata et al.
5,864,683	A		1/1999	Boebert et al.
5,907,619	A		5/1999	Davis
5,940,504	A	*	8/1999	Griswold 705/59
5,978,791	A		11/1999	Farber et al.
5,991,414	A		11/1999	Garay et al.
6,135,646	A		10/2000	Kahn et al.
6,415,280	B1		7/2002	Farber et al.
6,732,180	B1		5/2004	Hale et al.
6,928,442	B2		8/2005	Farber et al.
2002/0052884	A1		5/2002	Farber et al.
2002/0082999	A1		6/2002	Lee et al.
2003/0078888	A1		4/2003	Lee et al.
2003/0078889	A1		4/2003	Lee et al.
2003/0095660	A1		5/2003	Lee et al.
2004/0139097	A1		7/2004	Farber et al.
2005/0010792	A1		1/2005	Carpentier et al.
2005/0114296	A1		5/2005	Farber et al.
2007/0185848	A1		8/2007	Farber et al.
2008/0065635	A1		3/2008	Farber et al.
2008/0066191	A1		3/2008	Farber et al.
2008/0071855	A1		3/2008	Farber et al.
2008/0082551	A1		4/2008	Farber et al.

FOREIGN PATENT DOCUMENTS

EP	0 566 967	A2	10/1993
EP	0631 226	A1	12/1994
EP	0 654 920	A2	5/1995
EP	0 658 022	A2	6/1995
GB	2294132	A	4/1996
JP	59058564		4/1984
JP	63-106048		5/1988
JP	63-273961		11/1988
JP	2-127755		5/1990
JP	05162529		6/1993
JP	06187384	A2	7/1994
JP	06348558	A	12/1994
WO	WO 92/20021		11/1992
WO	WO 94/06087		3/1994
WO	WO 94/20913		9/1994
WO	WO 95/01599		1/1995
WO	WO 97/43717		11/1997

OTHER PUBLICATIONS

Berners–Lee, T. et al., RFC 1738, "Uniform Resource Locators (URL)," pp. 1–25, The Internet Engineering Task Force (IETF), Network Working Group, Dec. 1994.

Berners–Lee, T. et al., RFC 1945, "Hypertext Transfer Protocol—HTTP/1.0," The Internet Engineering Task Force (IETF), Network Working Group, May 1996, pp. 1–60.

Berners–Lee, T., RFC 1630 "Universal Resource Identifiers in WWW," The Internet Engineering Task Force (IETF), Network Working Group, Jun. 1994, pp. 1–28.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Tech. Report CU–CS–732–94, Dept. Comp. Sci., U. of. Colorado—Boulder, original date Aug. 1994, revised Mar. 1995, pp. 1–29.

Browne, Shirley et al., "Location–Independent Naming for Virtual Distributed Software Repositories," Univ. of Tennessee, Dept. Comp. Sci., Feb. 1995, also In ACM–SIGSOFT 1995 Symp. on Software Reusability, Seattle, Wash. Apr. 1995, 7 pages.

Falstrom, P. et al., RFC 1914, "How to Interact with a Whois++ Mesh," The Internet Engineering Task Force (IETF), Network Working Group, Feb. 1996, pp. 1–10.

Fielding, R. et al., RFC 2068, "Hypertext Transfer Protocol—HTTP/1.1," The Internet Engineering Task Force (IETF), Network Working Group, Jan. 1997, pp. 1–163.

Fielding, R. et al., RFC 2616, "Hypertext Transfer Protocol—HTTP/1.1," The Internet Engineering Task Force (IETF), Network Working Group, Jun. 1999, pp. 1–176.

Moats, R., RFC 2141, "URN Syntax," The Internet Engineering Task Force (IETF), Network Working Group, May 1997, pp. 1–8.

Vincenzetti, D. et al., "Anti Tampering Program," Proc. Fourth {USENIX} Security Symp. , Santa Clara, (USENIX Association) CA, Oct. 1993, 11 pages.

Vincenzetti, D. et al., "Anti Tampering Program," Proc. Fourth {USENIX} Security Symp., Santa Clara, CA, Oct. 1993, printed from http://www.ja.net/CERI/Vincenzetti_and_Cotrozzi/ATP_Anti_Tamp on Mar. 22, 2006, (USENIX Association), 8 pgs.

Fowler, et al. "A User–Level Replicated File System," AT&T Bell Laboratories Technical Memorandum 0112670–930414–05, Apr. 1993, and USENIX 1993 Summer Conference Proceedings, Cincinnati, OH, Jun. 1993.

Greene, D., et al., "Multi–Index Hashing for Information Retrieval", Nov. 20–22, 1994, Proceedings, 35th Annual Symp on Foundations of Computer Science, IEEE, pp. 722–731.

Hirano, et al, "Extendible hashing for concurrent insertions and retrievals," in Proc 4th Euromicro Workshop on Parallel and Distributed Processing, 1996 (PDP '96), Jan. 24, 1996 to Jan. 26, 1996, pp. 235–242, Braga, Portugal.

Preneel et al., "The Cryptographic Hash Function RIPEMD–160", appeared in CryptoBytes RSA Laboratories, vol. 3, No. 2, pp. 9–14, Fall, 1997 (also Bosselaers et al., "The RIPEMD–160 Cryptographic Hash Function", Jan. 1997, Dr. Dobb's Journal, pp. 24–28).

Prusker et al., "The Siphon: Managing Distant Replicated Repositories" Nov. 8–9, 1990, Proc. Management of Replicated Data IEEE.

Reply to Examination Report, Munich, Nov. 18, 2009, in Application No. EP 96 910 762.2 [19 pgs.].

Rich, K. et al, "Hobgoblin: A File and Directory Auditor", Sep. 30–Oct. 3, 1991, Lisa V., San Diego, CA.

Barbara, D., et al., "Exploiting symmetries for low–cost comparison of file copies," 8th Int'l Conf. on Distributed Computing Systems, Jun. 1988, pp. 471–479, San Jose, CA.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Aug. 4, 1994, pp. 1–27.

Brisco, T., "DNS Support for Load Balancing," IETF RFC 1794, Apr. 1995, pp. 1–7.

Browne, Shirley et al., "Location–Independent Naming for Virtual Distributed Software Repositories," Nov. 11, 1994, printed from http:/www.netlib.org/utk/papers/lifn/main.html on Mar. 22, 2006, 18 pages.

Campbell, M., "The Design of Text Signatures for Text Retrieval Systems," Tech. Report, Sep. 5, 1994, Deakin University, School of Computing & Math., Geelong, Australia.

Chang, W. W. et al., "A signature access method for the Starburst database system," in Proc. 15th Int'l Conf. on Very Large Data Bases (Amsterdam), The Netherlands), pp. 145–153.

Danzig, P.B., et al., "Distributed Indexing: A Scalable Mechanism For Distributed Information Retrieval," Proc. 14th Annual Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval, pp. 220–229, Oct. 13–16, 1991.

Faloutsos, C. "Access methods for text," ACM Comput. Surv. 17, 1 (Mar. 1985), 49–74.

Faloutsos, C. et al., "Description and performance analysis of signature file methods for office filing," ACM Trans. Inf. Syst. 5, 3 (Jul. 1987), 237–257.

Faloutsos, C. et al., "Signature files: an access method for documents and its analytical performance evaluation," ACM Trans. Inf. Syst. 2, 4 (Oct. 1984), 267–288.

Federal Information Processing Standards (FIPS) Publication 180–1; Secure Hash Standard, Apr. 17, 1995 [17 pgs.]

Feigenbaum, J. et al., "Cryptographic protection of databases and software," in Distributed Computing and Cryptography: Proc. DIMACS Workshop, Apr. 1991, pp. 161–172, American Mathematical Society, Boston, Mass.

Harrison, M. C., "Implementation of the substring test by hashing," Commun. ACM 14, 12 (Dec. 1971), 777–779.

Hauzeur, B. M., "A Model For Naming, Addressing, And Routing," ACM Trans. Inf. Syst. 4, Oct. 4, 1986), 293–311.

IEEE, The Authoritative Dictionary of IEEE Standards Terms, 7th ed., Copyright 2000, pp. 107, 176, 209, 240, 241, 432, 468, 505, 506, 682, 1016, 1113, 1266, and 1267.

International Search Report dated Jun. 24, 1996 in international application PCT/US1996/004733.

Ishikawa, Y., et al., "Evaluation of signature files as set access facilities in OODBs," In Proc. of the 1993 ACM SIGMOD Inter. Conf. on Management of Data (Washington, D.C., U.S., May 1993). P. Buneman & S. Jajodia, Eds. SIGMOD '93. ACM, NY, NY, 247–256.

Karp, R. M. and Rabin, M. O., "Efficient randomized pattern–matching algorithms," IBM J. Res. Dev. 31, 2 (Mar. 1987), 249–260.

Khare, R. and Lawrence, S., RFC 2817, "Upgrading to TLS Within HTTP/1.1," May 2000, pp. 1–12.

Khoshafian, S. N. et al. 1986. Object identity. In Conf. Proc. on Object–Oriented Programming Systems, Languages and Applications (Portland, Oregon, United States, Sep. 29–Oct. 2, 1986), N. Meyrowitz, Ed. OOPLSA '86. ACM Press, New York, NY, 406–416.

Kim et al. "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 23, 1995, pp. 1–18.

Kim, Gene H., and Spafford, Eugene H., "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool." COAST Labs. Dept. of Computer Sciences Purdue University, Mar. 12, 1994, pp. 1–23.

Knuth, D. E., "The Art of Computer Programming," 1973, vol. 3 "Sorting and Searching," Ch. 6.4 "Hashing," pp. 506–549.

Kumar, Vijay, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109–113.

Lantz, K. A., et al., "Towards a universal directory service." In Proc. 4th Annual ACM Symp. on Principles of Distributed Computing (Minaki, Ontario, Canada). PODC '85. ACM Press, New York, NY, 250–260.

Leach, P. J., et al. The file system of an integrated local network. In Proc. 1985 ACM 13th Annual Conf. on Comp. Sci. CSC '85. ACM Press, NY, NY, 309–324.

Leach, P.J., et al., "UIDs as Internal Names in a Distributed File System," In Proc. 1st ACM SIGACT–SIGOPS Symp. on Principles of Distributed Computing (Ottawa, Canada, Aug. 18–20, 1982). PODC '82. ACM Press, New York, NY, 34–41.

Ma, C. 1992. On building very large naming systems. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms For Distributed Systems Structuring (France, Sep. 21–23, 1992). EW 5. ACM Press, New York, NY, 1–5l.

Myers, J. and Rose, M., RFC 1864, "The Content–MD5 Header Field," Oct. 1995, pp. 1–4.

Panagopoulos, G., et al., "Bit–sliced signature files for very large text databases on a parallel machine architecture," In Proc. of the 4th Inter. Conf. on Extending Database Technology (EDBT), Cambridge, U.K., Mar. 1994, pp. 379–392 (Proc. LNCS 779 Springer 1994, ISBN 3–540–57818–8) [14 pgs.]

Patent Abstract, "Management System for Plural Versions," Pub. No. 63273961 A, published Nov. 11, 1988, NEC Corp.

Patent Abstracts of Japan, "Data Processor," Appln. No. 05135620, filed Jun. 7, 1993, Toshiba Corp.

Patent Abstracts of Japan, "Device for Generating Database and Method for the Same," Application No. 03–080504, Sun Microsyst. Inc., published Jun. 1993, 38 pages.

Patent Abstracts of Japan, "Method for Registering and Retrieving Data Base," Application No. 03–187303, Nippon Telegr. & Teleph. Corp., published Feb. 1993, 11 pages.

Peterson, L. L. 1988. A yellow–pages service for a local–area network. In Proc. ACM Workshop on Frontiers in Computer Communications Technology (Vermont, 1987). J. J. Garcia–Luna–Aceves, Ed. SIGCOMM '87. ACM Press, New York, NY, 235–242.

Ravindran, K. and Ramakrishnan, K. K. 1991. A naming system for feature–based service specification in distributed operating systems. SIGSMALL/PC Notes 17, 3–4 (Sep. 1991), 12–21.

Reed Wade (wade@cs.utk.edu), "re: Dienst and BFD/LIFN emails," Aug. 8, 1994, printed from http://www.webhistory.org/www.lists/www–talk1994q3/0416.html on Mar. 22, 2006, (7 pages).

Rivest, R., RFC 1320, "The MD4 Message–Digest Algorithm," Apr. 1992.

Rivest, R., RFC 1321, "The MD5 Message–Digest Algorithm," Apr. 1992, pp. 1–19 and errata sheet (1 page).

Rose, M., RFC 1544, "The Content–MD5 Header Field," Nov. 1993, pp. 1–3.

Ross, K., "Hash–Routing for Collections of Shared Web Caches," IEEE Network Magazine, pp. 37–44, Nov.–Dec. 1997.

Sacks–Davis, R., et al., "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst. 12, 4 (Nov. 1987), 655–696.

Schneier, B., "One–Way Hash Functions, Using Cryptographic Algorithms for Hashing," 1991, printed from http://202.179135.4/data/DDJ/articles/1991/9109/91909g/9109g.htm on Mar. 22, 2006 [9 pgs.]

Schwartz, M., et al. 1987. A name service for evolving heterogeneous systems. In Proc. 11th ACM Symp. on OS Principles (Texas, Nov. 8–11, 1987). SOSP '87. ACM Press, NY, NY, 52–62.

Shaheen–Gouda, A. and Loucks, L. 1992. Name borders. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms For Distributed Systems Structuring (Mont Saint–Michel, France, Sep. 21–23, 1992). EW 5. ACM Press, NY, NY, 1–6.

Siegel, A., et al., "Deceit: a Flexible Distributed File System," Proc. Workshop on the Management of Replicated Data, Houston, TX, pp. 15–17, Nov. 8–9, 1990.

Siegel, A., et al., "Deceit: a Flexible Distributed File System," Technical Report, TR89–1042, Cornell University, Nov. 1989.

Sun Microsystems, Inc., RFC 1094, "NFS: Network File System Protocol Specification," Mar. 1989, pp. 1–27.

Terry, D. B. 1984. An analysis of naming conventions for distributed computer systems. In Proc. ACM SIGCOMM Symp. on Communications Architectures and Protocols: Tutorials & Symp. SIGCOMM '84. ACM Press, NY, NY, 218–224.

Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, 1993, 11 pages.

* cited by examiner

EX PARTE
REEXAMINATION CERTIFICATE
ISSUED UNDER 35 U.S.C. 307

THE PATENT IS HEREBY AMENDED AS INDICATED BELOW.

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

The patentability of claims **1-12**, **14-21**, **23-35**, **37**, **39** and **42-55** is confirmed.

Claims **13**, **22**, **36**, **38**, **40**, **41** and **56** are cancelled.

* * * * *

US007802310B2

(12) **United States Patent**
Farber et al.

(10) **Patent No.: US 7,802,310 B2**
(45) **Date of Patent: *Sep. 21, 2010**

(54) **CONTROLLING ACCESS TO DATA IN A DATA PROCESSING SYSTEM**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignees: **Kinetech, Inc.**, Studio City, CA (US); **Level 3 Communications, LLC**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 79 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/980,687**

(22) Filed: **Oct. 31, 2007**

(65) **Prior Publication Data**

US 2008/0066191 A1 Mar. 13, 2008

Related U.S. Application Data

(60) Continuation of application No. 11/724,232, filed on Mar. 15, 2007, which is a continuation of application No. 11/017,650, filed on Dec. 22, 2004, which is a continuation of application No. 09/987,723, filed on Nov. 15, 2001, now Pat. No. 6,928,442, which is a continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned, application No. 11/980,687, which is a continuation of application No. 10/742,972, filed on Dec. 23, 2003, which is a division of application No. 09/987,723, filed on Nov. 15, 2001, now Pat. No. 6,928,442, which is a continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**
H04L 29/06 (2006.01)
G06F 21/00 (2006.01)

(52) **U.S. Cl.** **726/28**; 711/163

(58) **Field of Classification Search** 726/28, 726/27, 29; 713/181

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,668,647 A 6/1972 Evangelisti et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP 0 268 069 A2 5/1988

(Continued)

OTHER PUBLICATIONS

Cheriton, David R. and Mann, Timothy P., "Decentralizing a global naming service for improved performance and fault tolerance", ACM Transactions on Computer Systems, vol. 7, No. 2, May 1989, pp. 147-183.

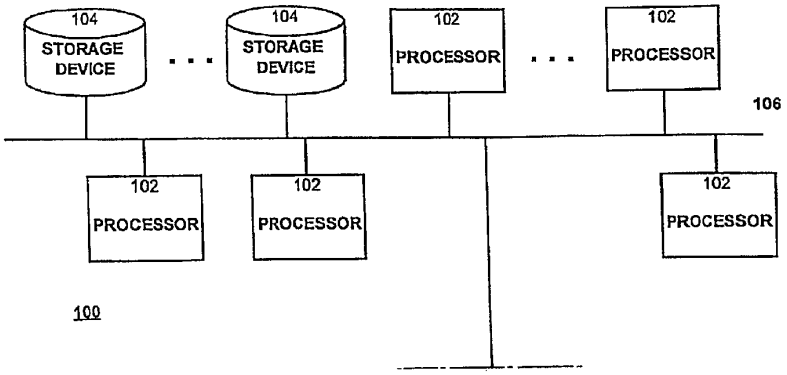
(Continued)

Primary Examiner—Gilberto Barron, Jr.
Assistant Examiner—Samson B Lemma
(74) *Attorney, Agent, or Firm*—Davidson Berquist Jackson & Gowdey, LLP; Brian Siritzky

(57) **ABSTRACT**

Access to and delivery of licensed content is controlled using content names that were determined based on the content. A name for a data item is obtained, the name having been determined based at least in part on the data which comprise the contents of the data item. Access to the data item is authorized based at least in part on the name. Once authorized, access may be granted from more than one computer. The name may have been determined using a hash or message digest function such as MD4, MD5 or SHA. The data item may comprise a file, a portion of a file, a page in memory, a digital message, a digital image, a video signal or an audio signal.

87 Claims, 31 Drawing Sheets



U.S. PATENT DOCUMENTS

3,835,260	A		9/1974	Prescher et al.
4,096,568	A		6/1978	Bennett et al.
4,215,402	A		7/1980	Mitchell et al.
4,221,003	A		9/1980	Chang et al.
4,290,105	A		9/1981	Cichelli et al.
4,376,299	A		3/1983	Rivest
4,405,829	A		9/1983	Rivest et al.
4,412,285	A		10/1983	Neches et al.
4,414,624	A		11/1983	Summer, Jr. et al.
4,441,155	A		4/1984	Fletcher et al.
4,464,713	A		8/1984	Benhase et al.
4,490,782	A		12/1984	Dixon et al.
4,558,413	A		12/1985	Schmidt et al.
4,571,700	A		2/1986	Emry, Jr. et al.
4,577,293	A		3/1986	Matick et al.
4,642,793	A		2/1987	Meaden
4,658,093	A		4/1987	Hellman
4,675,810	A		6/1987	Gruner et al.
4,691,299	A		9/1987	Rivest et al.
4,725,945	A		2/1988	Kronstadt et al.
4,773,039	A		9/1988	Zamora
4,821,184	A		4/1989	Clancy et al.
4,887,235	A		12/1989	Holloway et al.
4,888,681	A		12/1989	Barnes et al.
4,914,586	A		4/1990	Swinehart et al.
4,922,414	A		5/1990	Holloway et al.
4,922,417	A		5/1990	Churm et al.
4,949,302	A		8/1990	Arnold et al.
4,972,367	A		11/1990	Burke
5,007,658	A		4/1991	Bendert et al.
5,014,192	A		5/1991	Mansfield et al.
5,025,421	A		6/1991	Cho
5,047,918	A		9/1991	Schwartz et al.
5,050,074	A		9/1991	Marca
5,050,212	A		9/1991	Dyson
5,057,837	A		10/1991	Colwell et al.
5,077,658	A		12/1991	Bendert
5,084,815	A		1/1992	Mazzario
5,117,351	A		5/1992	Miller
5,129,081	A		7/1992	Kobayashi et al.
5,129,082	A		7/1992	Tirfing et al.
5,144,667	A		9/1992	Pogue, Jr. et al.
5,163,147	A	*	11/1992	Orita 707/9
5,179,680	A		1/1993	Colwell et al.
5,182,799	A		1/1993	Tamura et al.
5,199,073	A		3/1993	Scott
5,202,982	A	*	4/1993	Gramlich et al. 1/1
5,204,897	A		4/1993	Wyman
5,204,958	A		4/1993	Cheng et al.
5,204,966	A		4/1993	Wittenberg et al.
5,208,858	A		5/1993	Vollert et al.
5,247,620	A		9/1993	Fukuzawa et al.
5,260,999	A		11/1993	Wyman
5,276,869	A		1/1994	Forrest et al.
5,276,901	A		1/1994	Howell et al.
5,287,499	A		2/1994	Nemes
5,287,514	A		2/1994	Gram
5,297,279	A		3/1994	Bannon et al.
5,301,286	A		4/1994	Rajani
5,301,316	A		4/1994	Hamilton et al.
5,317,693	A		5/1994	Cuenod et al.
5,341,477	A		8/1994	Pitkin et al.
5,343,527	A		8/1994	Moore
5,347,653	A		9/1994	Flynn et al.
5,351,302	A		9/1994	Leighton et al.
5,357,440	A		10/1994	Talbott et al.
5,357,623	A		10/1994	Megory-Cohen
5,359,523	A		10/1994	Talbott et al.
5,361,356	A		11/1994	Clark et al.
5,371,897	A		12/1994	Brown et al.
5,384,565	A		1/1995	Cannon
5,394,555	A		2/1995	Hunter et al.
5,403,639	A		4/1995	Belsan et al.
5,404,508	A		4/1995	Konrad et al.
5,438,508	A		8/1995	Wyman
5,442,343	A		8/1995	Cato et al.
5,448,668	A	*	9/1995	Perelson et al. 714/21
5,448,718	A		9/1995	Cohn et al.
5,452,447	A		9/1995	Nelson et al.
5,454,000	A	*	9/1995	Dorfman 714/54
5,454,039	A		9/1995	Coppersmith et al.
5,459,860	A		10/1995	Burnett
5,465,365	A		11/1995	Winterbottom
5,467,471	A		11/1995	Bader
5,475,826	A		12/1995	Fischer
5,479,654	A	*	12/1995	Squibb 707/695
5,491,817	A		2/1996	Gopal et al.
5,499,294	A		3/1996	Friedman
5,504,879	A		4/1996	Eisenberg et al.
5,530,757	A	*	6/1996	Krawczyk 713/188
5,537,585	A	*	7/1996	Blickenstaff et al. 707/205
5,542,087	A		7/1996	Neimat et al.
5,553,143	A		9/1996	Ross et al.
5,568,181	A		10/1996	Greenwood et al.
5,581,615	A		12/1996	Stern
5,581,758	A		12/1996	Burnett
5,581,764	A		12/1996	Fitzgerald et al.
5,583,995	A		12/1996	Gardner et al.
5,588,147	A		12/1996	Neeman et al.
5,600,834	A		2/1997	Howard
5,604,803	A		2/1997	Aziz
5,604,892	A		2/1997	Nuttall et al.
5,630,067	A		5/1997	Kindell et al.
5,632,031	A		5/1997	Velissaropoulos et al.
5,638,443	A		6/1997	Stefik et al.
5,640,564	A		6/1997	Hamilton et al.
5,649,196	A		7/1997	Woodhill et al.
5,677,952	A		10/1997	Blakley, III et al.
5,678,038	A		10/1997	Dockter et al.
5,678,046	A		10/1997	Cahill et al.
5,694,472	A	*	12/1997	Johnson et al. 713/189
5,694,596	A		12/1997	Campbell
5,701,316	A		12/1997	Alferness et al.
5,710,922	A		1/1998	Alley et al.
5,724,425	A		3/1998	Chang et al.
5,724,552	A		3/1998	Taoda
5,742,807	A	*	4/1998	Masinter 707/1
5,745,879	A		4/1998	Wyman
5,757,913	A		5/1998	Bellare et al.
5,757,915	A		5/1998	Aucsmith et al.
5,781,629	A		7/1998	Haber et al.
5,802,291	A		9/1998	Balick et al.
5,809,494	A		9/1998	Nguyen
5,826,049	A		10/1998	Ogata et al.
5,835,087	A		11/1998	Herz et al.
5,864,683	A		1/1999	Boebert et al.
5,907,619	A		5/1999	Davis
5,907,704	A		5/1999	Gudmundson et al.
5,940,504	A		8/1999	Griswold
5,978,791	A		11/1999	Farber et al.
5,991,414	A		11/1999	Garay et al.
6,006,018	A		12/1999	Burnett et al.
6,134,603	A		10/2000	Jones et al.
6,135,646	A		10/2000	Kahn et al.
6,415,280	B1		7/2002	Farber et al.
6,732,180	B1		5/2004	Hale et al.
6,816,872	B1	*	11/2004	Squibb 1/1
6,928,442	B2		8/2005	Farber et al.
2002/0052884	A1		5/2002	Farber et al.
2002/0082999	A1		6/2002	Lee et al.
2003/0078888	A1		4/2003	Lee et al.
2003/0078889	A1		4/2003	Lee et al.
2003/0095660	A1		5/2003	Lee et al.
2004/0139097	A1		7/2004	Farber et al.

2005/0010792	A1*	1/2005	Carpentier et al.	713/193
2005/0114296	A1	5/2005	Farber et al.	
2007/0185848	A1	8/2007	Farber et al.	
2008/0065635	A1	3/2008	Farber et al.	
2008/0066191	A1	3/2008	Farber et al.	
2008/0071855	A1	3/2008	Farber et al.	
2008/0082551	A1	4/2008	Farber et al.	

FOREIGN PATENT DOCUMENTS

EP	0 315 425		5/1989
EP	0 558 945	A2	9/1993
EP	0 566 967	A2	10/1993
EP	0592045		4/1994
EP	0631 226	A1	12/1994
EP	0 654 920	A2	5/1995
EP	0 658 022	A2	6/1995
GB	2294132	A	4/1996
JP	59058564		4/1984
JP	63-106048		5/1988
JP	63-273961		11/1988
JP	2-127755		5/1990
JP	05162529		6/1993
JP	06187384	A2	7/1994
JP	06348558	A	12/1994
WO	WO 92/20021		11/1992
WO	WO 94/06087		3/1994
WO	WO 94/20913		9/1994
WO	WO 95/01599		1/1995
WO	WO 97/43717		11/1997

OTHER PUBLICATIONS

Request for Reexamination of U.S. Patent No. 6,928,442: Reexam U.S. Appl. No. 90/010,260, filed on Aug. 29, 2008.

Advances in Cryptology-AUSCRYPT '92—Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia Dec. 13-16, 1992 Proceedings.

Advances in Cryptology-EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23-27, 1993 Proceedings.

Affidavit of Timothy P. Walker in Support of CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.

Akamai and MIT's Memorandum in Support of Their Claim Construction of USPAT 5,978,791, dated Aug. 31, 2001, from Civil Action No. 00-cv-11851RWZ.

Akamai's Answer, Affirmative Defenses and Counterclaims to Amended Complaint, filed Dec. 6, 2002, in Civil Action No. 02-CV-11430RWZ.

Akamai's Brief on Claim Construction, dated Aug. 8, 2003, from Civil Action No. 02-11430 RWZ.

Albert Langer (cmf851@anu.oz.au), http://groups.google.com/groups?selm=1991Aug7.225159.786%40newshost.anu. edu.au &oe=UTF-8&output=gplain, Aug. 7, 1991.

Alexander Dupuy (dupuy@smarts.com), "MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0081.html, Apr. 17, 1994.

Alexander Dupuy (dupuy@smarts.com), "RE: MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0113.html, Apr. 26, 1994.

Answer of Defendant RIAA to First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).

Berners-Lee, T. et al., "Hypertext Transfer Protocol—HTTP/1.0," May 1996, pp. 1-54.

Berners-Lee, T. et al., "Uniform Resource Locators (URL)," pp. 1-25, Dec. 1994.

Berners-Lee, T., "Universal Resource Identifiers in WWW," Jun. 1994, pp. 1-25.

Bert dem Boer, et al., Collisions for the compression function of MD.sub.5 pp. 292-304, 1994.

Birgit Pfitzman, Sorting Out Signature Schemes, Nov. 1993, 1.sup.st Conf. Computer & Comm. Security '93, p. 74-85.

Birgit Pfitzmann, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93 pp. 74-85.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Aug. 4, 1994, pp. 1-27.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Mar. 12, 1995, pp. 1-29.

Brisco, T., "DNS Support for Load Balancing," Apr. 1995, pp. 1-7.

Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, 7 pages.

Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, printed from http:/www.netlib.org/utk/papers/lifn/main.html on Mar. 22, 2006, 18 pages.

Carter, J. Lawrence, et al. "Universal Classes of Hash Functions." Journal of Computer and System Sciences, vol. 18, No. 2, Apr. 1979, pp. 143-154.

Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, pp. 156-164, 1993.

Civil Minutes General dated Jan. 25, 2005, from Civil Action No. CV 04-7456-JFW (CTx).

Clifford Lynch (Calur@uccmvsa.bitnet), "ietf url/uri overview draft paper (long)", www.acl.lanl.gov/URI/archive/uri-93q1.messages/0015.html, Mar. 25, 1993.

Complaint for Patent Infringement, Permanent Injunction, and Damages, dated Sep. 8, 2004, from Civil Action No. CV 04-7456 JFW (AJWx).

Cormen, Thomas H., et al. Introduction to Algorithms, The MIT Press, Cambridge, Massachusetts, 1994, pp. 219-243, 991-993.

CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.

CWIS' Reply Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Aug. 15, 2003, from Civil Action No. 02-11430 RWZ.

Danzig, P.B., et al., ""Distributed Indexing: A Scalable Mechanism For Distributed Information Retrieval,"" Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 220-229, Oct. 13-16, 1991.

Davis, James R., "A Server for a Distributed Digital Technical Report Library," Jan. 15, 1994, pp. 1-8.

Declaration of Robert B.K. Dewar in Support of CWIS' Construction of the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-cv-11430RWZ.

Deering, Stephen, et al. "Multicast Routing in Datagram Internetworks and Extended LANs." ACM Transactions on Computer Systems, vol. 8, No. 2, May 1990, pp. 85-110.

Defendant Digital Island's Opening Brief on Claim Construction Issues dated Aug. 17, 2001, from Civil Action No. 00-cv-11851-RWZ.

Defendant Lime Wire, LLC's Answer, Affirmative Defenses and Counterclaims dated Nov. 15, 2007, from Civil Action No. 07-06161 VBF (PLAx).

Defendant Media Sentry, Inc.'s Reply Memorandum of Points and Authorities in Further Support of Its Motion to Dismiss, dated Nov. 15, 2004, from Civil Action No. CV04-7456 JFW (CTx).

Defendant MediaSentry Inc.'s Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW.

Defendant MediaSentry, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaims, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).

Defendant RIAA's Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW (CTx).

Defendants Loudeye Corp.'s and Overpeer, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. 04-7456 JFW (AJWx).

Defendants' Preliminary Invalidity Contentions dated Dec. 14, 2006, from Civil Action No. CV 06-5086 SJO (Ex).

Devine, Robert. "Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm." In Proc. of 4th International Conference on Foundations of Data Organizations and Algorithms, 1993, pp. 101-114.

European Search Report issued Dec. 23, 2004 in correpsonding European Application No. 96910762.2-2201.

Expert Report of Professor Ellis Horowitz, dated Mar. 6, 2006, from Civil Action No. 04-7456 JFW (CTx).

Expert Report of the Honorable Gerald J. Mossinghoff, dated Mar. 13, 2006, from Civil Action No. 04-7456 JFW (CTx).

Faltstrom, P. et al., "How to Interact with a Whois++ Mesh," Feb. 1996, pp. 1-9.

Feeley, Michael, et al. "Implementing Global Memory Management in a Workstation Cluster." In Proc. of the 15th ACM Symp. on Operating Systems Principles, 1995, pp. 201-212.

Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jan. 1997, pp. 1-163.

Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jun. 1999, pp. 1-157.

First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, dated Nov. 24, 2004, from Civil Action No. CV 04-7456 JFW (CTx).

Floyd, Sally, et al. "A reliable Multicast Framework for Light-Weight Sessions and Application Level Framing." In Proceeding of ACM SIGCOMM '95, pp. 342-356.

Fredman, Michael, et al. "Storing a Sparse Table with 0(1) Worst Case Access Time." Journal of the Association for Computing Machinery, vol. 31, No. 3, Jul. 1984, pp. 538-544.

G. L. Friedman, Digital Camera With Apparatus for Authentication of Images Produced From an Image File, NASA Case No. NPO-19108-1-CU, U.S. Appl. No. 08/159,980, Nov. 24, 1993.

Grigni, Michelangelo, et al. "Tight Bounds on Minimum Broadcasts Networks." SIAM Journal of Discrete Mathematics, vol. 4, No. 2, May 1991, pp. 207-222.

Gwertzman, James, et al. "The Case for Geographical Push-Caching." Technical Report HU TR 34-94 (excerpt), Harvard University, DAS, Cambridge, MA 02138, 1994, 2 pgs.

H. Goodman, Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Structures and File Management Report Documentation p. Ad-A275 385-94-04277.

H. Goodman, Feb. 9, 1994 Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Sructures and File Management Report Documentation P. AD-A275 385-94-04277.

Hauzeur, B. M., "A Model for Naming, Addressing, and Routing," ACM Trans. Inf. Syst. 4, Oct. 4, 1986), 293-311.

International Search Report dated Jun. 24, 1996 in corresponding international application PCT/US1996/004733.

K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names", www.w3.org/Addressing/rfc1737.txt, Dec. 1994, pp. 1-7.

Khare, R. and Lawrence, S., "Upgrading to TLS Within HTTP/1.1," May 2000, pp. 1-12.

Khoshafian, S. N. et al. 1986. Object identity. In Conf. Proc. On Object-Oriented Programming Systems, Languages and Applications (Portland, Oregon, United States, Sep. 29-Oct. 2, 1986). N. Meyrowitz, Ed. OOPLSA '86. ACM Press, New York, NY, 406-416.

Kim et al., "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1-12.

Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 23, 1995, pp. 1-18.

Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1-21.

Kim, Gene H., and Spafford, Eugene H., "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool." COAST Labs. Dept. of Computer Sciences Purdue University, Mar. 12, 1994, pp. 1-23.

Knuth, Donald E., "The Art of Computer Programming," 1973, vol. 3, Ch. 6.4, pp. 506-549.

Lantz, K. A., et al., "Towards a universal directory service." In Proc. 4th Annual ACM Symp. on Principles of Distributed Computing (Minaki, Ontario, Canada). PODC '85. ACM Press, New York, NY, 250-260.

Leach, P. J., et al.. The file system of an integrated local network. In Proc. 1985 ACM 13th Annual Conf. on Comp. Sci. CSC '85. ACM Press, NY, NY, 309-324.

Leach, P.J., et al., "UIDs as Internal Names in a Distributed File System," In Proc. 1st ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (Ottawa, Canada, Aug. 18-20, 1982). PODC '82. ACM Press, New York, NY, 34-41.

Ma, C. 1992. On building very large naming systems. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms For Distributed Systems Structuring (France, Sep. 21-23, 1992). EW 5. ACM Press, New York, NY, 1-5.

Memorandum of Points and Authorities in Support of Loudeye's and Overpeer's Motion to Dismiss the First Amended Complaint for Failure to State a Claim or, In the Alternative, for a More Definitive Statement, dated Dec. 13, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).

Ming-Ling Lo et al., On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69-78, May 1993.

Moats, R., "URN Syntax," May 1997, pp. 1-8.

Murlidhar Koushik, Dynamic Hashing With Distributed Overflow Space: A File Organization With Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299-317.

Myers, J. and Rose, M., "The Content-MD5 Header Field," Oct. 1995, pp. 1-4.

Naor, Moni, et al. "The Load, Capacity and Availability of Quorum Systems." In Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Nov. 1994, pp. 214-225.

Nisan, Noam. "Psuedorandom Generators for Space-Bounded Computation." In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, May 1990, pp. 204-212.

Office Action in corresponding Japanese Application No. 531,073/1996 mailed on Apr. 25, 2006.

Office Communication in corresponding European Application No. 96910762.2-1225 dated Jan. 17, 2007.

Order Re Claim Construction dated Nov. 8, 2001, from Civil Action No. 00-11851-RWZ.

Palmer, Mark, et al. "Fido: A Cache that Learns to Fetch." In Proceedings of the 17th International Conference on Very Large Data Bases, Sep. 1991, pp. 255-264.

Patent Abstracts of Japan, "Device for Generating Database and Method for the Same," Application No. 03-080504, Sun Microsyst. Inc., published Jun. 1993, 38 pages.

Patent Abstracts of Japan, "Electronic Mail Multiplexing System and Communication Control Method in The System." Jun. 30, 1993, JP 051625293.

Patent Abstracts of Japan, "Method for Registering and Retrieving Data Base," Application No. 03-187303, Nippon Telegr. & Teleph. Corp., published Feb. 1993, 11 pages.

Peleg, David, et al. "The Availability of Quorum Systems." Information and Computation 123, 1995, 210-223.

Peter Deutsch (peterd@bunyip.com), "Re: MD5 and LiFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0106.html, Apr. 26, 1994.

Peterson, L. L. 1988. A yellow-pages service for a local-area network. In Proc. ACM Workshop on Frontiers in Computer Communications Technology (Vermont, 1987). J. J. Garcia-Luna-Aceves, Ed. SIGCOMM '87. ACM Press, New York, NY, 235-242.

Plaintiffs' Memorandum of Points and Authorities in Opposition to Loudeye Defendants' Motion to Dismiss, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).

Plaintiffs' Opposition to Media Sentry's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV 04-7456 JFW (CTx).

Plaintiff's Opposition to Recording Industry Association of America's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (CTx).

Plaintiff's Reply to Defendant Loudeye Corp.'s and Overpeer, Inc.'s Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).
Plaintiff's Reply to Defendant MediaSentry's Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).
Plaintiff's Reply to Defendant RIAA's Counterclaims, dated Mar. 3, 2005, from Civil Action No. 04-7456 JFW (CTx).
Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.
Rabin, Michael. "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." Journal of the ACM, vol. 36, No. 2, Apr. 1989, pp. 335-348.
Ravi, R., "Rapid Rumor Ramification: Approximating the Minimum Broadcast Time." In Proc. of the 35th IEEE Symp. on Foundation of Computer Science, Nov. 1994, pp. 202-213.
Ravindran, K. and Ramakrishnan, K. K. 1991. A naming system for feature-based service specification in distributed operating systems. SIGSMALL/PC Notes 17, 3-4 (Sep. 1991), 12-21.
Reed Wade (wade@cs.utk.edu), "re: Dienst and BFD/LIFN document," Aug. 8, 1994, printed from http://www.webhistory.org/www.lists/www-talk1994q3/0416.html on Mar. 22, 2006, (7 pages).
Rivest, R., "The MD5 Message-Digest Algorithm," Apr. 1992, pp. 1-19 and errata sheet (1 page).
Rose, M., "The Content-MD5 Header Field," Nov. 1993, pp. 1-3.
Ross, K., "Hash-Routing for Collections of Shared Web Caches," IEEE Network Magazine, pp. 37-44, Nov.-Dec. 1997.
Sakti Pramanik et al., Multi-Directory Hasing, 1993, Info. Sys., vol. 18, No. 1, pp. 63-74.
Schmidt, Jeanette, et al. "Chernoff-Hoeffding Bounds for Applications with Limited Independence." In Proceedings of the 4th ACS-SIAM Symposium on Discrete Algorithms, 1993, pp. 331-340.
Schneier, Bruce, "One-Way Hash Functions, Using Crypographic Algorithms for Hashing," 1991, printed from http://202.179135.4/data/DDJ/articles/1991/9109/91909g/9109g.htm on Mar. 22, 2006.
Schwartz, M., et al. 1987. A name service for evolving heterogeneous systems. In Proc. 11th ACM Symp. on OS Principles (Texas, Nov. 8-11, 1987). SOSP '87. ACM Press, NY, NY 52-62.
Search Report dated Jun. 24, 1996.
Shaheen-Gouda, A. And Loucks, L. 1992. Name borders. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms For Distributed Systems Structuring (Mont Saint-Michel, France, Sep. 21-23, 1992). EW 5. ACM Press, NY, NY, 1-6.
Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," Mar. 1989, pp. 1-25.
Tarjan, Robert Endre, et al. "Storing a Sparse Table." Communications of the ACM, vol. 22, No. 11, Nov. 1979, pp. 606-611.
Terry, D. B. 1984. An analysis of naming conventions for distributed computer systems. In Proc. ACM SIGCOMM Symp. on Communications Architectures and Protocols: Tutorials & Symp. SIGCOMM '84. ACM Press, NY, NY, 218-224.
Thomas A. Berson, Differential Cryptanalysis Mod 2.sup.32 with Applications to MD5, pp. 69-81, 1992.
Vijay Kumar, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109-113.
Vijay Kumar, A concurrency Control Mechanism based on Extendible Hashing for Main Memory Database Systems, pp. 109-113, ACM, vol. 3, 1989.
Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, 1993, 11 pages.
Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, undated, printed from http://www.ja.net/CERI/Vincenzetti_and_Cotrozzi/ATP_Anti_Tamp on Mar. 22, 2006, 8 pages.
Vitter, Jeffrey Scott, et al. "Optimal Prefetching via Data Compression." In Proceedings of 32nd IEEE Symposium on Foundations of Computer Science, Nov. 1991, pp. 121-130.
W3C:ID, HTTP: A protocol for networked information, "Basic HTTP as defined in 1992", www.w3.org/Protocols/HTTP2.html, 1992.
Wegman, Mark, et al. "New Hash Functions and Their Use in Authentication and Set Equality." Journal of Computer and System Sciences vol. 22, Jun. 1981, pp. 265-279.
William Perrizo, et al., Distributed Join Processing Performance Evaluation, 1994. Twenty-Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236-244.
Witold Litwin et al., LH.sup.—Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21, Jun. 1993, pp. 1-22.
Witold Litwin et al., Linear Hashing for Distributed Files, ACM SIGMOD, May 1993, pp. 327-336.
Witold Litwin, et al., LH—Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21 Jun. 1993, pp. 1-22.
Yao, Andrew Chi-Chih. "Should Tables be Sorted?" Journal of the Association for Computing Machinery, vol. 28, No. 3, Jul. 1981, pp. 615-628.
Yuliang Zheng et al., Haval—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83-105.
Yuliang Zheng, et al., Haval—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83-105, Advances in Cryptology, AUSCRIPT '92, 1992.
Zhiyu Tian, et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3-13, SIGIR Forum, 1993.
Zhiyu Tian, et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3-13, SIGIR Forum, Spring 1993.
[Proposed] Order Regarding Construction of Terms, filed Mar. 29, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [9 pgs.].
Analysis of Plaintiffs' Claim Chart for the '280 Patent As Against Defendant Media Sentry, Inc. 11 pages.
Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Media Sentry, Inc. (11916.001.0150.a) pp. 1-48.
Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Overpeer pp. 1-40.
Barbara, D., et al., "Exploiting symmetries for low-cost comparison of file copies", 8th Int'l Conf. on Distributed Computing Systems, Jun. 1988, pp. 471-479, San Jose, CA.
Campbell, M., "The Design of Text Signatures for Text Retrieval Systems," Tech. Report, Sep. 5, 1994, Deakin University, School of Computing & Math., Geelong, Australia.
Chang, W. W. et al., "A signature access method for the Starburst database system," in Proc. 15th Int'l Conf. on Very Large Data Bases (Amsterdam, The Netherlands), pp. 145-153.
Changes to Mar. 23, 2007 Deposition of Robert B. K. Dewar, in C.D Cal. case No. CV 06-5086 SJO (Ex) [3 pgs + cover letter.].
Communication from EPO in European Application No. 96 910 762.2-1225 dated May 8, 2009 [4 pgs.].
Communication pursuant to Article 96(2) EPC from EPO (Examination Report), Jan. 17, 2007, in Application No. EP 96 910 762.2-1225 [1 pg. with 5 pg. annex].
Complaint for Patent Infringement, Permanent Injunction and Damages, Aug. 8, 2006, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].
Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 21, 2007 in C.D. Cal. Case No. Cv 07-06161 VBF (PLAx) [10 pgs.].
Declaration of Charles S. Baker in Support of Defendant Lime Wire's Motion to Stay Pending Reexamination of Patent and Request for Extension of Deadlines, Aug. 29, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Defendant Lime Wire, LLC's First Amended Answer, Affirmative Defenses and Counterclaims, Oct. 2, 2008, C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].
Defendant Lime Wire, LLC's Second Amended Answer, Affirmative Defenses and Counterclaims, Oct. 27, 2008, From C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].
Defendant Michael Weiss's Answer to Plaintiff's Complaint for Patent Infringement, Permanent Injunction and Damages; Demand for Jury Trial, Sep. 15, 2006, case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendant Recording Industry Association of America's Amended Notice of Motion and Motion for Partial Summary Judgment on Plaintiffs' Claims for Patent Infringement and Inducing Patent Infringement, Memorandum of Points and Authorities, May 22, 2006, redacted, original confidential, filed under seal, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [19 pgs].

Defendant Recording Industry Association of America's and Mediasentry, Inc.'s Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, In the Alternative, Based on Patent Misuse and Unclean Hands, May 22, 2006, Redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [21 pgs.].
Defendant Recording Industry Association of America's and Mediasentry, Inc's Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, In the Alternative, Based on Patent Misuse and Unclean Hands, May 8, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [20 pgs.].
Defendant StreamCast Networks Inc.'s Answer to Plaintiff's Complaint for Patent Infringement, Permanent Injunction and Damages; Demand for Jury Trial, Sep. 5, 2006, C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendants' Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 7, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendant's Second Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 9, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Dewar, Rebuttal Expert Report of Robert B.K. Dewar, in C.D. Cal. case No. CV 04-7456 JFW (CTx), Apr. 10, 2006 [87 pgs].
Faloutsos, C. "Access methods for text," ACM Comput. Surv. 17, 1 (Mar. 1985), 49-74.
Faloutsos, C. et al., "Description and performance analysis of signature file methods for office filing," ACM Trans. Inf. Syst. 5, 3 (Jul. 1987), 237-257.
Faloutsos, C. et al., "Signature files: an access method for documents and its analytical performance evaluation," ACM Trans. Inf. Syst. 2, 4 (Oct. 1984), 267-288.
Federal Information Processing Standards (FIPS) Publication 180-1; Secure Hash Standard, Apr. 17, 1995 [17 pgs.].
Feigenbaum, J. et al., "Cryptographic protection of databases and software," in Distributed Computing and Cryptography: Proc. DIMACS Workshop, Apr. 1991, pp. 161-172, American Mathematical Society, Boston, Mass.
First Amended Answer of Defendant Mediasentry to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [29 pgs.].
First Amended Answer of Defendant RIAA to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [27 pgs.].
First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [10 pgs.].
Harrison, M. C., "Implementation of the substring test by hashing," Commun. ACM 14, 12 (Dec. 1971), 777-779.
IEEE, The Authoritative Dictionary of IEEE Standards Terms, 7th ed., Copyright 2000, pp. 107, 176, 209, 240, 241, 432, 468, 505, 506, 682, 1016, 1113, 1266, and 1267.
Ishikawa, Y., et al., "Evaluation of signature files as set access facilities in OODBs," In Proc. of the 1993 ACM SIGMOD Inter. Conf. on Management of Data (Washington, D.C., U.S., May 1993). P. Buneman & S. Jajodia, Eds. SIGMOD '93. ACM, NY, NY, 247-256.
Joint Claim Construction and Prehearing Statement, N. D. Cal. Rule 4-3, Feb. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [20 pgs.].
Karp, R. M. and Rabin, M. O., "Efficient randomized pattern-matching algorithms," IBM J. Res. Dev. 31, 2 (Mar. 1987), 249-260.
List of Asserted Claims and Infringement Chart for Each Asserted Claim, Jul. 28, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [31 pgs.].
McGregor D. R. And Mariani, J. A. "Fingerprinting—A technique for file identification and maintenance," Software: Practice and Experience, vol. 12, No. 12, Dec. 1982, pp. 1165-1166.
Notice of Interested Parties, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Notice of Motion and Motion of Defendant Lime Wire to Stay Litigation Pending Reexamination of Patent and Request for Extension of Deadlines, Sep. 22, 2008, C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [11 pgs.].
Notice of Related Cases, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Office Action from PTO in U.S. Appl. No. 11/980,679, May 6, 2009.
Panagopoulos, G., et al., "Bit-sliced signature files for very large text databases on a parallel machine architecture," In Proc. of the 4th Inter. Conf. on Extending Database Technology (EDBT), Cambridge, U.K., Mar. 1994, pp. 379-392 (Proc. LNCS 779 Springer 1994, ISBN 3-540-57818-8) [14 pgs.].
Patent Abstract, " Management System for Plural Versions," Pub. No. 63273961 A, published Nov. 11, 1988, NEC Corp.
Patent Abstracts of Japan, "Data Processor," Appin. No. 05135620, filed Jun. 7, 1993, Toshiba Corp.
Plaintiff Kinetech, Inc.'s Responses to Defendant Mediasentry's First set of Interrogatories, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [14 pgs.].
Plaintiff-Counterclaim Defendant Altnet, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Brilliant Digital Entertainment, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Kinetech, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiffs Altnet, Inc., Brilliant Digital, Inc , and Kinetech, Inc.'s Responses to Defendant Recording Industry Association of America's First Set of Requests for Admissions, Jan. 6, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [26 pgs.].
Plaintiffs' Claim Construction Opening Brief and Exhibits A-D, F, G; May 7, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [112 pgs.].
Plaintiffs' Preliminary Claim Constructions and Extrinsic Evidence, Feb. 6, 2006, in case CV 06-5086 SJO (Ex) [20 pgs.].
Plaintiff's Reply to Defendant Mediasentry's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiff's Reply to Defendant RIAA's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiffs' Reply to Defendants' Claim Construction Brief, filed Apr. 23, 2007 in C.D Cal. case No. CV 06-5086 ODW (Ex) [15 pgs.].
Reply to Examination Report, Jul. 19, 2007, in Application No. EP 96 910 762.2-1225 [7 pgs.].
Response to Non-Final Office Action filed May 19, 2009 in U.S. Appl. No. 11/017,650 [19 pgs.].
Rivest, R., RFC 1320, "The MD4 Message-Digest Algorithm," Apr. 1992.
Sacks-Davis, R., et al., "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst. 12, 4 (Nov. 1987), 655-696.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Proc. Workshop on the Management of Replicated Data, Houston, TX, pp. 15-17, Nov. 8-9, 1990.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Technical Report, TR89-1042, Cornell University, Nov. 1989.
Stipulation and Proposed order to (1) Amend the Complaint, (2) Amend pretrial Schedule, and (3) Withdraw Motion to Stay, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [6 pgs.].
Streamcast Networks Inc.'s Supplemental Responses to Certain of Plaintiffs' First Set of Interrogatories, Apr. 16, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
StreamCast's Brief Re Claim Construction, Apr. 12, 2007, in C.D. Cal, case No. CV 06-5086 SJO (Ex) [11 pgs.].
Transcript of Deposition of David Farber, Feb. 16, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [94 pgs.].
Transcript of Deposition of Robert B. K. Dewar, Mar. 23, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
Transcript of Deposition of Ronald Lachman, Feb 1, 2006, C.D. Cal. case No. CV 04-7456 JFW (CTx) [96 pgs.].
USPTO, Final Office Action mailed Aug. 18, 2009 in U.S. Appl. No. 11/017,650.
USPTO, Final Office Action mailed Sep. 30, 2009 in U.S. Appl. No. 11/724,232.

USPTO, Non-Final Office action mailed Jun. 18, 2009 in U.S. Appl. No. 90/010,260.
Fowler, et al. "A User-Level Replicated File System," AT&T Bell Laboratories Technical Memorandum 0112670-930414-05, Apr. 1993, and USENIX 1993 Summer Conference Proceedings, Cincinnati, OH, Jun. 1993.
Greene, D., et al., "Multi-Index Hashing for Information Retrieval", Nov. 20-22, 1994, Proceedings, 35th Annual Symp on Foundations of Computer Science, IEEE, pp. 722-731.
Hirano, et al, "Extendible hashing for concurrent insertions and retrievals," in Proc 4th Euromicro Workshop on Parallel and Distributed Processing, 1996 (PDP '96), Jan. 24, 1996 to Jan. 26, 1996, pp. 235-242, Braga , Portugal.
Preneel et al., "The Cryptographic Hash Function RIPEMD-160", appeared in CryptoBytes RSA Laboratories, vol. 3, No. 2, pp. 9-14, Fall, 1997 (also Bosselaers et al., "The RIPEMD-160 Cryptographic Hash Function", Jan. 1997, Dr. Dobb's Journal, pp. 24-28).
Prusker et al., "The Siphon: Managing Distant Replicated Repositories" Nov. 8-9, 1990, Proc. Management of Replicated Data IEEE.
Reply to Examination Report, Munich, Nov. 18, 2009, in Application No. EP 96 910 762.2 [19 pgs.].
Rich, K. et al, "Hobgoblin: A File and Directory Auditor", Sep. 30-Oct. 3, 1991, Lisa V., San Diego, CA.
U.S. Reexam U.S. Appl. No. 90/010,260—Apr. 8, 2010 PTO Notice of Intent to Issue Ex Parte Reexamination Certificate.
USPTO Final Office Action in U.S. Appl. No. 10/742,972, Dec. 22, 2009.
USPTO, Advisory Action, Mar. 23, 2010, in U.S. Appl. No. 11/980,679.
USPTO, Final Office Action in U.S. Reexam U.S. Appl. No. 90/010,260, Jan. 29, 2010.
USPTO, Final Office Action mailed Jan. 12, 2010 in U.S. Appl. No. 11/980,679.
WIPO, International Preliminary Examination Report (IPER), Jul. 1997, PCT/US96/04733 [5 pgs.].
USPTO, Non-Final Office Action in U.S. Appl. No. 11/980,677, Jun. 4, 2010.
USPTO, Non-Final Office Action mailed Jul. 2, 2010 in U.S. Appl. No. 11/980,688.
USPTO, U.S. Reexam Control No. 90/010,260, Notice of Intent to Issue Ex Parte Reexamination Certificate, Apr. 8, 2010.
WIPO, International Preliminary Examination Report (IPER), Jul. 1997, PCT/US96/04733 [5 pgs.].
Karp, R. M. and Rabin, M. O., "Efficient randomized pattern-matching algorithms," IBM J. Res. Dev. 31, 2 (Mar. 1987), 249-260.
List of Asserted Claims and Infringement Chart for Each Asserted Claim, Jul. 28, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [31 pgs.].
McGregor D. R. and Mariani, J. A. "Fingerprinting—A technique for file identification and maintenance," Software: Practice and Experience, vol. 12, No. 12, Dec. 1982, pp. 1165-1166.
Notice of Interested Parties, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Notice of Motion and Motion of Defendant Lime Wire to Stay Litigation Pending Reexamination of Patent and Request for Extension of Deadlines, Sep. 22, 2008, C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [11 pgs.].
Notice of Related Cases, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Panagopoulos, G., et al., "Bit-sliced signature files for very large text databases on a parallel machine architecture," In Proc. of the 4th Inter. Conf. on Extending Database Technology (EDBT), Cambridge, U.K., Mar. 1994, pp. 379-392 (Proc. LNCS 779 Springer 1994, ISBN 3-540-57818-8) [14 pgs.].
Patent Abstract, "Management System for Plural Versions," Pub. No. 63273961 A, published Nov. 11, 1988, NEC Corp.
Patent Abstracts of Japan, "Data Processor," Appln. No. 05135620, filed Jun. 7, 1993, Toshiba Corp.
Plaintiff Kinetech, Inc.'s Responses to Defendant Mediasentry's First set of Interrogatories, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [14 pgs.].
Plaintiff-Counterclaim Defendant Altnet, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW(CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Brilliant Digital Entertainment, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Kinetech, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiffs Altnet, Inc., Brilliant Digital, Inc., and Kinetech, Inc.'s Responses to Defendant Recording Industry Association of America's First Set of Requests for Admissions, Jan. 6, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [26 pgs.].
Plaintiffs' Claim Construction Opening Brief and Exhibits A-D, F, G; May 7, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [112 pgs.].
Plaintiffs' Preliminary Claim Constructions and Extrinsic Evidence, Feb. 6, 2006, in case CV 06-5086 SJO (Ex) [20 pgs.].
Plaintiffs Reply to Defendant Mediasentry's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiffs Reply to Defendant RIAA's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiffs' Reply to Defendants' Claim Construction Brief, filed Apr. 23, 2007 in C.D. Cal. case No. CV 06-5086 ODW (Ex) [15 pgs.].
Reply to Examination Report, Jul. 19, 2007, in Application No. EP 96 910 762.2-1225 [7 pgs.].
Response to Non-Final Office Action filed May 19, 2009 in U.S. Appl. No. 11/017,650 [19 pgs.].
Rivest, R., RFC 1320, "The MD4 Message-Digest Algorithm," the Internet Engineering Task Force (IETF), Apr. 1992.
Sacks-Davis, R., et al., "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst. 12, 4 (Nov. 1987), 655-696.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Proc. Workshop on the Management of Replicated Data, Houston, TX, pp. 15-17, Nov. 8-9, 1990.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Technical Report, TR89-1042, Cornell University, Nov. 1989.
Stipulation and Proposed order to (1) Amend the Complaint, (2) Amend pretrial Schedule, and (3) Withdraw Motion to Stay, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF(PLAx) [6 pgs.].
Streamcast Networks Inc.'s Supplemental Responses to Certain of Plaintiffs' First Set of Interrogatories, Apr. 16, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
StreamCast's Brief Re Claim Construction, Apr. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].
Transcript of Deposition of David Farber, Feb. 16, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [94 pgs.].
Transcript of Deposition of Robert B. K. Dewar, Mar. 23, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
Transcript of Deposition of Ronald Lachman, Feb. 1, 2006, C.D. Cal. case No. CV 04-7456 JFW (CTx) [96 pgs.].
USPTO, Non-Final Office Action mailed May 6, 2009 in U.S. Appl. No. 11/980,679.
USPTO, Non-Final Office action mailed Jun. 15, 2009 in U.S. Appl. No. 11/980,687.
USPTO, Non-Final Office action mailed Jun. 18, 2009 in Reexam No. 90/010,260.
Cheriton, David R. and Mann, Timothy P., "Decentralizing a global naming service for improved performance and fault tolerance", ACM Transactions on Computer Systems, vol. 7, No. 2, May 1989, pp. 147-183.
Request for Reexamination of U.S. Patent No. 6,928,442: Reexam Control No. 90/010,260, filed on Aug. 29, 2008.

* cited by examiner

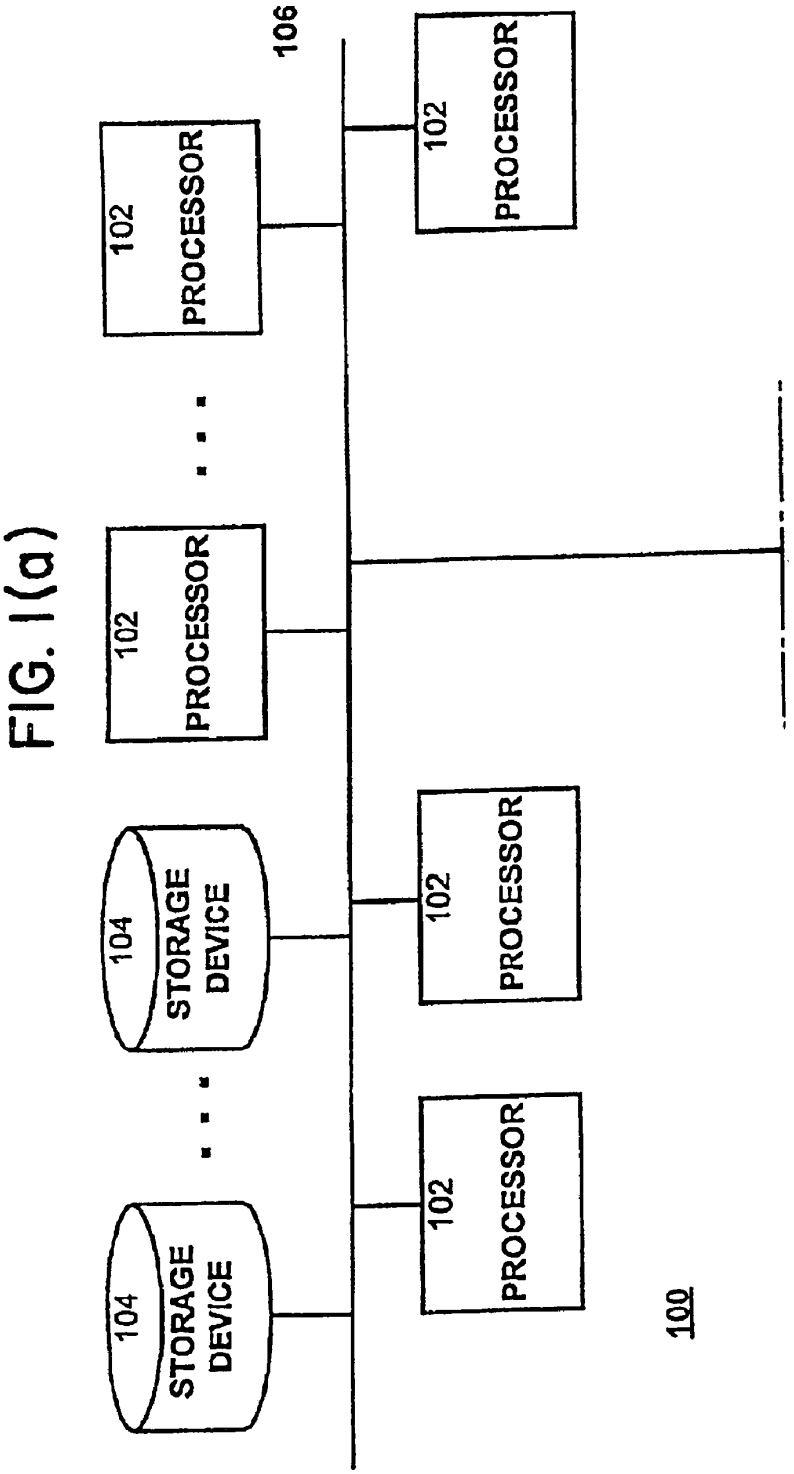

106
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
104
STORAGE DEVICE
104
STORAGE DEVICE
100

FIG. 1(a)

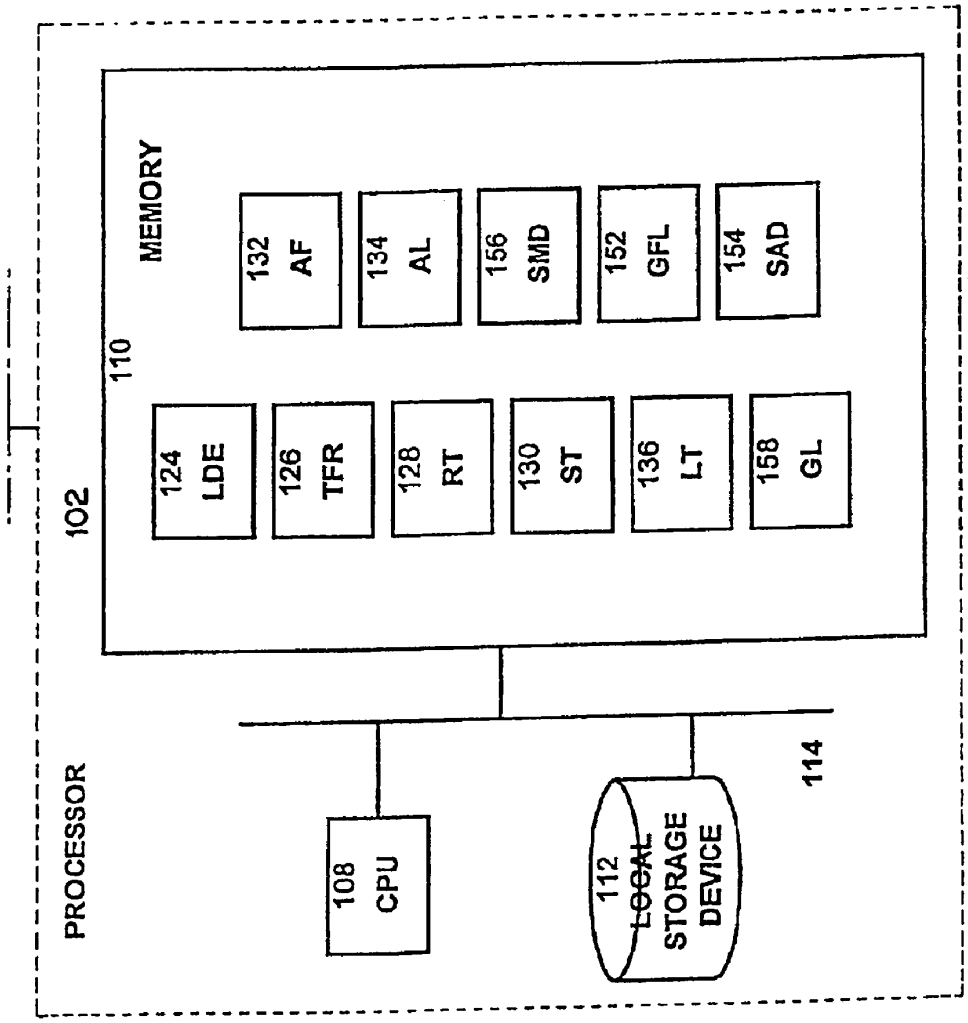

PROCESSOR
102
MEMORY
110
124 LDE
126 TFR
128 RT
130 ST
136 LT
158 GL
132 AF
134 AL
156 SMD
152 GFL
154 SAD
108 CPU
112 LOCAL STORAGE DEVICE
114

FIG. 1(b)

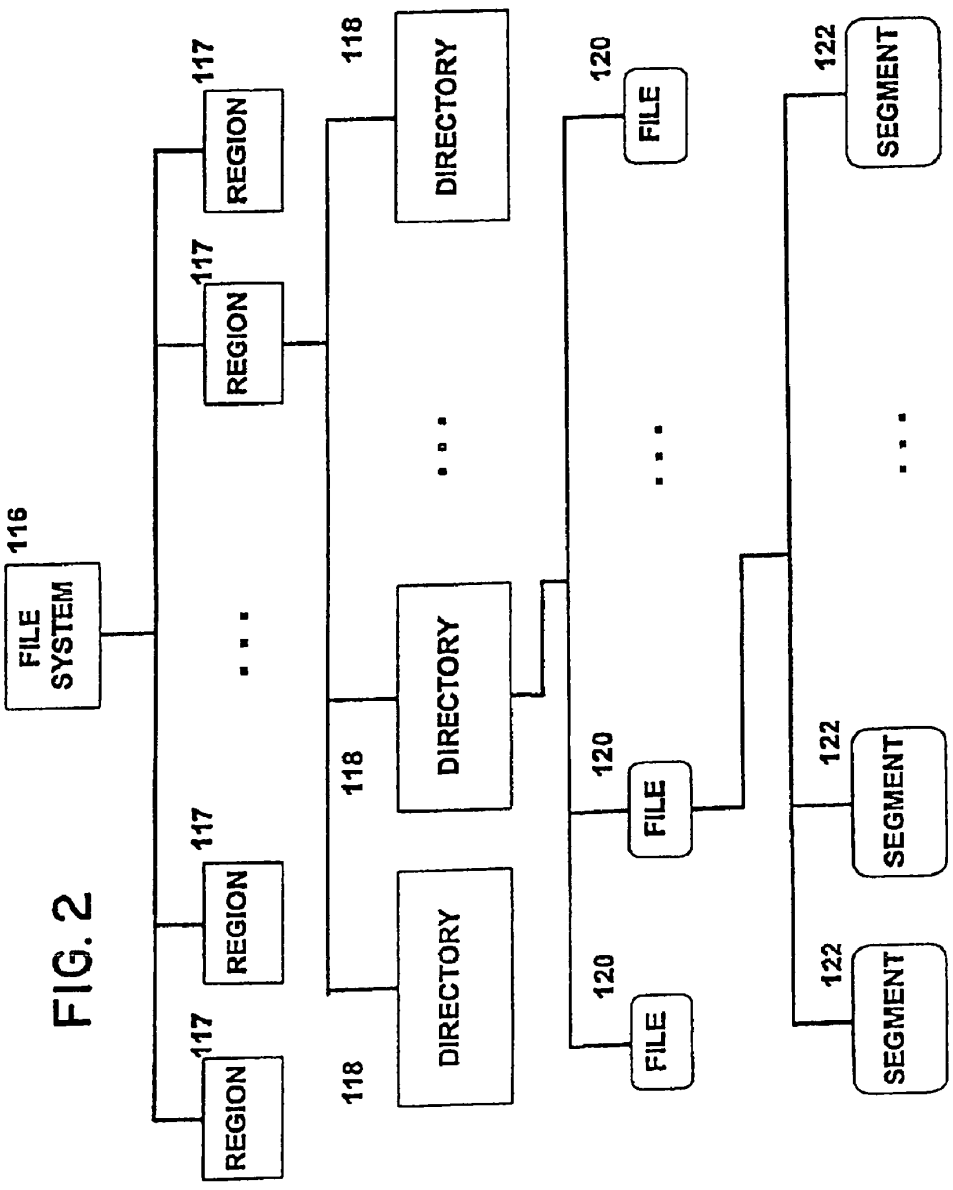

116
FILE SYSTEM
117
REGION
117
REGION
117
REGION
118
DIRECTORY
118
DIRECTORY
118
DIRECTORY
120
FILE
120
FILE
120
FILE
122
SEGMENT
122
SEGMENT
122
SEGMENT

FIG. 2

FIG. 3

138

Region ID
Pathname
True Name
Type
File ID
Time of last access
Time of last modification
Safe flag
Lock flag
Size
Owner

FIG. 4

140

True Name
File ID
Compressed File ID
Source IDs
Dependent processors
Use count
Time of last access
Expiration
Grooming delete count

142

Region ID
Region file system
Region pathname
Region status
Mirror processor(s)
Mirror duplication count
Policy

FIG. 5

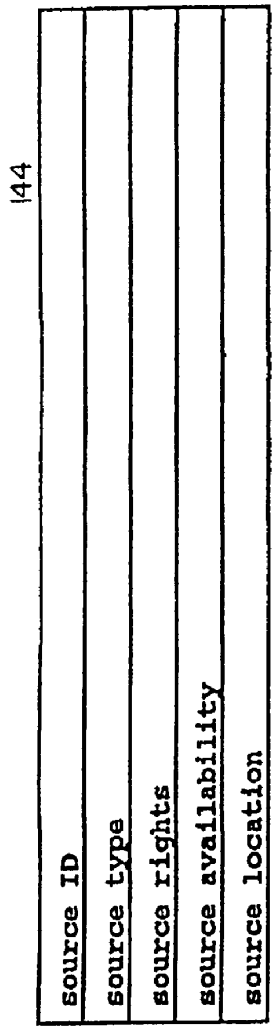


FIG. 6

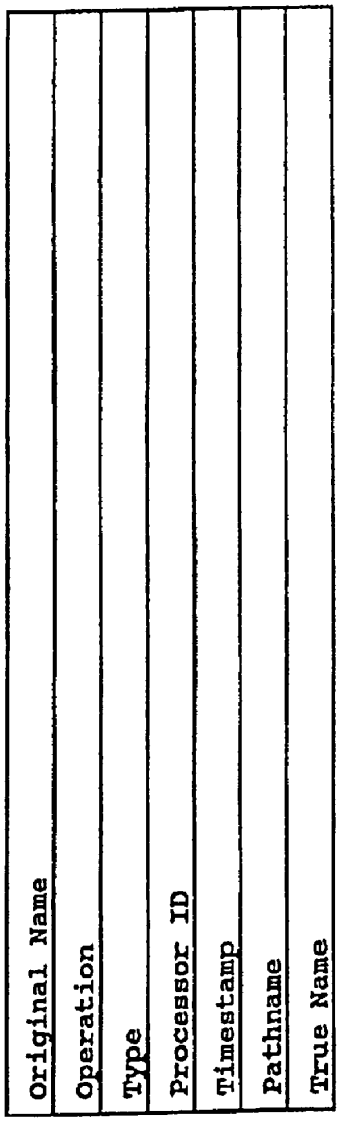


FIG. 7

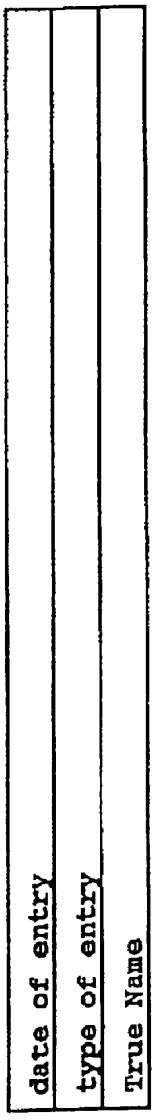


FIG. 8

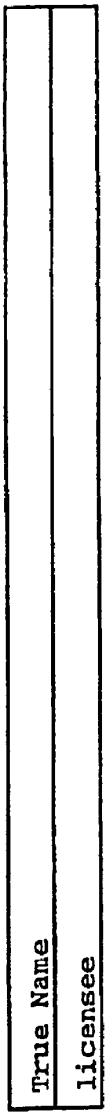
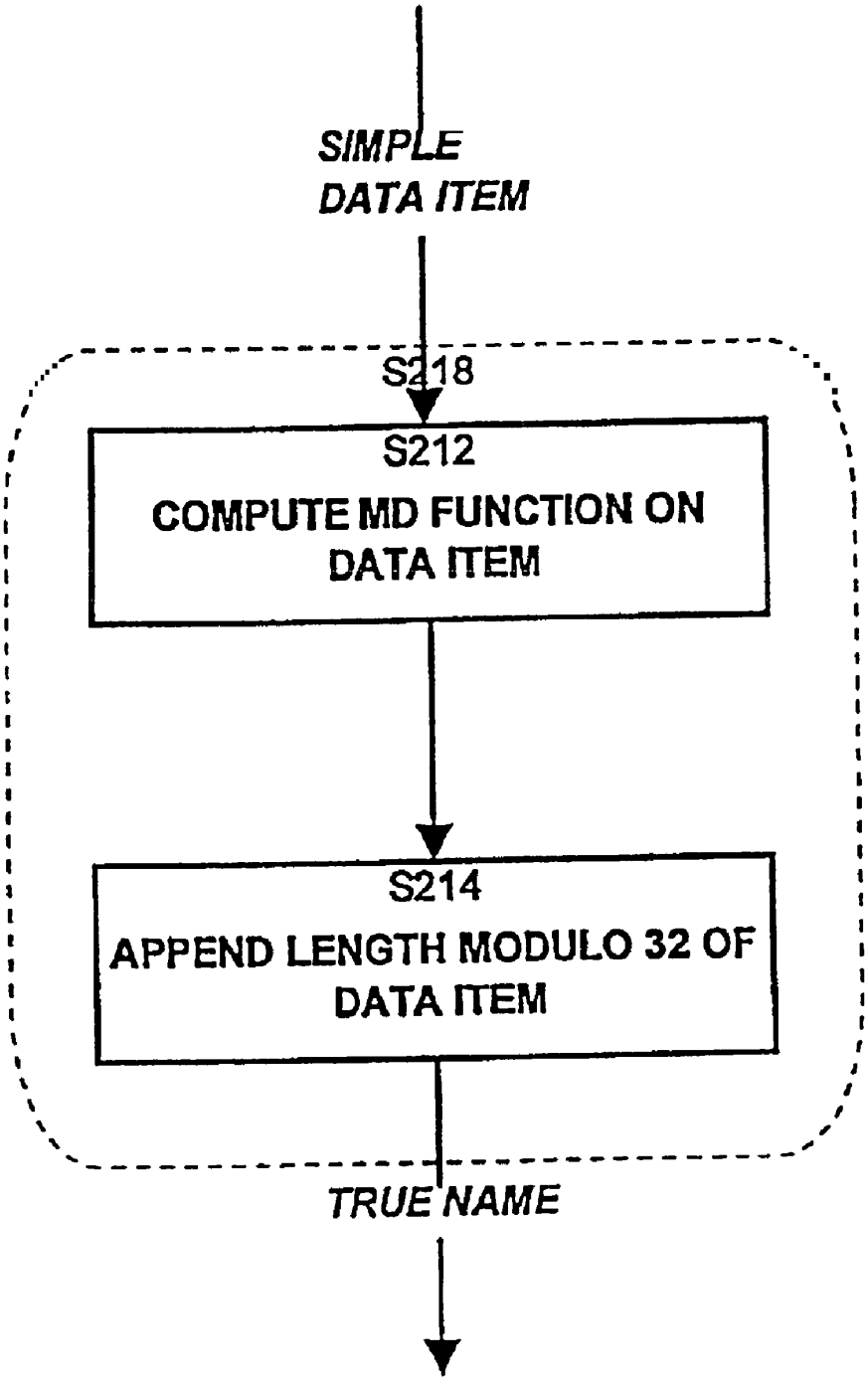


FIG. 9

FIG. 10(a)


SIMPLE
DATA ITEM
S218
S212
COMPUTE MD FUNCTION ON DATA ITEM
S214
APPEND LENGTH MODULO 32 OF DATA ITEM
TRUE NAME

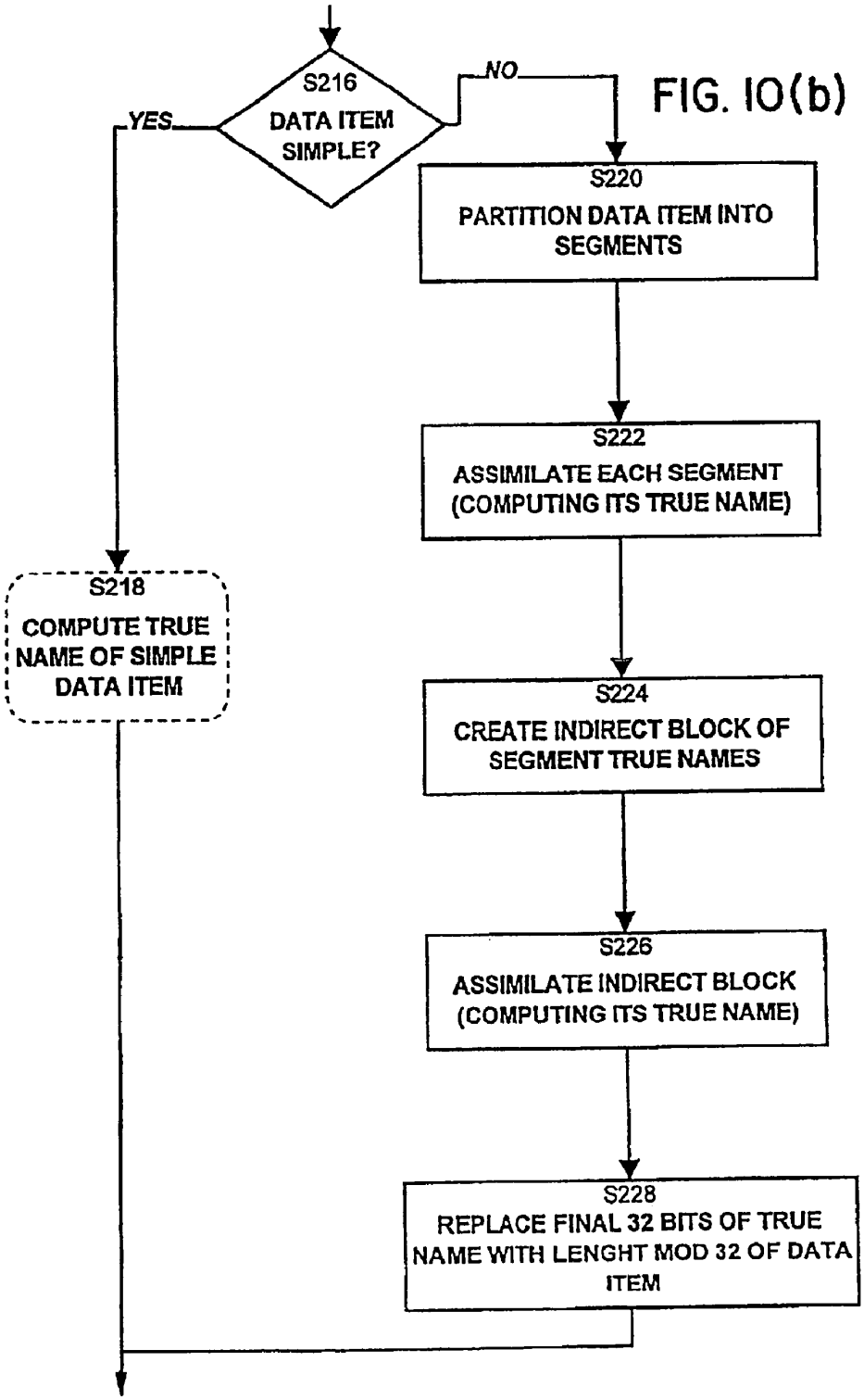

S216
DATA ITEM SIMPLE?
YES
NO
S218
COMPUTE TRUE NAME OF SIMPLE DATA ITEM
S220
PARTITION DATA ITEM INTO SEGMENTS
S222
ASSIMILATE EACH SEGMENT (COMPUTING ITS TRUE NAME)
S224
CREATE INDIRECT BLOCK OF SEGMENT TRUE NAMES
S226
ASSIMILATE INDIRECT BLOCK (COMPUTING ITS TRUE NAME)
S228
REPLACE FINAL 32 BITS OF TRUE NAME WITH LENGHT MOD 32 OF DATA ITEM

FIG. 10(b)

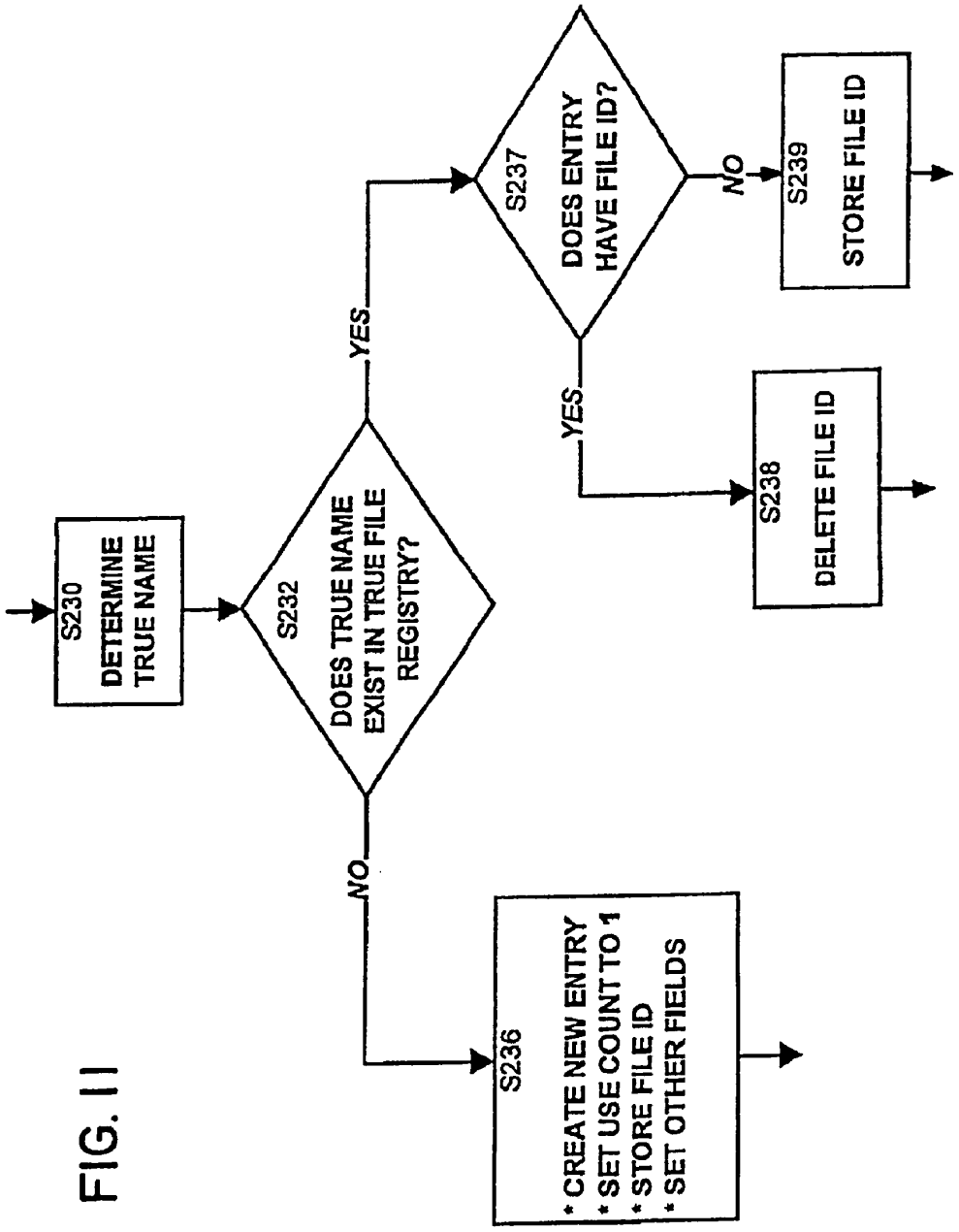

S230
DETERMINE TRUE NAME
S232
DOES TRUE NAME EXIST IN TRUE FILE REGISTRY?
YES
NO
S237
DOES ENTRY HAVE FILE ID?
YES
NO
S238
DELETE FILE ID
S239
STORE FILE ID
S236
* CREATE NEW ENTRY
* SET USE COUNT TO 1
* STORE FILE ID
* SET OTHER FIELDS

FIG. 11

FIG. 12

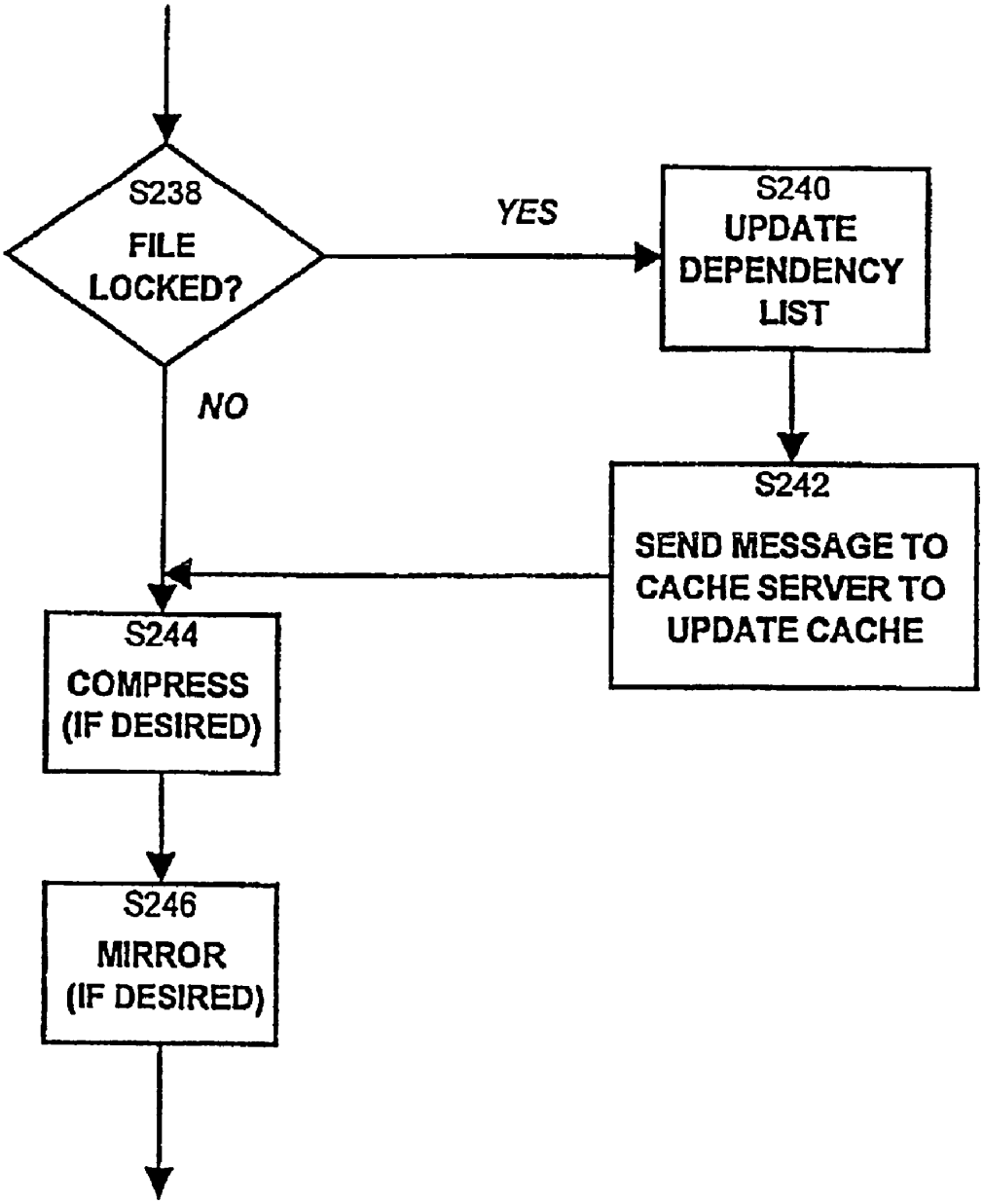
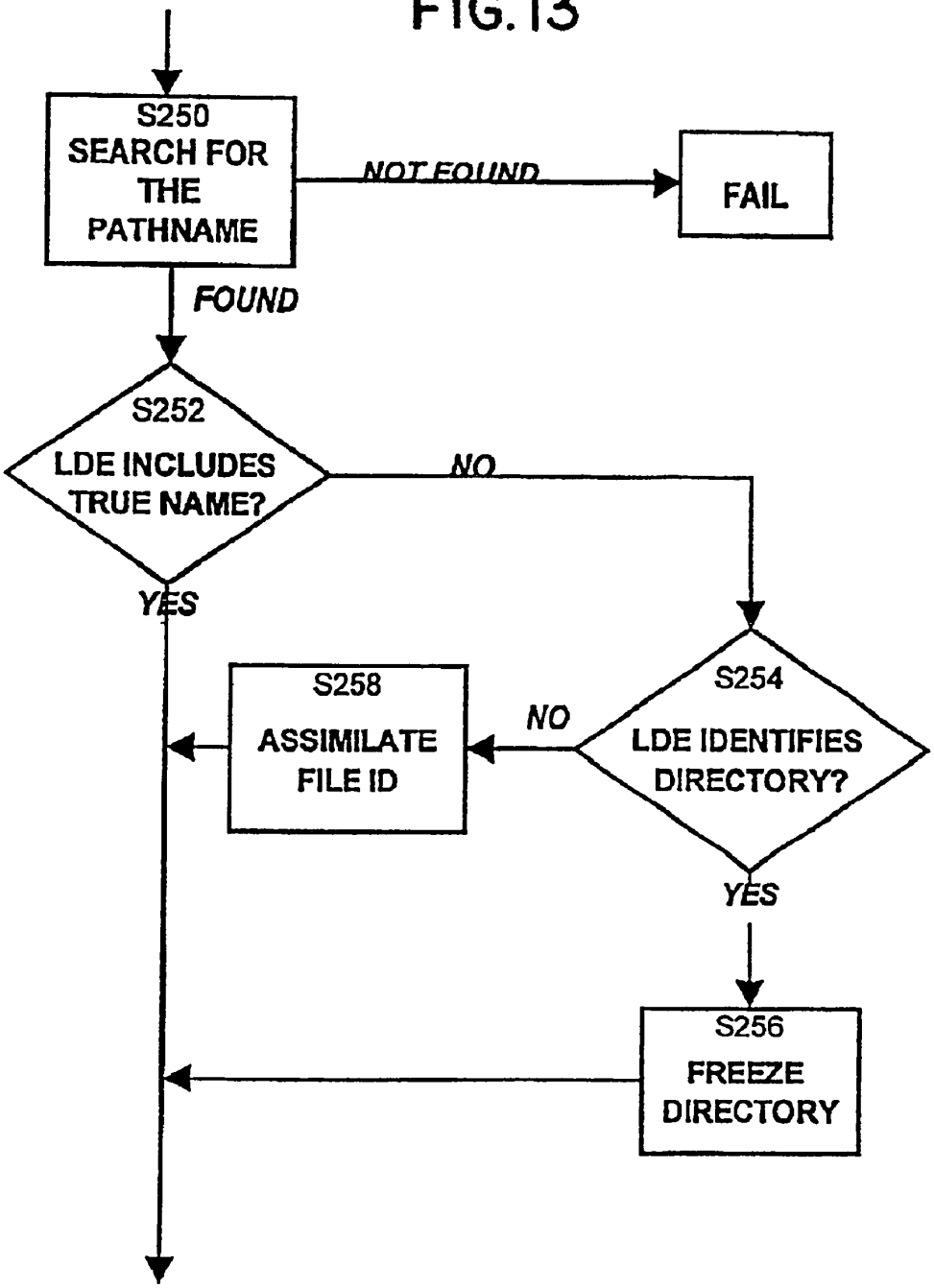

S238
FILE LOCKED?
YES
S240
UPDATE DEPENDENCY LIST
NO
S242
SEND MESSAGE TO CACHE SERVER TO UPDATE CACHE
S244
COMPRESS (IF DESIRED)
S246
MIRROR (IF DESIRED)

FIG. 13


S250
SEARCH FOR THE PATHNAME
NOT FOUND
FAIL
FOUND
S252
LDE INCLUDES TRUE NAME?
NO
YES
S254
LDE IDENTIFIES DIRECTORY?
NO
S258
ASSIMILATE FILE ID
YES
S256
FREEZE DIRECTORY

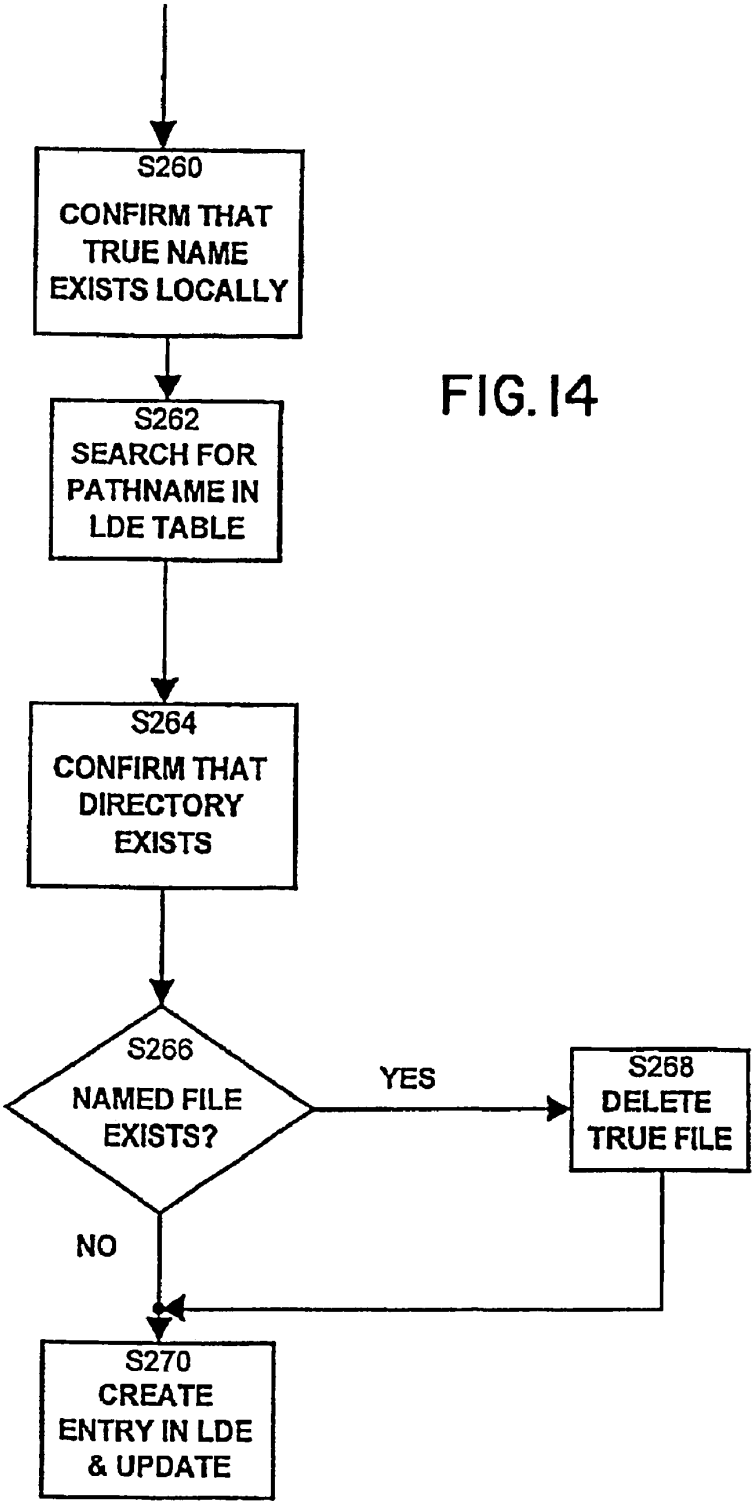

S260
CONFIRM THAT TRUE NAME EXISTS LOCALLY
S262
SEARCH FOR PATHNAME IN LDE TABLE
S264
CONFIRM THAT DIRECTORY EXISTS
S266
NAMED FILE EXISTS?
YES
S268
DELETE TRUE FILE
NO
S270
CREATE ENTRY IN LDE & UPDATE

FIG. 14

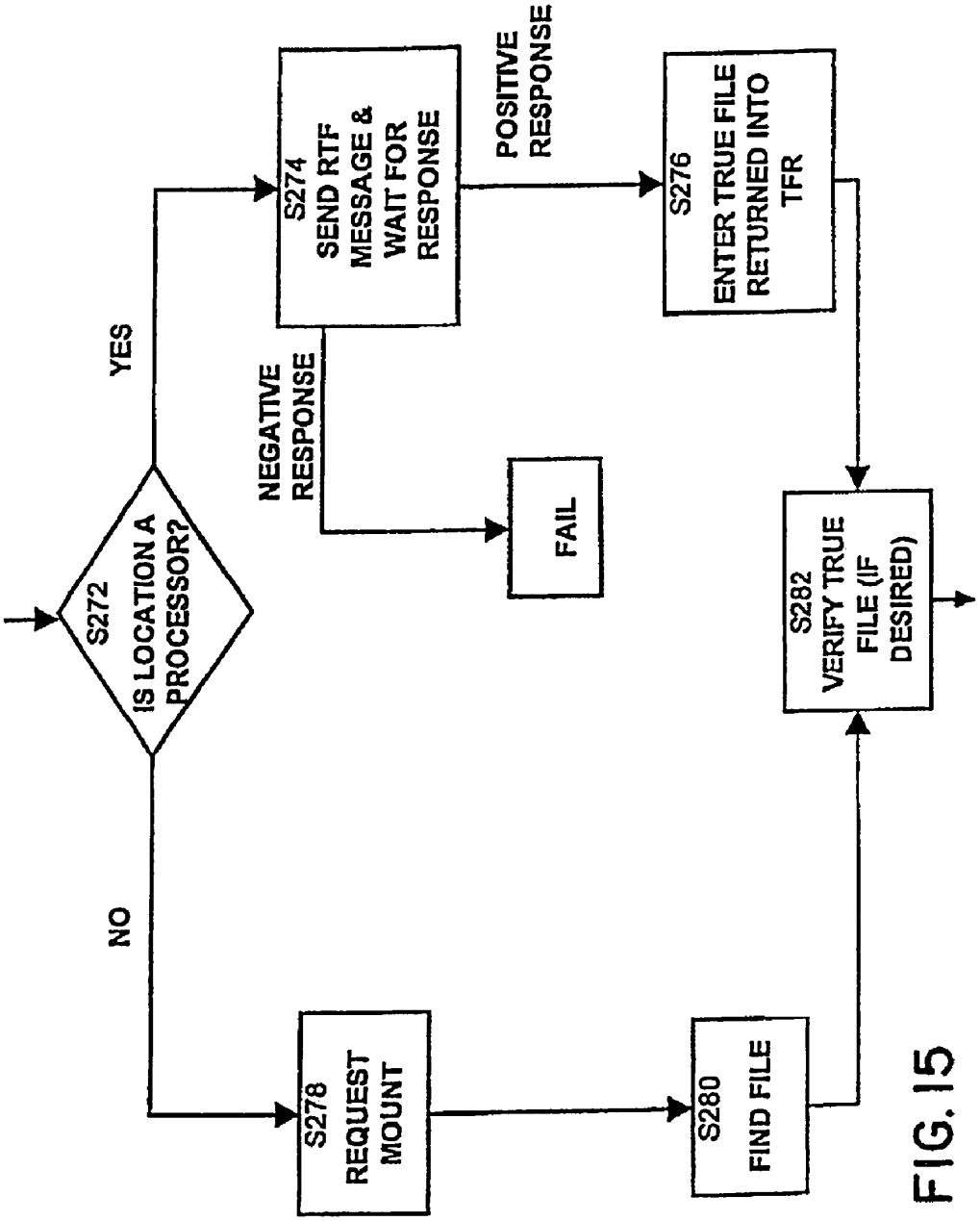

S272
IS LOCATION A PROCESSOR?
YES
NO
S274
SEND RTF MESSAGE & WAIT FOR RESPONSE
POSITIVE RESPONSE
NEGATIVE RESPONSE
FAIL
S276
ENTER TRUE FILE RETURNED INTO TFR
S278
REQUEST MOUNT
S280
FIND FILE
S282
VERIFY TRUE FILE (IF DESIRED)

FIG. 15

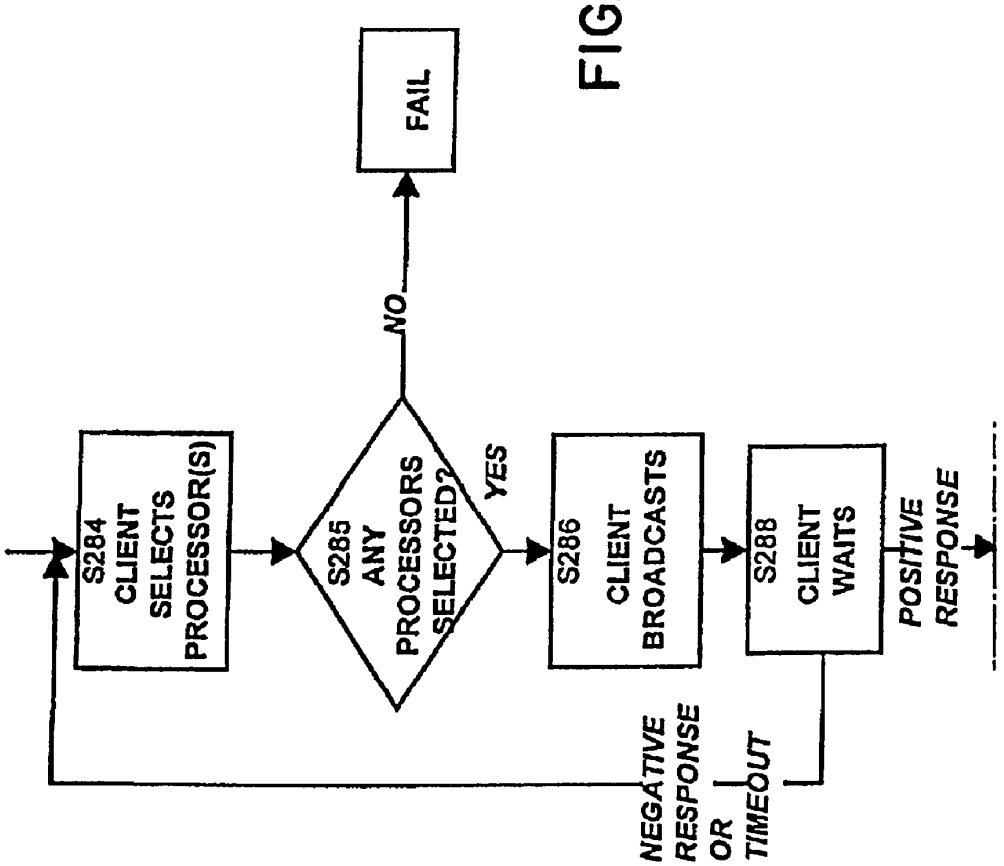

S284
CLIENT
SELECTS
PROCESSOR(S)
S285
ANY
PROCESSORS
SELECTED?
NO
FAIL
YES
S286
CLIENT
BROADCASTS
S288
CLIENT
WAITS
POSITIVE
RESPONSE
NEGATIVE
RESPONSE
OR
TIMEOUT

FIG. 16(a)

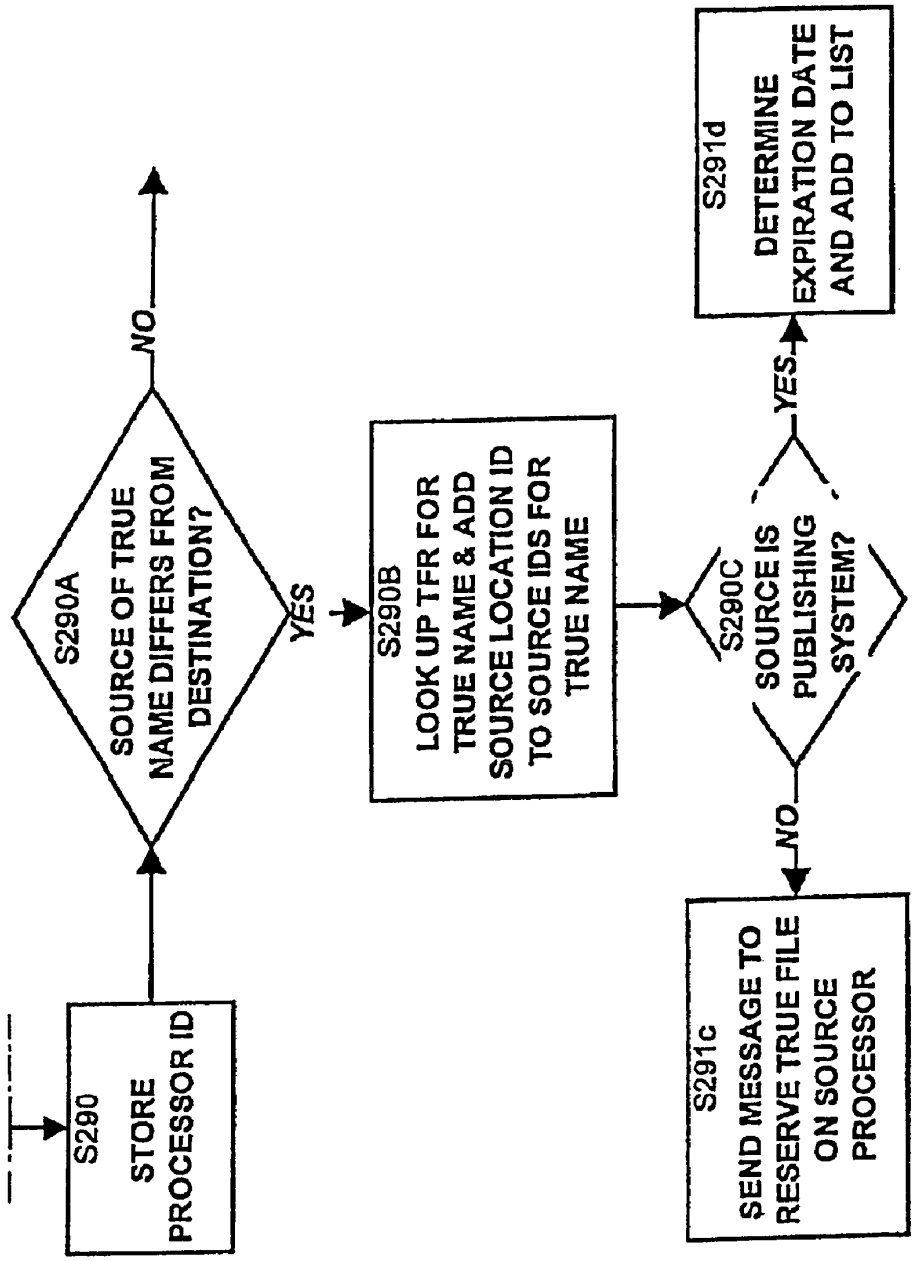

S290
STORE PROCESSOR ID
S290A
SOURCE OF TRUE NAME DIFFERS FROM DESTINATION?
NO
YES
S290B
LOOK UP TFR FOR TRUE NAME & ADD SOURCE LOCATION ID TO SOURCE IDS FOR TRUE NAME
S290C
SOURCE IS PUBLISHING SYSTEM?
YES
NO
S291d
DETERMINE EXPIRATION DATE AND ADD TO LIST
S291c
SEND MESSAGE TO RESERVE TRUE FILE ON SOURCE PROCESSOR

FIG. 16(b)

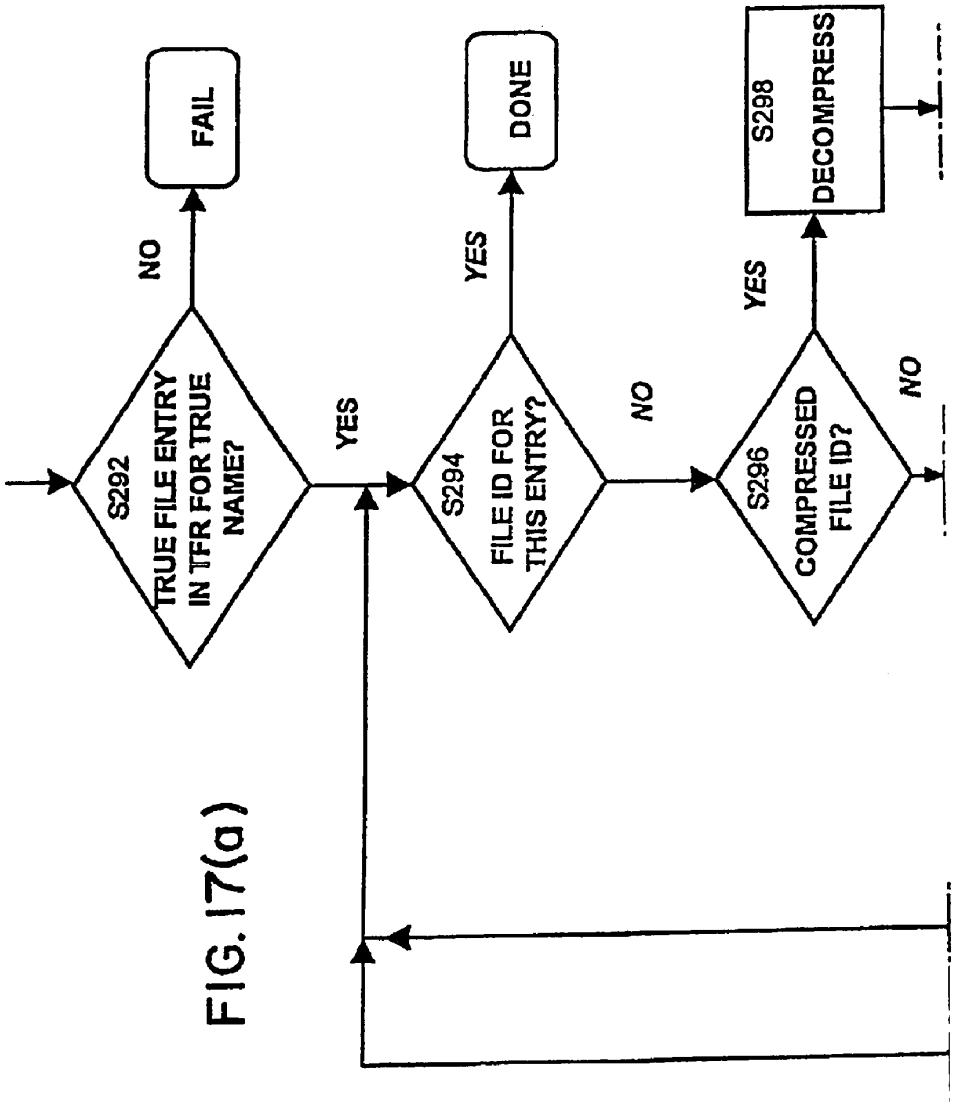

S292
TRUE FILE ENTRY IN TFR FOR TRUE NAME?
NO
FAIL
YES
S294
FILE ID FOR THIS ENTRY?
YES
DONE
NO
S296
COMPRESSED FILE ID?
YES
S298
DECOMPRESS
NO

FIG. 17(a)

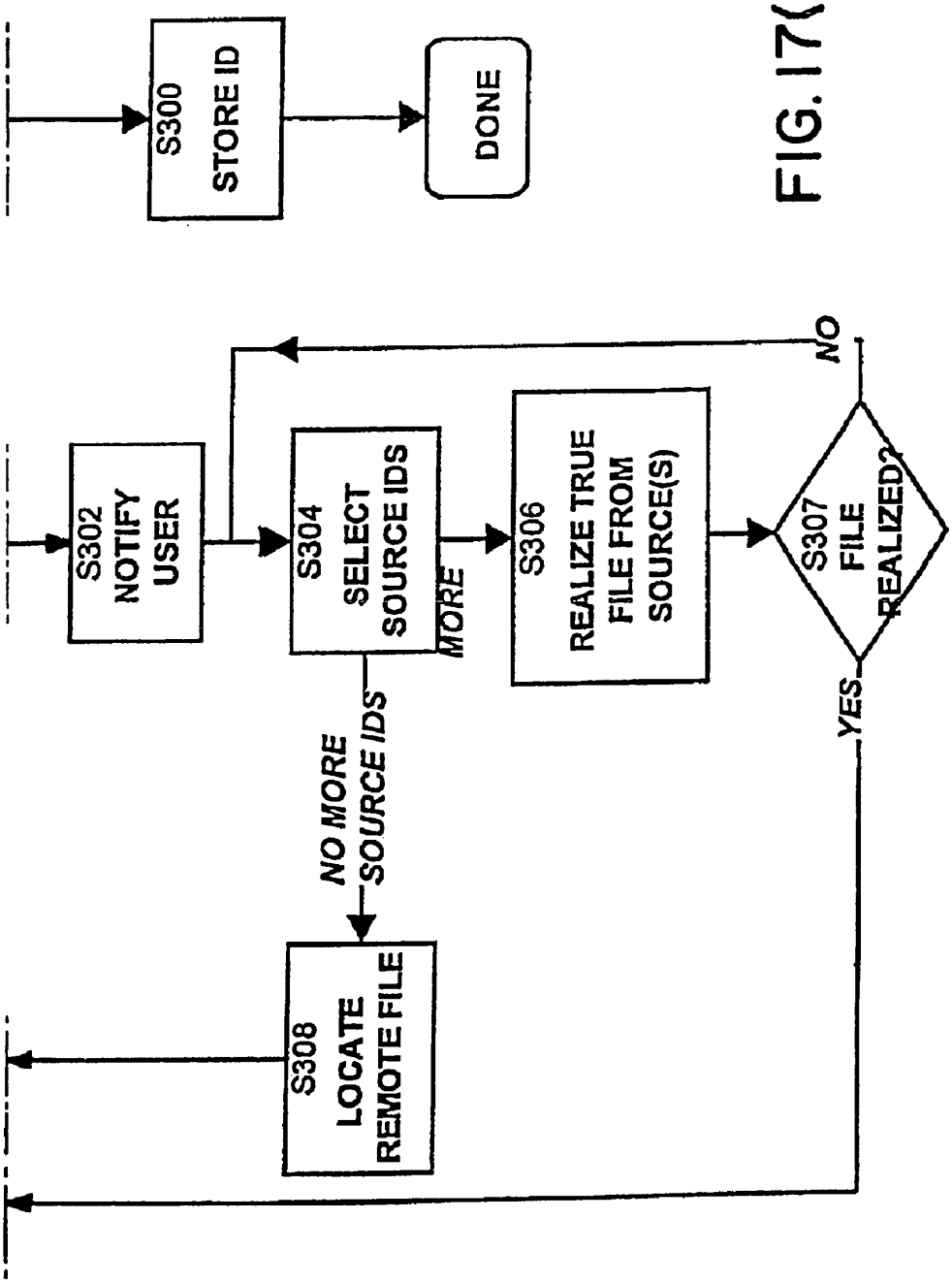
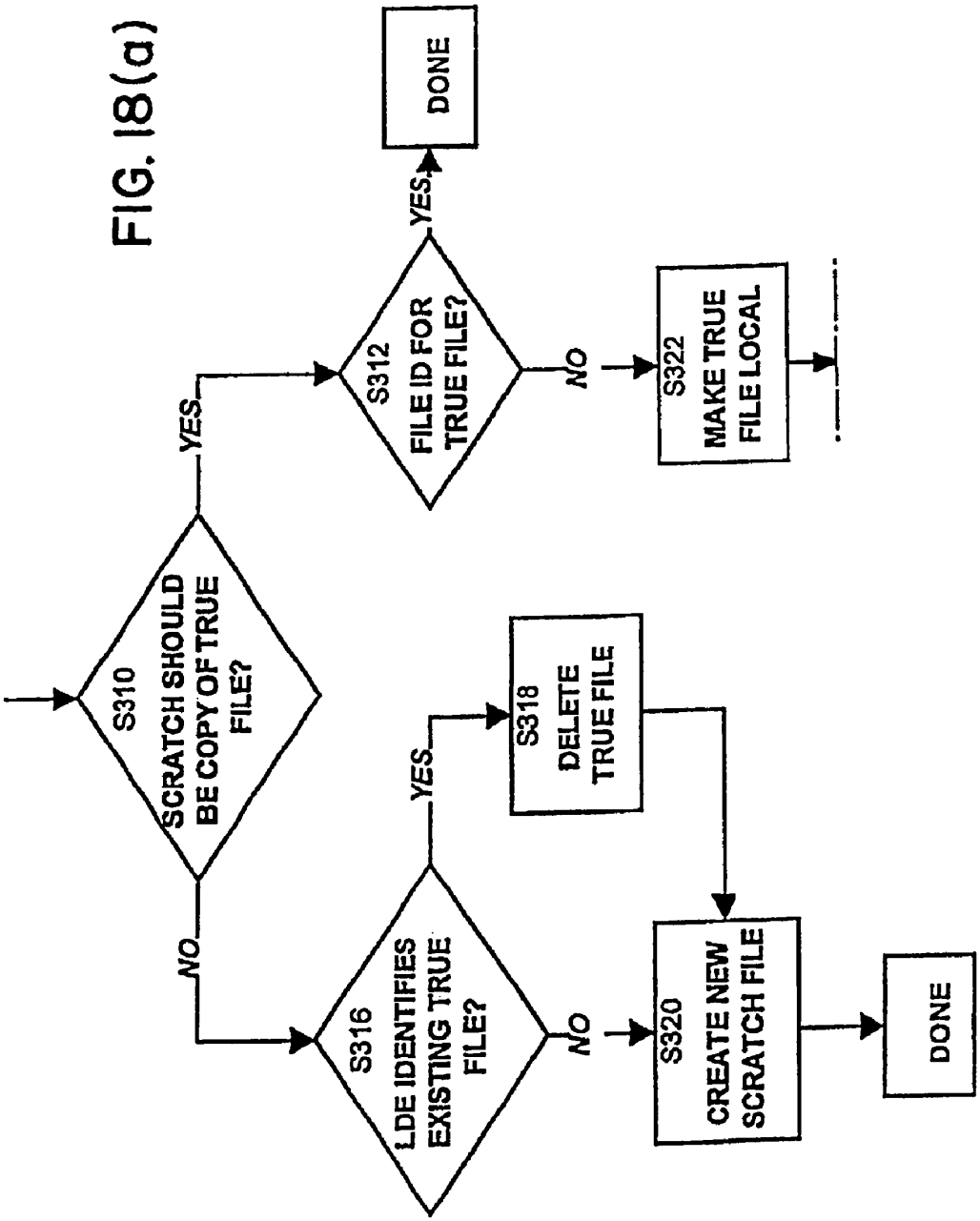

S300
STORE ID
DONE
S302
NOTIFY USER
S304
SELECT SOURCE IDS
MORE
NO MORE SOURCE IDS
S306
REALIZE TRUE FILE FROM SOURCE(S)
S307
FILE REALIZED?
NO
YES
S308
LOCATE REMOTE FILE

FIG. 17(b)

FIG. 18(a)


S310
SCRATCH SHOULD BE COPY OF TRUE FILE?
YES
NO
S312
FILE ID FOR TRUE FILE?
YES
NO
DONE
S322
MAKE TRUE FILE LOCAL
S316
LDE IDENTIFIES EXISTING TRUE FILE?
YES
NO
S318
DELETE TRUE FILE
S320
CREATE NEW SCRATCH FILE
DONE

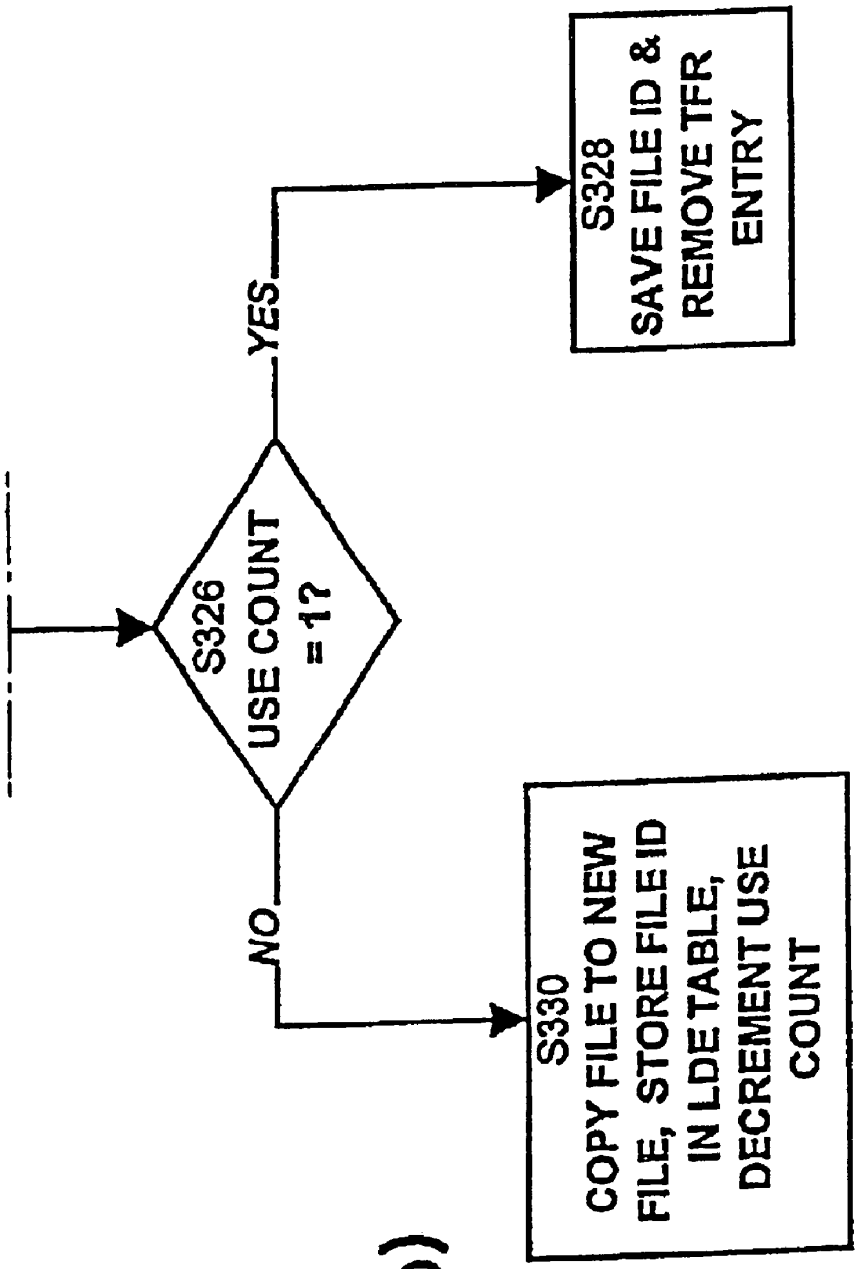
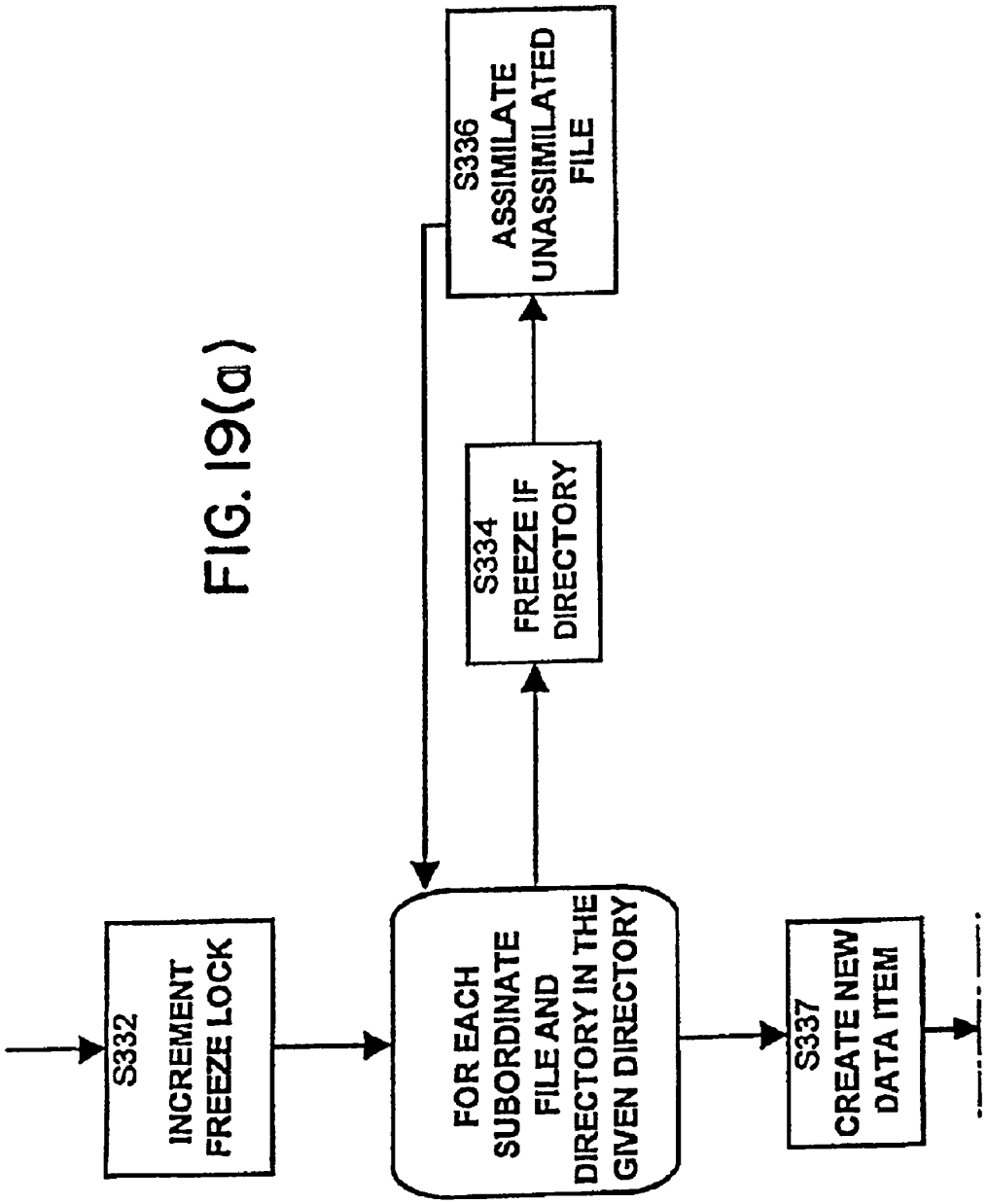

S326
USE COUNT
= 1?
YES
NO
S328
SAVE FILE ID &
REMOVE TFR
ENTRY
S330
COPY FILE TO NEW
FILE, STORE FILE ID
IN LDE TABLE,
DECREMENT USE
COUNT

FIG. 18(b)

FIG. 19(a)


S332
INCREMENT FREEZE LOCK
FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S334
FREEZE IF DIRECTORY
S336
ASSIMILATE UNASSIMILATED FILE
S337
CREATE NEW DATA ITEM

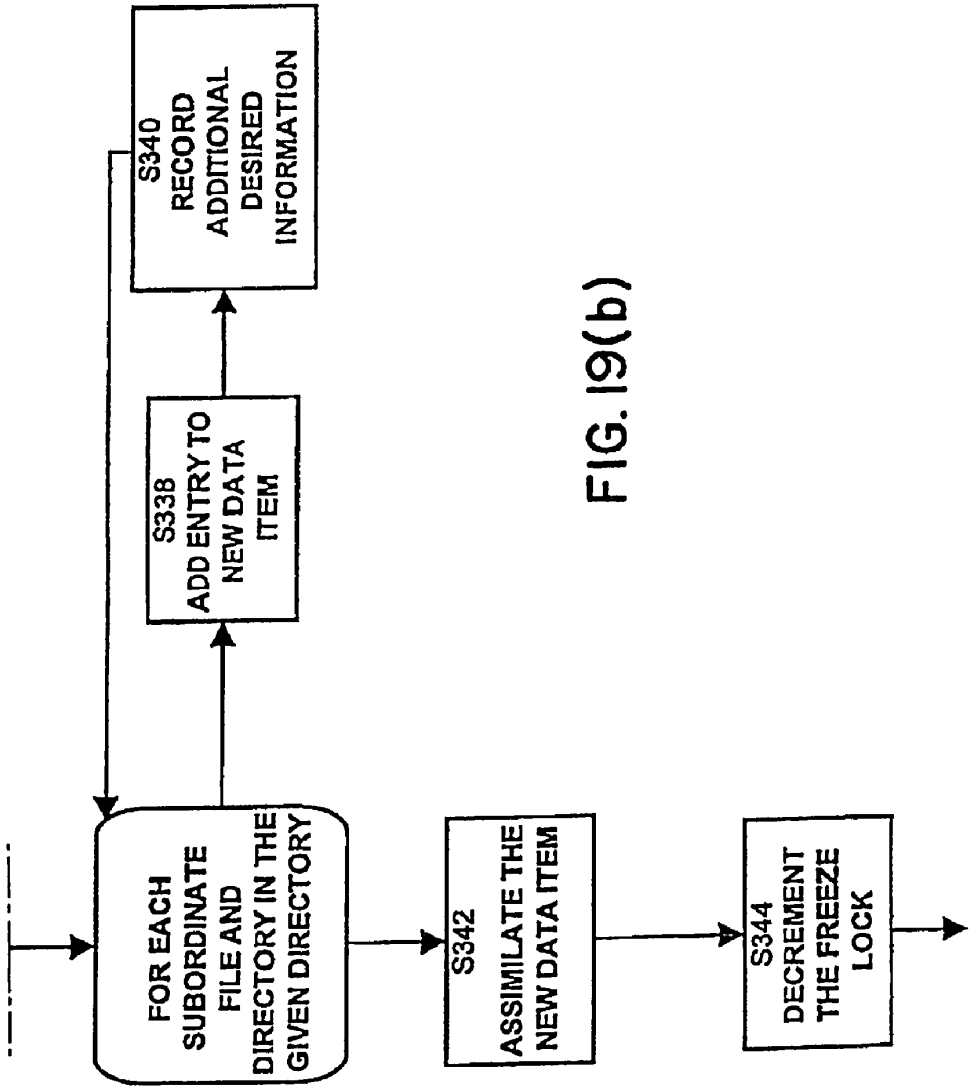

FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S338 ADD ENTRY TO NEW DATA ITEM
S340 RECORD ADDITIONAL DESIRED INFORMATION
S342 ASSIMILATE THE NEW DATA ITEM
S344 DECREMENT THE FREEZE LOCK

FIG. 19(b)

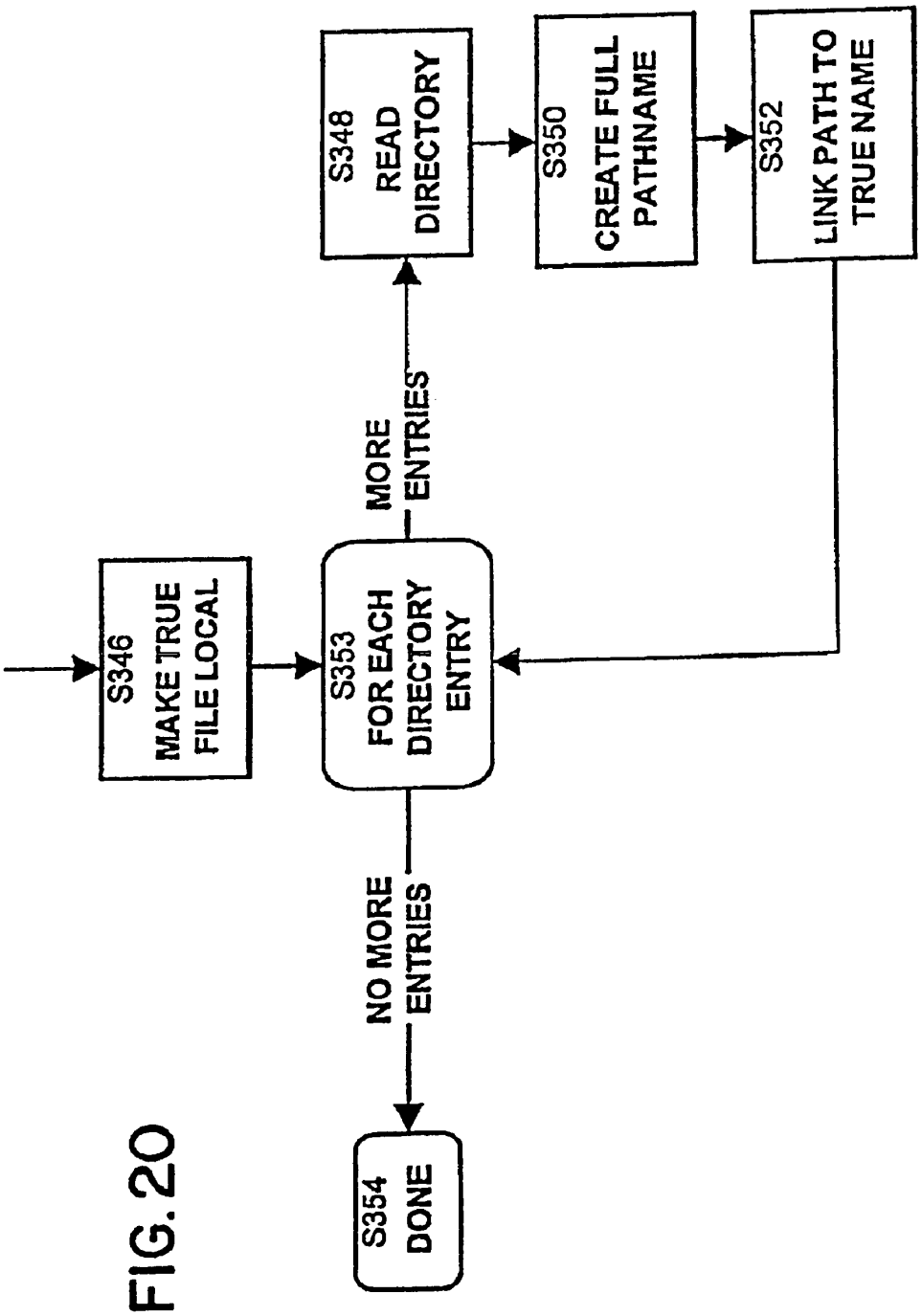

S346
MAKE TRUE FILE LOCAL
S353
FOR EACH DIRECTORY ENTRY
MORE ENTRIES
S348
READ DIRECTORY
S350
CREATE FULL PATHNAME
S352
LINK PATH TO TRUE NAME
NO MORE ENTRIES
S354
DONE

FIG. 20

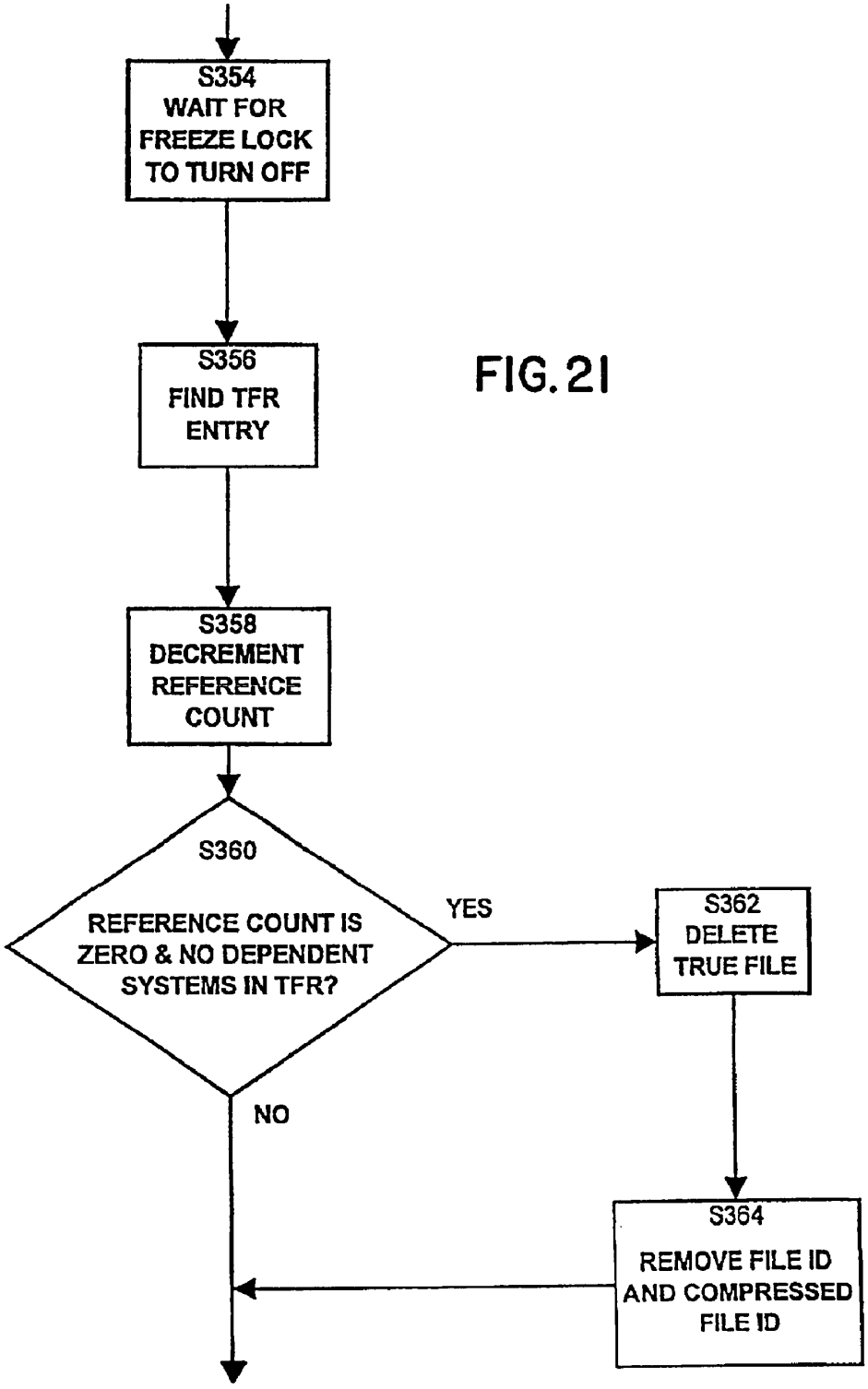

S354
WAIT FOR FREEZE LOCK TO TURN OFF
S356
FIND TFR ENTRY
S358
DECREMENT REFERENCE COUNT
S360
REFERENCE COUNT IS ZERO & NO DEPENDENT SYSTEMS IN TFR?
YES
S362
DELETE TRUE FILE
NO
S364
REMOVE FILE ID AND COMPRESSED FILE ID

FIG. 21

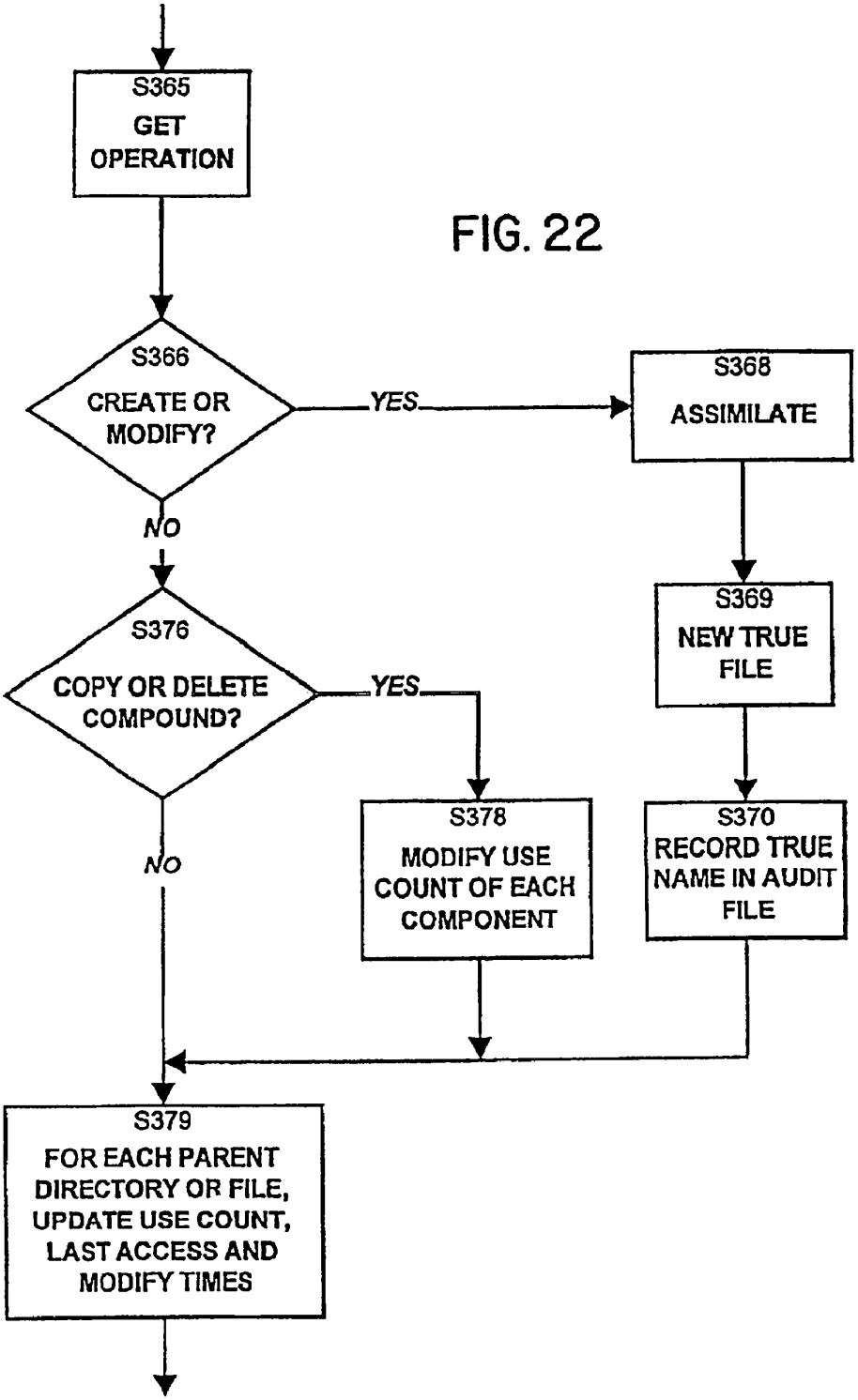
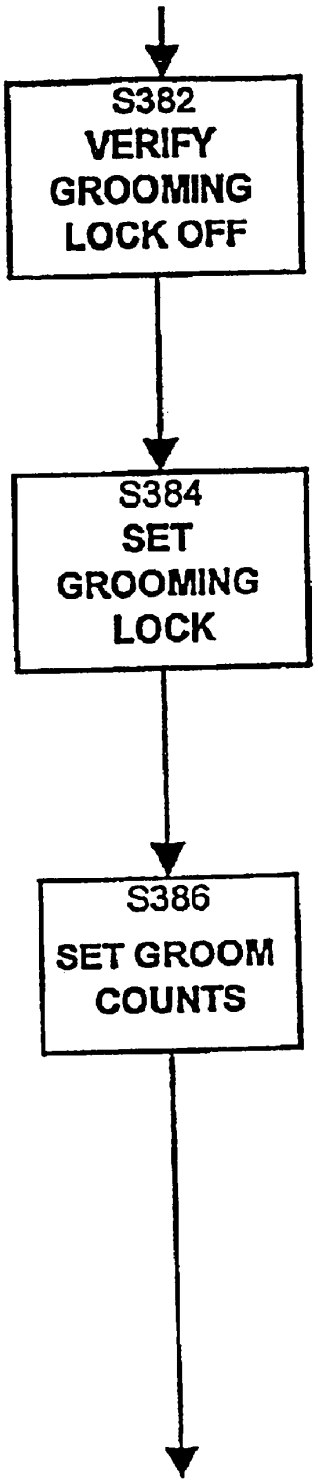

S365
GET OPERATION
S366
CREATE OR MODIFY?
YES
S368
ASSIMILATE
NO
S369
NEW TRUE FILE
S376
COPY OR DELETE COMPOUND?
YES
S378
MODIFY USE COUNT OF EACH COMPONENT
S370
RECORD TRUE NAME IN AUDIT FILE
NO
S379
FOR EACH PARENT DIRECTORY OR FILE, UPDATE USE COUNT, LAST ACCESS AND MODIFY TIMES

FIG. 22

FIG. 23


S382
VERIFY GROOMING LOCK OFF
S384
SET GROOMING LOCK
S386
SET GROOM COUNTS

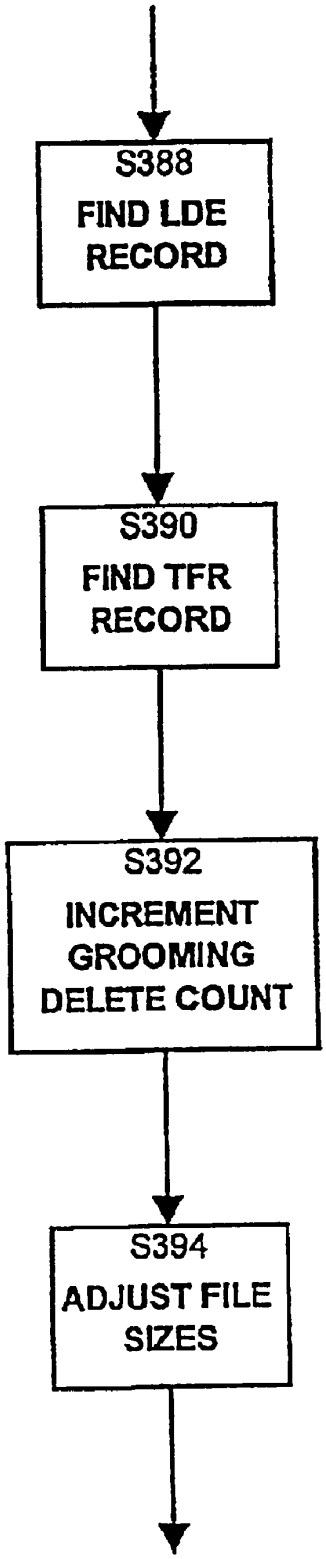

S388
FIND LDE RECORD
S390
FIND TFR RECORD
S392
INCREMENT GROOMING DELETE COUNT
S394
ADJUST FILE SIZES

FIG. 24

FIG. 25

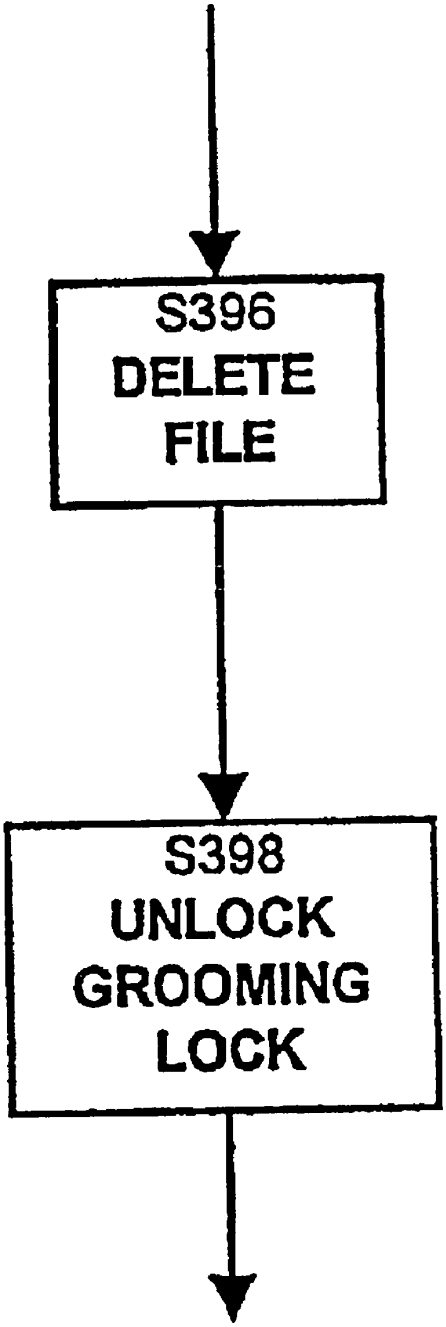
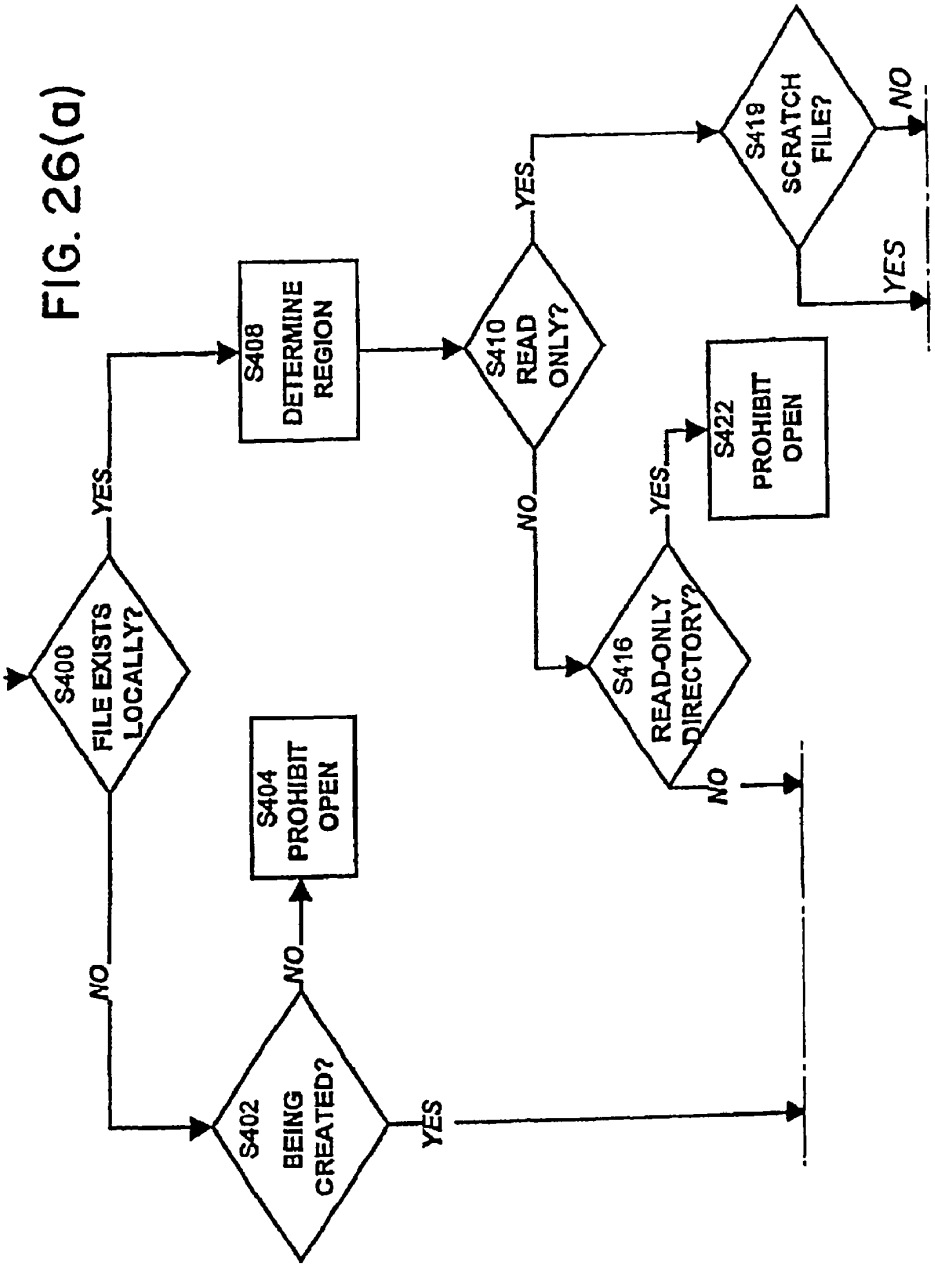


FIG. 26(a)


S400
FILE EXISTS LOCALLY?
YES
NO
S408
DETERMINE REGION
S410
READ ONLY?
YES
NO
S418
SCRATCH FILE?
NO
YES
S422
PROHIBIT OPEN
YES
S416
READ-ONLY DIRECTORY?
NO
S404
PROHIBIT OPEN
NO
S402
BEING CREATED?
YES

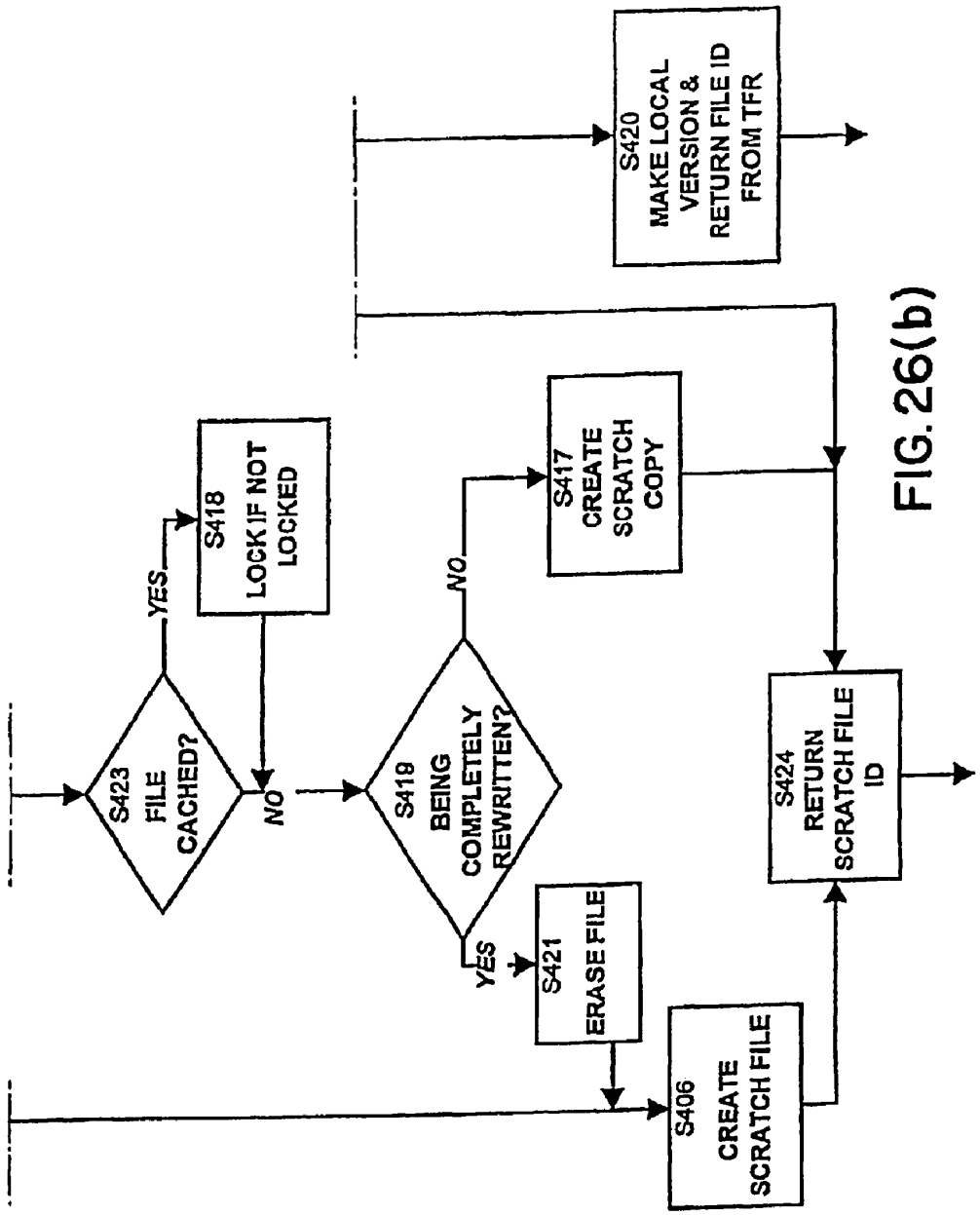

S420
MAKE LOCAL VERSION & RETURN FILE ID FROM TFR
S417
CREATE SCRATCH COPY
S418
LOCK IF NOT LOCKED
YES
NO
S423
FILE CACHED?
NO
S419
BEING COMPLETELY REWRITTEN?
YES
S421
ERASE FILE
S406
CREATE SCRATCH FILE
S424
RETURN SCRATCH FILE ID

FIG. 26(b)

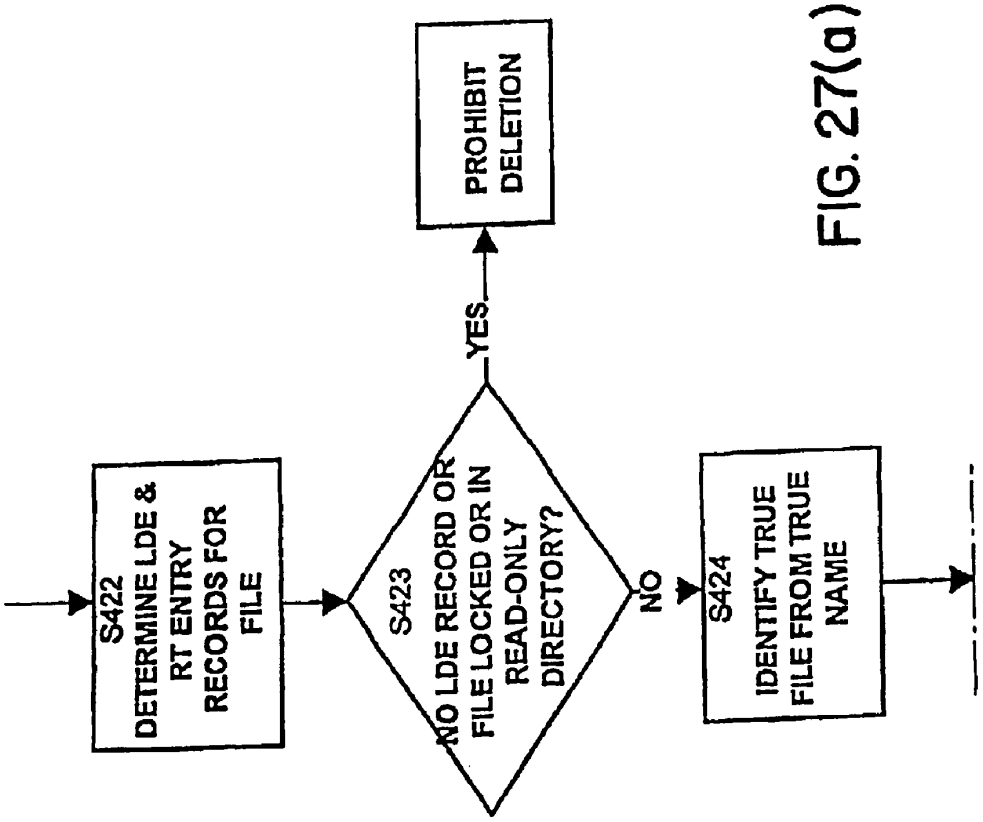

S422
DETERMINE LDE & RT ENTRY RECORDS FOR FILE
S423
NO LDE RECORD OR FILE LOCKED OR IN READ-ONLY DIRECTORY?
YES
PROHIBIT DELETION
NO
S424
IDENTIFY TRUE FILE FROM TRUE NAME

FIG. 27(a)

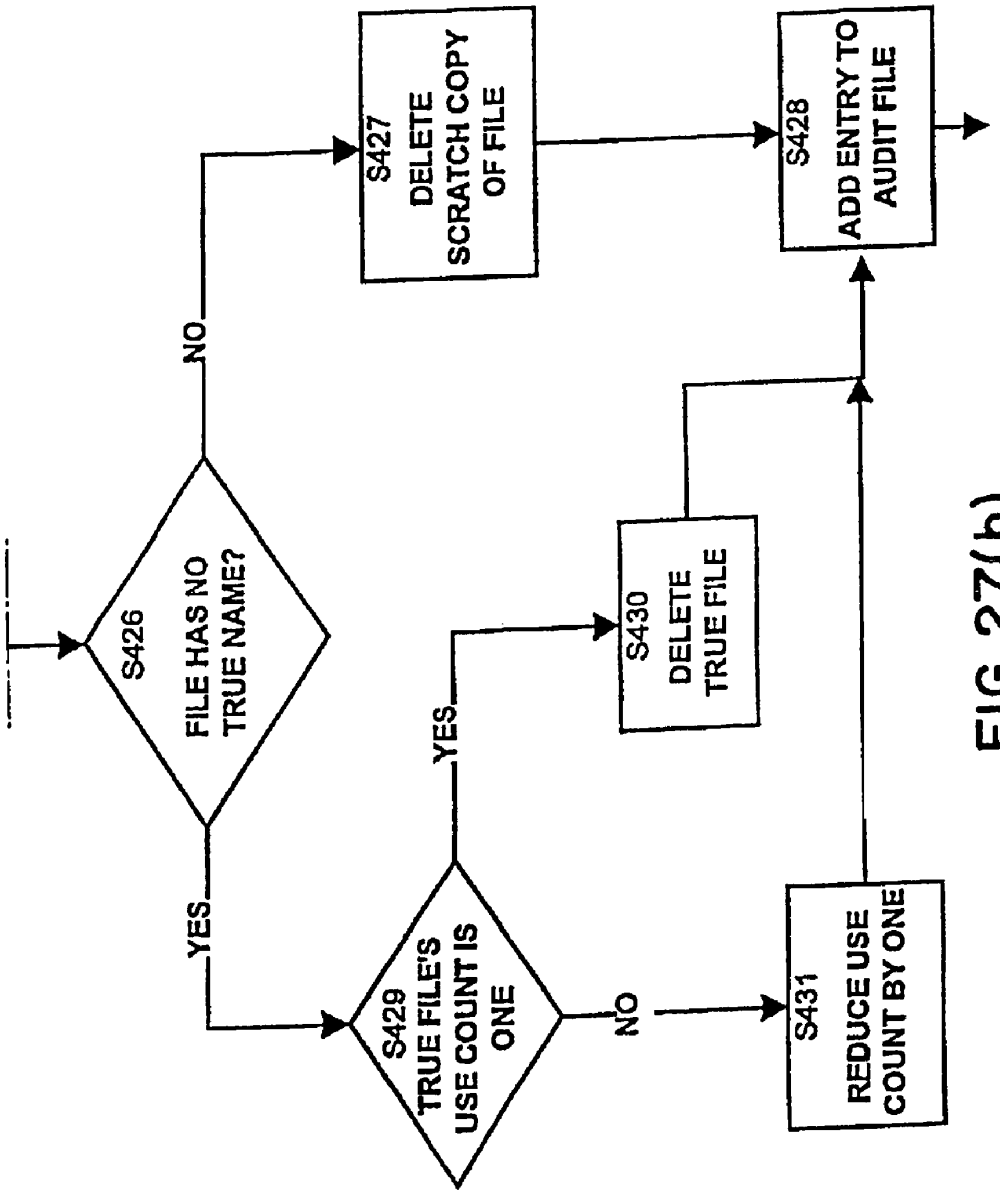
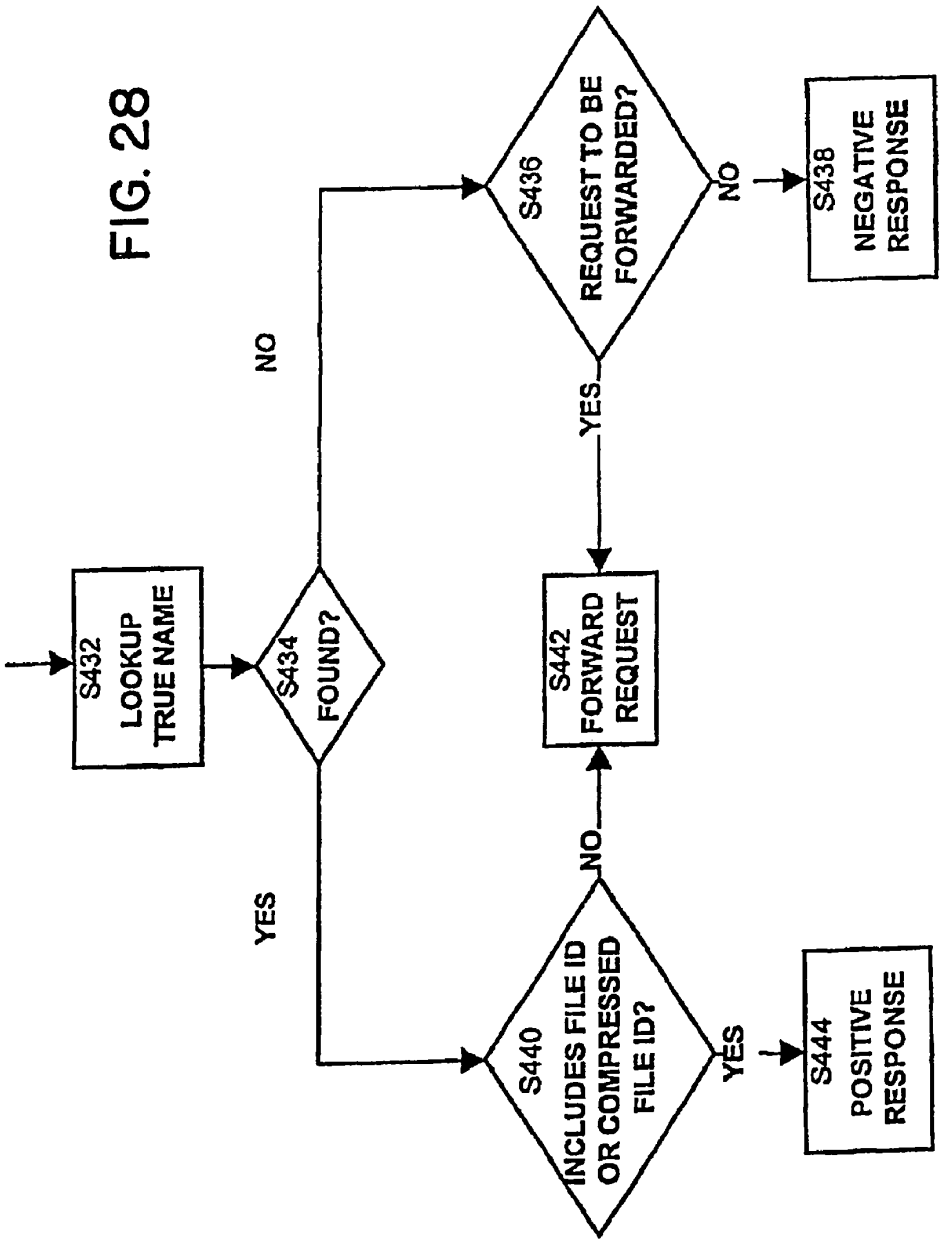

S426
FILE HAS NO TRUE NAME?
NO
YES
S427
DELETE SCRATCH COPY OF FILE
S428
ADD ENTRY TO AUDIT FILE
S429
TRUE FILE'S USE COUNT IS ONE
YES
NO
S430
DELETE TRUE FILE
S431
REDUCE USE COUNT BY ONE

FIG. 27(b)

FIG. 28


S432
LOOKUP TRUE NAME
S434
FOUND?
NO
YES
S436
REQUEST TO BE FORWARDED?
NO
S438
NEGATIVE RESPONSE
YES
S442
FORWARD REQUEST
NO
S440
INCLUDES FILE ID OR COMPRESSED FILE ID?
YES
S444
POSITIVE RESPONSE

CONTROLLING ACCESS TO DATA IN A DATA PROCESSING SYSTEM

RELATED APPLICATIONS

This application is a continuation of an claims priority to pending U.S. patent application Ser. No. 11/724,232, which is a continuation of application Ser. No. 11/017,650, filed Dec. 22, 2004, which is a continuation of pending application Ser. No. 10/742,972, filed Dec. 23, 2003, which is a continuation of Ser. No. 09/987,723, filed Nov. 15, 2001, patented as U.S. Pat. No. 6,928,442; which is a which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference. This application is a continuation of and claims priority to application Ser. No. 11/017,650, filed Dec. 22, 2004, which is a continuation of application Ser. No. 09/987,723, filed Nov. 15, 2001, now U.S. Pat. No. 6,928,442, which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference. This is also a continuation of and claims priority to application Ser. No. 10/742,972, filed Dec. 23, 2003, which is a division of application Ser. No. 09/987,723, filed Nov. 15, 2001, now U.S. Pat. No. 6,928,442, which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to data processing systems and, more particularly, to data processing systems wherein data items are identified by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items.

2. Background of the Invention

Data processing (DP) systems, computers, networks of computers, or the like, typically offer users and programs various ways to identify the data in the systems.

Users typically identify data in the data processing system by giving the data some form of name. For example, a typical operating system (OS) on a computer provides a file system in which data items are named by alphanumeric identifiers. Programs typically identify data in the data processing system using a location or address. For example, a program may identify a record in a file or database by using a record number which serves to locate that record.

In all but the most primitive operating systems, users and programs are able to create and use collections of named data items, these collections themselves being named by identifiers. These named collections can then, themselves, be made part of other named collections. For example, an OS may provide mechanisms to group files (data items) into directories (collections). These directories can then, themselves be made part of other directories. A data item may thus be identified relative to these nested directories using a sequence of names, or a so-called pathname, which defines a path through the directories to a particular data item (file or directory).

As another example, a database management system may group data records (data items) into tables and then group these tables into database files (collections). The complete address of any data record can then be specified using the database file name, the table name, and the record number of that data record.

Other examples of identifying data items include: identifying files in a network file system, identifying objects in an object-oriented database, identifying images in an image database, and identifying articles in a text database.

In general, the terms "data" and "data item" as used herein refer to sequences of bits. Thus a data item may be the contents of a file, a portion of a file, a page in memory, an object in an object-oriented program, a digital message, a digital scanned image, a part of a video or audio signal, or any other entity which can be represented by a sequence of bits. The term "data processing" herein refers to the processing of data items, and is sometimes dependent on the type of data item being processed. For example, a data processor for a digital image may differ from a data processor for an audio signal.

In all of the prior data processing systems the names or identifiers provided to identify data items (the data items being files, directories, records in the database, objects in object-oriented programming, locations in memory or on a physical device, or the like) are always defined relative to a specific context. For instance, the file identified by a particular file name can only be determined when the directory containing the file (the context) is known. The file identified by a pathname can be determined only when the file system (context) is known. Similarly, the addresses in a process address space, the keys in a database table, or domain names on a global computer network such as the Internet are meaningful only because they are specified relative to a context.

In prior art systems for identifying data items there is no direct relationship between the data names and the data item. The same data name in two different contexts may refer to different data items, and two different data names in the same context may refer to the same data item.

In addition, because there is no correlation between a data name and the data it refers to, there is no a priori way to confirm that a given data item is in fact the one named by a data name. For instance, in a DP system, if one processor requests that another processor deliver a data item with a given data name, the requesting processor cannot, in general, verify that the data delivered is the correct data (given only the name). Therefore it may require further processing, typically on the part of the requester, to verify that the data item it has obtained is, in fact, the item it requested.

A common operation in a DP system is adding a new data item to the system. When a new data item is added to the system, a name can be assigned to it only by updating the context in which names are defined. Thus such systems require a centralized mechanism for the management of names. Such a mechanism is required even in a multi-processing system when data items are created and identified at separate processors in distinct locations, and in which there is no other need for communication when data items are added.

In many data processing systems or environments, data items are transferred between different locations in the system. These locations may be processors in the data processing system, storage devices, memory, or the like. For example, one processor may obtain a data item from another processor

or from an external storage device, such as a floppy disk, and may incorporate that data item into its system (using the name provided with that data item).

However, when a processor (or some location) obtains a data item from another location in the DP system, it is possible that this obtained data item is already present in the system (either at the location of the processor or at some other location accessible by the processor) and therefore a duplicate of the data item is created. This situation is common in a network data processing environment where proprietary software products are installed from floppy disks onto several processors sharing a common file server. In these systems, it is often the case that the same product will be installed on several systems, so that several copies of each file will reside on the common file server.

In some data processing systems in which several processors are connected in a network, one system is designated as a cache server to maintain master copies of data items, and other systems are designated as cache clients to copy local copies of the master data items into a local cache on an as-needed basis. Before using a cached item, a cache client must either reload the cached item, be informed of changes to the cached item, or confirm that the master item corresponding to the cached item has not changed. In other words, a cache client must synchronize its data items with those on the cache server. This synchronization may involve reloading data items onto the cache client. The need to keep the cache synchronized or reload it adds significant overhead to existing caching mechanisms.

In view of the above and other problems with prior art systems, it is therefore desirable to have a mechanism which allows each processor in a multiprocessor system to determine a common and substantially unique identifier for a data item, using only the data in the data item and not relying on any sort of context.

It is further desirable to have a mechanism for reducing multiple copies of data items in a data processing system and to have a mechanism which enables the identification of identical data items so as to reduce multiple copies. It is further desirable to determine whether two instances of a data item are in fact the same data item, and to perform various other systems' functions and applications on data items without relying on any context information or properties of the data item.

It is also desirable to provide such a mechanism in such a way as to make it transparent to users of the data processing system, and it is desirable that a single mechanism be used to address each of the problems described above.

SUMMARY OF THE INVENTION

This invention provides, in a data processing system, a method and apparatus for identifying a data item in the system, where the identity of the data item depends on all of the data in the data item and only on the data in the data item. Thus the identity of a data item is independent of its name, origin, location, address, or other information not derivable directly from the data, and depends only on the data itself.

This invention further provides an apparatus and a method for determining whether a particular data item is present in the system or at a location in the system, by examining only the data identities of a plurality of data items.

Using the method or apparatus of the present invention, the efficiency and integrity of a data processing system can be improved. The present invention improves the design and operation of a data storage system, file system, relational database, object-oriented database, or the like that stores a plurality of data items, by making possible or improving the design and operation of at least some or all of the following features:

the system stores at most one copy of any data item at a given location, even when multiple data names in the system refer to the same contents;

the system avoids copying data from source to destination locations when the destination locations already have the data;

the system provides transparent access to any data item by reference only to its identity and independent of its present location, whether it be local, remote, or offline;

the system caches data items from a server, so that only the most recently accessed data items need be retained;

when the system is being used to cache data items, problems of maintaining cache consistency are avoided;

the system maintains a desired level of redundancy of data items in a network of servers, to protect against failure by ensuring that multiple copies of the data items are present at different locations in the system;

the system automatically archives data items as they are created or modified;

the system provides the size, age, and location of groups of data items in order to decide whether they can be safely removed from a local file system;

the system can efficiently record and preserve any collection of data items;

the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items;

the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of the data items;

the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM, the inventory is independent of other properties of the data items such as their name, location, and date of creation;

the system allows closely related sets of data items, such as matching or corresponding directories on disconnected computers, to be periodically resynchronized with one another;

the system can verify that data retrieved from another location is the desired or requested data, using only the data identifier used to retrieve the data;

the system can prove possession of specific data items by content without disclosing the content of the data items, for purposes of later legal verification and to provide anonymity;

the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes.

Other objects, features, and characteristics of the present invention as well as the methods of operation and functions of the related elements of structure, and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1**(*a*) and **1**(*b*) depict a typical data processing system in which a preferred embodiment of the present invention operates;

FIG. **2** depicts a hierarchy of data items stored at any location in such a data processing system;

FIGS. **3-9** depict data structures used to implement an embodiment of the present invention; and

FIGS. **10**(*a*)-**28** are flow charts depicting operation of various aspects of the present invention.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

An embodiment of the present invention is now described with reference to a typical data processing system **100**, which, with reference to FIGS. **1**(*a*) and **1**(*b*), includes one or more processors (or computers) **102** and various storage devices **104** connected in some way, for example by a bus **106**.

Each processor **102** includes a CPU **108**, a memory **110** and one or more local storage devices **112**. The CPU **108**, memory **110**, and local storage device **112** may be internally connected, for example by a bus **114**. Each processor **102** may also include other devices (not shown), such as a keyboard, a display, a printer, and the like.

In a data processing system **100**, wherein more than one processor **102** is used, that is, in a multiprocessor system, the processors may be in one of various relationships. For example, two processors **102** may be in a client/server, client/client, or a server/server relationship. These inter-processor relationships may be dynamic, changing depending on particular situations and functions. Thus, a particular processor **102** may change its relationship to other processors as needed, essentially setting up a peer-to-peer relationship with other processors. In a peer-to-peer relationship, sometimes a particular processor **102** acts as a client processor, whereas at other times the same processor acts as a server processor. In other words, there is no hierarchy imposed on or required of processors **102**.

In a multiprocessor system, the processors **102** may be homogeneous or heterogeneous. Further, in a multiprocessor data processing system **100**, some or all of the processors **102** may be disconnected from the network of processors for periods of time. Such disconnection may be part of the normal operation of the system **100** or it may be because a particular processor **102** is in need of repair.

Within a data processing system **100**, the data may be organized to form a hierarchy of data storage elements, wherein lower level data storage elements are combined to form higher level elements. This hierarchy can consist of, for example, processors, file systems, regions, directories, data files, segments, and the like. For example, with reference to FIG. **2**, the data items on a particular processor **102** may be organized or structured as a file system **116** which comprises regions **117**, each of which comprises directories **118**, each of which can contain other directories **118** or files **120**. Each file **120** being made up of one or more data segments **122**.

In a typical data processing system, some or all of these elements can be named by users given certain implementation specific naming conventions, the name (or pathname) of an element being relative to a context. In the context of a data processing system **100**, a pathname is fully specified by a processor name, a filesystem name, a sequence of zero or more directory names identifying nested directories, and a final file name. (Usually the lowest level elements, in this case segments **122**, cannot be named by users.)

In other words, a file system **116** is a collection of directories **118**. A directory **118** is a collection of named files **120**—both data files **120** and other directory files **118**. A file **120** is a named data item which is either a data file (which may be simple or compound) or a directory file **118**. A simple file **120** consists of a single data segment **122**. A compound file **120** consists of a sequence of data segments **122**. A data segment **122** is a fixed sequence of bytes. An important property of any data segment is its size, the number of bytes in the sequence.

A single processor **102** may access one or more file systems **116**, and a single storage device **104** may contain one or more file systems **116**, or portions of a file system **116**. For instance, a file system **116** may span several storage devices **104**.

In order to implement controls in a file system, file system **116** may be divided into distinct regions, where each region is a unit of management and control. A region consists of a given directory **118** and is identified by the pathname (user defined) of the directory.

In the following, the term "location", with respect to a data processing system **100**, refers to any of a particular processor **102** in the system, a memory of a particular processor, a storage device, a removable storage medium (such as a floppy disk or compact disk), or any other physical location in the system. The term "local" with respect to a particular processor **102** refers to the memory and storage devices of that particular processor.

In the following, the terms "True Name", "data identity" and "data identifier" refer to the substantially unique data identifier for a particular data item. The term "True File" refers to the actual file, segment, or data item identified by a True Name.

A file system for a data processing system **100** is now described which is intended to work with an existing operating system by augmenting some of the operating system's file management system codes. The embodiment provided relies on the standard file management primitives for actually storing to and retrieving data items from disk, but uses the mechanisms of the present invention to reference and access those data items.

The processes and mechanisms (services) provided in this embodiment are grouped into the following categories: primitive mechanisms, operating system mechanisms, remote mechanisms, background mechanisms, and extended mechanisms.

Primitive mechanisms provide fundamental capabilities used to support other mechanisms. The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

Operating system mechanisms provide typical familiar file system mechanisms, while maintaining the data structures required to offer the mechanisms of the, present invention. Operating system mechanisms are designed to augment existing operating systems, and in this way to make the present invention compatible with, and generally transparent to, existing applications. The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;

4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

Remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation. The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

Background mechanisms are intended to run occasionally and at a low priority. These provide automated management capabilities with respect to the present invention. The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links; and
4. Verify Region; and
5. Groom Source List.

Extended mechanisms run within application programs over the operating system. These mechanisms provide solutions to specific problems and applications. The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at location;
7. Verify True File;
8. Track for accounting purposes; and
9. Track for licensing purposes.

The file system herein described maintains sufficient information to provide a variety of mechanisms not ordinarily offered by an operating system, some of which are listed and described here. Various processing performed by this embodiment of the present invention will now be described in greater detail.

In some embodiments, some files **120** in a data processing system **100** do not have True Names because they have been recently received or created or modified, and thus their True Names have not yet been computed. A file that does not yet have a True Name is called a scratch file. The process of assigning a True Name to a file is referred to as assimilation, and is described later. Note that a scratch file may have a user provided name.

Some of the processing performed by the present invention can take place in a background mode or on a delayed or as-needed basis. This background processing is used to determine information that is not immediately required by the system or which may never be required. As an example, in some cases a scratch file is being changed at a rate greater than the rate at which it is useful to determine its True Name. In these cases, determining the True Name of the file can be postponed or performed in the background.

Data Structures

The following data structures, stored in memory **110** of one of more processors **102** are used to implement the mechanisms described herein. The data structures can be local to each processor **102** of the system **100**, or they can reside on only some of the processors **102**.

The data structures described are assumed to reside on individual peer processors **102** in the data processing system **100**. However, they can also be shared by placing them on a remote, shared file server (for instance, in a local area network of machines). In order to accommodate sharing data structures, it is necessary that the processors accessing the shared database use the appropriate locking techniques to ensure that changes to the shared database do not interfere with one another but are appropriately serialized. These locking techniques are well understood by ordinarily skilled programmers of distributed applications.

It is sometimes desirable to allow some regions to be local to a particular processor **102** and other regions to be shared among processors **102**. (Recall that a region is a unit of file system management and control consisting of a given directory identified by the pathname of the directory.) In the case of local and shared regions, there would be both local and shared versions of each data structure. Simple changes to the processes described below must be made to ensure that appropriate data structures are selected for a given operation.

The local directory extensions (LDE) table **124** is a data structure which provides information about files **120** and directories **118** in the data processing system **100**. The local directory extensions table **124** is indexed by a pathname or contextual name (that is, a user provided name) of a file and includes the True Name for most files. The information in local directory extension table **124** is in addition to that provided by the native file system of the operating system.

The True File registry (TFR) **126** is a data store for listing actual data items which have True Names, both files **120** and segments **122**. When such data items occur in the True File registry **126** they are known as True Files. True Files are identified in True File registry **126** by their True Names or identities. The table True File registry **126** also stores location, dependency, and migration information about True Files.

The region table (RT) **128** defines areas in the network storage which are to be managed separately. Region table **128** defines the rules for access to and migration of files **120** among various regions with the local file system **116** and remote peer file systems.

The source table (ST) **130** is a list of the sources of True Files other than the current True File registry **126**. The source table **130** includes removable volumes and remote processors.

The audit file (AF) **132** is a list of records indicating changes to be made in local or remote files, these changes to be processed in background.

The accounting log (AL) **134** is a log of file transactions used to create accounting information in a manner which preserves the identity of files being tracked independent of their name or location.

The license table (LT) **136** is a table identifying files, which may only be used by licensed users, in a manner independent of their name or location, and the users licensed to use them.

Detailed Descriptions of the Data Structures

The following table summarizes the fields of an local directory extensions table entry, as illustrated by record **138** in FIG. **3**.

Field	Description
Region ID	identifies the region in which this file is contained.
Pathname	the user provided name or contextual name of the file or directory, relative to the region in which it occurs.
True Name	the computed True Name or identity of the file or directory. This True Name is not always up to date, and it is set to a special value when a file is modified and is later recomputed in the background.
Type	indicates whether the file is a data file or a directory.
Scratch File ID	the physical location of the file in the file system, when no True Name has been calculated for the file. As noted above, such a file is called a scratch file.
Time of last access	the last access time to this file. If this file is a directory, this is the last access time to any file in the directory.
Time of last modification	the time of last change of this file. If this file is a directory, this is the last modification time of any file in the directory.
Safe flag	indicates that this file (and, if this file is a directory, all of its subordinate files) have been backed up on some other system, and it is therefore safe to remove them.
Lock flag	indicates whether a file is locked, that is, it is being modified by the local processor or a remote processor. Only one processor may modify a file at a time.
Size	the full size of this directory (including all subordinate files), if all files in it were fully expanded and duplicated. For a file that is not a directory this is the size of the actual True File.
Owner	the identity of the user who owns this file, for accounting and license tracking purposes.

Each record of the True File registry **126** has the fields shown in the True File registry record **140** in FIG. **4**. The True File registry **126** consists of the database described in the table below as well as the actual True Files identified by the True File IDs below.

Field	Description
True Name	computed True Name or identity of the file.
Compressed File ID	compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.
Grooming delete count	tentative count of how many references have been selected for deletion during a grooming operation.
Time of last access	most recent date and time the content of this file was accessed.
Expiration	date and time after which this file may be deleted by this server.
Dependent processors	processor IDs of other processors which contain references to this True File.
Source IDs	source ID(s) of zero or more sources from which this file or data item may be retrieved.
True File ID	identity or disk location of the actual physical representation of the file or file segment. It is sufficient to use a filename in the registration directory of the underlying operating system. The True File ID is absent if the actual file is not currently present at the current location.
Use count	number of other records on this processor which identify this True File.

A region table **128**, specified by a directory pathname, records storage policies which allow files in the file system to be stored, accessed and migrated in different ways. Storage policies are programmed in a configurable way using a set of rules described below.

Each region table record **142** of region table **128** includes the fields described in the following table (with reference to FIG. **5**):

Field	Description
Region ID	internally used identifier for this region.
Region file system	file system on the local processor of which this region is a part.
Region pathname	a pathname relative to the region file system which defines the location of this region. The region consists of all files and directories subordinate to this pathname, except those in a region subordinate to this region.
Mirror processor(s)	zero or more identifiers of processors which are to keep mirror or archival copies of all files in the current region. Multiple mirror processors can be defined to form a mirror group.
Mirror duplication count	number of copies of each file in this region that should be retained in a mirror group.
Region status	specifies whether this region is local to a single processor 102, shared by several processors 102 (if, for instance, it resides on a shared file server), or managed by a remote processor.
Policy	the migration policy to apply to this region. A single region might participate in several policies. The policies are as follows (parameters in brackets are specified as part of the policy): region is a cached version from [processor ID]; region is a member of a mirror set defined by [processor ID]. region is to be archived on [processor ID]. region is to be backed up locally, by placing new copies in [region ID]. region is read only and may not be changed. region is published and expires on [date]. Files in this region should be compressed.

A source table **130** identifies a source location for True Files. The source table **130** is also used to identify client processors making reservations on the current processor. Each source record **144** of the source table **130** includes the fields summarized in the following table, with reference to FIG. **6**:

Field	Description
source ID	internal identifier used to identify a particular source.
source type	type of source location: Removable Storage Volume Local Region Cache Server Mirror Group Server Cooperative Server Publishing Server Client
source rights	includes information about the rights of this processor, such as whether it can ask the local processor to store data items for it.
source availability	measurement of the bandwidth, cost, and reliability of the connection to this source of True Files. The availability is used to select from among several possible sources.
source location	information on how the local processor is to access the source. This may be, for example, the name of a removable storage volume, or the processor ID and region path of a region on a remote processor.

The audit file **132** is a table of events ordered by timestamp, each record **146** in audit file **132** including the fields summarized in the following table (with reference to FIG. **7**):

Field	Description
Original Name	path of the file in question.
Operation	whether the file was created, read, written, copied or deleted.
Type	specifies whether the source is a file or a directory.
Processor ID	ID of the remote processor generating this event (if not local).
Timestamp	time and date file was closed (required only for accessed/modified files).
Pathname	Name of the file (required only for rename).
True Name	computed True Name of the file. This is used by remote systems to mirror changes to the directory and is filled in during background processing.

Each record **148** of the accounting log **134** records an event which may later be used to provide information for billing mechanisms. Each accounting log entry record **148** includes at least the information summarized in the following table, with reference to FIG. **8**:

Field	Description
date of entry	date and time of this log entry.
type of entry	Entry types include create file, delete file, and transmit file.
True Name	True Name of data item in question.
owner	identity of the user responsible for this action.

Each record **150** of the license table **136** records a relationship between a licensable data item and the user licensed to have access to it. Each license table record **150** includes the information summarized in the following table, with reference to FIG. **9**:

Field	Description
True Name	True Name of a data item subject to license validation.
licensee	identity of a user authorized to have access to this object.

Various other data structures are employed on some or all of the processors **102** in the data processing system **100**. Each processor **102** has a global freeze lock (GFL) **152** (FIG. **1**), which is used to prevent synchronization errors when a directory is frozen or copied. Any processor **102** may include a special archive directory (SAD) **154** into which directories may be copied for the purposes of archival. Any processor **102** may include a special media directory (SMD) **156**, into which the directories of removable volumes are stored to form a media inventory. Each processor has a grooming lock **158**, which is set during a grooming operation. During this period the grooming delete count of True File registry entries **140** is active, and no True Files should be deleted until grooming is complete. While grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

Primitive Mechanisms

The first of the mechanisms provided by the present invention, primitive mechanisms, are now described. The mechanisms described here depend on underlying data management mechanisms to create, copy, read, and delete data items in the True File registry **126**, as identified by a True File ID. This support may be provided by an underlying operating system or disk storage manager.

The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link Path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

1. Calculate True Name

A True Name is computed using a function, MD, which reduces a data block B of arbitrary length to a relatively small, fixed size identifier, the True Name of the data block, such that the True Name of the data block is virtually guaranteed to represent the data block B and only data block B.

The function MD must have the following properties:

1. The domain of the function MD is the set of all data items. The range of the function MD is the set of True Names.
2. The function MD must take a data item of arbitrary length and reduce it to an integer value in the range 0 to N−1, where N is the cardinality of the set of True Names. That is, for an arbitrary length data block B, $0 \leqq MD(B) < N$.
3. The results of MD(B) must be evenly and randomly distributed over the range of N, in such a way that simple or regular changes to B are virtually guaranteed to produce a different value of MD(B).
4. It must be computationally difficult to find a different value B' such that MD(B)=MD(B').
5. The function MD(B) must be efficiently computed.

A family of functions with the above properties are the so-called message digest functions, which are used in digital security systems as techniques for authentification of data. These functions (or algorithms) include MD4, MD5, and SHA.

In the presently preferred embodiments, either MD5 or SHA is employed as the basis for the computation of True Names. Whichever of these two message digest functions is employed, that same function must be employed on a system-wide basis.

It is impossible to define a function having a unique output for each possible input when the number of elements in the range of the function is smaller than the number of elements in its domain. However, a crucial observation is that the actual data items that will be encountered in the operation of any system embodying this invention form a very sparse subset of all the possible inputs.

A colliding set of data items is defined as a set wherein, for one or more pairs x and y in the set, MD(x)=MD(y). Since a function conforming to the requirements for MD must evenly and randomly distribute its outputs, it is possible, by making the range of the function large enough, to make the probability arbitrarily small that actual inputs encountered in the operation of an embodiment of this invention will form a colliding set.

To roughly quantify the probability of a collision, assume that there are no more than 2^{30} storage devices in the world,

and that each storage device has an average of at most 2^{20} different data items. Then there are at most 2^{50} data items in the world. If the outputs of MD range between 0 and 2^{128}, it can be demonstrated that the probability of a collision is approximately 1 in 2^{29}. Details on the derivation of these probability values are found, for example, in P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," *Lecture Notes in Computer Science* 434: *Advances in Cryptology—Eurocrypt '89 Proceedings*, Springer-Verlag, pp. 329-354.

Note that for some less preferred embodiments of the present invention, lower probabilities of uniqueness may be acceptable, depending on the types of applications and mechanisms used. In some embodiments it may also be useful to have more than one level of True Names, with some of the True Names having different degrees of uniqueness. If such a scheme is implemented, it is necessary to ensure that less unique True. Names are not propagated in the system.

While the invention is described herein using only the True Name of a data item as the identifier for the data item, other preferred embodiments use tagged, typed, categorized or classified data items and use a combination of both the True Name and the tag, type, category or class of the data item as an identifier. Examples of such categorizations are files, directories, and segments; executable files and data files, and the like. Examples of classes are classes of objects in an object-oriented system. In such a system, a lower degree of True Name uniqueness is acceptable over the entire universe of data items, as long as sufficient uniqueness. is provided per category of data items. This is because the tags provide an additional level of uniqueness.

A mechanism for calculating a True Name given a data item is now described, with reference to FIGS. **10**(*a*) and **10**(*b*).

A simple data item is a data item whose size is less than a particular given size (which must be defined in each particular implementation of the invention). To determine the True Name of a simple data item, with reference to FIG. **10**(*a*), first compute the MD function (described above) on the given simple data item (Step S**212**). Then append to the resulting 128 bits, the byte length modulo 32 of the data item (Step S**214**). The resulting 160-bit value is the True Name of the simple data item.

A compound data item is one whose size is greater than the particular given size of a simple data item. To determine the True Name of an arbitrary (simple or compound) data item, with reference to FIG. **10**(*b*), first determine if the data item is a simple or a compound data item (Step S**216**). If the data item is a simple data item, then compute its True Name in step S**218** (using steps S**212** and S**214** described above), otherwise partition the data item into segments (Step S**220**) and assimilate each segment (Step S**222**) (the primitive mechanism, Assimilate a Data Item, is described below), computing the True Name of the segment. Then create an indirect block consisting of the computed segment True Names (Step S**224**). An indirect block is a data item which consists of the sequence of True Names of the segments. Then, in step S**226**, assimilate the indirect block and compute its True Name. Finally, replace the final thirty-two (32) bits of the resulting True Name (that is, the length of the indirect block) by the length modulo 32 of the compound data item (Step S**228**). The result is the True Name of the compound data item.

Note that the compound data item may be so large that the indirect block of segment True Names is itself a compound data item. In this case the mechanism is invoked recursively until only simple data items are being processed.

Both the use of segments and the attachment of a length to the True Name are not strictly required in a system using the present invention, but are currently considered desirable features in the preferred embodiment.

2. Assimilate Data Item

A mechanism for assimilating a data item (scratch file or segment) into a file system, given the scratch file ID of the data item, is now described with reference to FIG. **11**. The purpose of this mechanism is to add a given data item to the True File registry **126**. If the data item already exists in the True File registry **126**, this will be discovered and used during this process, and the duplicate will be eliminated.

Thereby the system stores at most one copy of any data item or file by content, even when multiple names refer to the same content.

First, determine the True Name of the data item corresponding to the given scratch File ID using the Calculate True Name primitive mechanism (Step S**230**). Next, look for an entry for the True Name in the True File registry **126** (Step S**232**) and determine whether a True Name entry, record **140**, exists in the True File registry **126**. If the entry record includes a corresponding True File ID or compressed File ID (Step S**237**), delete the file with the scratch File ID (Step S**238**). Otherwise store the given True File ID in the entry record (step S**239**).

If it is determined (in step S**232**) that no True Name entry exists in the True File registry **126**, then, in Step S**236**, create a new entry in the True File registry **126** for this True Name. Set the True Name of the entry to the calculated True Name, set the use count for the new entry to one, store the given True File ID in the entry and set the other fields of the entry as appropriate.

Because this procedure may take some time to compute, it is intended to run in background after a file has ceased to change. In the meantime, the file is considered an unassimilated scratch file.

3. New True File

The New True File process is invoked when processing the audit file **132**, some time after a True File has been assimilated (using the Assimilate Data Item primitive mechanism). Given a local directory extensions table entry record **138** in the local directory extensions table **124**, the New True File process can provide the following steps (with reference to FIG. **12**), depending on how the local processor is configured:

First, in step S**238**, examine the local directory extensions table entry record **138** to determine whether the file is locked by a cache server. If the file is locked, then add the ID of the cache server to the dependent processor list of the True File registry table **126**, and then send a message to the cache server to update the cache of the current processor using the Update Cache remote mechanism (Step **242**).

If desired, compress the True File (Step S**246**), and, if desired, mirror the True File using the Mirror True File background mechanism (Step S**248**).

4. Get True Name from Path

The True Name of a file can be used to identify a file by contents, to confirm that a file matches its original contents, or to compare two files. The mechanism to get a True Name given the pathname of a file is now described with reference to FIG. **13**.

First, search the local directory extensions table **124** for the entry record **138** with the given pathname (Step S**250**). If the pathname is not found, this process fails and no True Name corresponding to the given pathname exists. Next, determine whether the local directory extensions table entry record **138** includes a True Name (Step S**252**), and if so, the mechanism's task is complete. Otherwise, determine whether the local directory extensions table entry record **138** identifies a directory (Step S**254**), and if so, freeze the directory (Step S**256**) (the primitive mechanism Freeze Directory is described below).

Otherwise, in step S**258**, assimilate the file (using the Assimilate Data Item primitive mechanism) defined by the

File ID field to generate its True Name and store its True Name in the local directory extensions entry record. Then return the True Name identified by the local directory extensions table **124**.

5. Link Path to True Name

The mechanism to link a path to a True Name provides a way of creating a new directory entry record identifying an existing, assimilated file. This basic process may be used to copy, move, and rename files without a need to copy their contents. The mechanism to link a path to a True Name is now described with reference to FIG. **14**.

First, if desired, confirm that the True Name exists locally by searching for it in the True Name registry or local directory extensions table **135** (Step S**260**). Most uses of this mechanism will require this form of validation. Next, search for the path in the local directory extensions table **135** (Step S**262**). Confirm that the directory containing the file named in the path already exists (Step S**264**). If the named file itself exists, delete the File using the Delete True File operating system mechanism (see below) (Step S**268**).

Then, create an entry record in the local directory extensions with the specified path (Step S**270**) and update the entry record and other data structures as follows: fill in the True Name field of the entry with the specified True Name; increment the use count for the True File registry entry record **140** of the corresponding True Name; note whether the entry is a directory by reading the True File to see if it contains a tag (magic number) indicating that it represents a frozen directory (see also the description of the Freeze Directory primitive mechanism regarding the tag); and compute and set the other fields of the local directory extensions appropriately. For instance, search the region table **128** to identify the region of the path, and set the time of last access and time of last modification to the current time.

6. Realize True File from Location

This mechanism is used to try to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. This mechanism is now described with reference to FIG. **15**.

First, in step S**272**, determine whether the location specified is a processor. If it is determined that the location specified is a processor, then send a Request True File message (using the Request True File remote mechanism) to the remote processor and wait for a response (Step S**274**). If a negative response is received or no response is received after a timeout period, this mechanism fails. If a positive response is received, enter the True File returned in the True File registry **126** (Step S**276**). (If the file received was compressed, enter the True File ID in the compressed File ID field.)

If, on the other hand, it is determined in step S**272** that the location specified is not a processor, then, if necessary, request the user or operator to mount the indicated volume (Step S**278**). Then (Step S**280**) find the indicated file on the given volume and assimilate the file using the Assimilate Data Item primitive mechanism. If the volume does not contain a True File registry **126**, search the media inventory to find the path of the file on the volume. If no such file can be found, this mechanism fails.

At this point, whether or not the location is determined (in step S**272**) to be a processor, if desired, verify the True File (in step S**282**).

7. Locate Remote File

This mechanism allows a processor to locate a file or data item from a remote source of True Files, when a specific source is unknown or unavailable. A client processor system may ask one of several or many sources whether it can supply a data object with a given True Name. The steps to perform this mechanism are as follows (with reference to FIGS. **16**(*a*) and **16**(*b*)).

The client processor **102** uses the source table **145** to select one or more source processors (Step S**284**). If no source processor can be found, the mechanism fails. Next, the client processor **102** broadcasts to the selected sources a request to locate the file with the given True Name using the Locate True File remote mechanism (Step S**286**). The request to locate may be augmented by asking to propagate this request to distant servers. The client processor then waits for one or more servers to respond positively (Step S**288**). After all servers respond negatively, or after a timeout period with no positive response, the mechanism repeats selection (Step S**284**) to attempt to identify alternative sources. If any selected source processor responds, its processor ID is the result of this mechanism. Store the processor ID in the source field of the True File registry entry record **140** of the given True Name (Step S**290**).

If the source location of the True Name is a different processor or medium than the destination (Step S**290***a*), perform the following steps:

(i) Look up the True File registry entry record **140** for the corresponding True Name, and add the source location ID to the list of sources for the True Name (Step S**290***b*); and

(ii) If the source is a publishing system, determine the expiration date on the publishing system for the True Name and add that to the list of sources. If the source is not a publishing system, send a message to reserve the True File on the source processor (Step S**290***c*).

Source selection in step S**284** may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost, and ignoring previously selected processors which did not respond in step S**288**.

8. Make True File Local

This mechanism is used when a True Name is known and a locally accessible copy of the corresponding file or data item is required. This mechanism makes it possible to actually read the data in a True File. The mechanism takes a True Name and returns when there is a local, accessible copy of the True File in the True File registry **126**. This mechanism is described here with reference to the flow chart of FIGS. **17**(*a*) and **17**(*b*).

First, look in the True File registry **126** for a True File entry record **140** for the corresponding True Name (Step S**292**). If no such entry is found this mechanism fails. If there is already a True File ID for the entry (Step S**294**), this mechanism's task is complete. If there is a compressed file ID for the entry (Step S**296**), decompress the file corresponding to the file ID (Step S**298**) and store the decompressed file ID in the entry (Step S**300**). This mechanism is then complete.

If there is no True File ID for the entry (Step S**294**) and there is no compressed file ID for the entry (Step S**296**), then continue searching for the requested file. At this time it may be necessary to notify the user that the system is searching for the requested file.

If there are one or more source IDs, then select an order in which to attempt to realize the source ID (Step S**304**). The order may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost. For each source in the order chosen, realize the True File from the source location (using the Realize True File from Location primitive mechanism), until the True File is realized (Step S**306**). If it is realized, continue with step S**294**. If no known source can realize the True File, use the Locate Remote File primitive mechanism to attempt to find the True File (Step S**308**). If this succeeds, realize the True File from the identified source location and continue with step S**296**.

9. Create Scratch File

A scratch copy of a file is required when a file is being created or is about to be modified. The scratch copy is stored in the file system of the underlying operating system. The

scratch copy is eventually assimilated when the audit file record entry **146** is processed by the Process Audit File Entry primitive mechanism. This Create Scratch File mechanism requires a local directory extensions table entry record **138**. When it succeeds, the local directory extensions table entry record **138** contains the scratch file ID of a scratch file that is not contained in the True File registry **126** and that may be modified. This mechanism is now described with reference to FIGS. **18**(*a*) and **18**(*b*).

First determine whether the scratch file should be a copy of the existing True File (Step S**310**). If so, continue with step S**312**. Otherwise, determine whether the local directory extensions table entry record **138** identifies an existing True File (Step S**316**), and if so, delete the True File using the Delete True File primitive mechanism (Step S**318**). Then create a new, empty scratch file and store its scratch file ID in the local directory extensions table entry record **138** (step S**320**). This mechanism is then complete.

If the local directory extensions table entry record **138** identifies a scratch file ID (Step S**312**), then the entry already has a scratch file, so this mechanism succeeds.

If the local directory extensions table entry record **138** identifies a True File (S**316**), and there is no True File ID for the True File (S**312**), then make the True File local using the Make True File Local primitive mechanism (Step S**322**). If there is still no True File ID, this mechanism fails.

There is now a local True File for this file. If the use count in the corresponding True File registry entry record **140** is one (Step S**326**), save the True File ID in the scratch file ID of the local directory extensions table entry record **138**, and remove the True File registry entry record **140** (Step S**328**). (This step makes the True File into a scratch file.) This mechanism's task is complete.

Otherwise, if the use count in the corresponding True File registry entry record **140** is not one (in step S**326**), copy the file with the given True File ID to a new scratch file, using the Read File OS mechanism and store its file ID in the local directory extensions table entry record **138** (Step S**330**), and reduce the use count for the True File by one. If there is insufficient space to make a copy, this mechanism fails.

10. Freeze Directory

This mechanism freezes a directory in order to calculate its True Name. Since the True Name of a directory is a function of the files within the directory, they must not change during the computation of the True Name of the directory. This mechanism requires the pathname of a directory to freeze. This mechanism is described with reference to FIGS. **19**(*a*) and **19**(*b*).

In step S**332**, add one to the global freeze lock. Then search the local directory extensions table **124** to find each subordinate data file and directory of the given directory, and freeze each subordinate directory found using the Freeze Directory primitive mechanism (Step S**334**). Assimilate each unassimilated data file in the directory using the Assimilate Data Item primitive mechanism (Step S**336**). Then create a data item which begins with a tag or marker (a "magic number") being a unique data item indicating that this data item is a frozen directory (Step S**337**). Then list the file name and True Name for each file in the current directory (Step S**338**). Record any additional information required, such as the type, time of last access and modification, and size (Step S**340**). Next, in step S**342**, using the Assimilate Data Item primitive mechanism, assimilate the data item created in step S**338**. The resulting True Name is the True Name of the frozen directory. Finally, subtract one from the global freeze lock (Step S**344**).

11. Expand Frozen Directory

This mechanism expands a frozen directory in a given location. It requires a given pathname into which to expand the directory, and the True Name of the directory and is described with reference to FIG. **20**.

First, in step S**346**, make the True File with the given True Name local using the Make True File Local primitive mechanism. Then read each directory entry in the local file created in step S**346** (Step S**348**). For each such directory entry, do the following:

Create a full pathname using the given pathname and the file name of the entry (Step S**350**); and

link the created path to the True Name (Step S**352**) using the Link Path to True Name primitive mechanism.

12. Delete True File

This mechanism deletes a reference to a True Name. The underlying True File is not removed from the True File registry **126** unless there are no additional references to the file. With reference to FIG. **21**, this mechanism is performed as follows:

If the global freeze lock is on, wait until the global freeze lock is turned off (Step S**354**). This prevents deleting a True File while a directory which might refer to it is being frozen. Next, find the True File registry entry record **140** given the True Name (Step S**356**). If the reference count field of the True File registry **126** is greater than zero, subtract one from the reference count field (Step S**358**). If it is determined (in step S**360**) that the reference count field of the True File registry entry record **140** is zero, and if there are no dependent systems listed in the True File registry entry record **140**, then perform the following steps:

(i) If the True File is a simple data item, then delete the True File, otherwise,

(ii) (the True File is a compound data item) for each True Name in the data item, recursively delete the True File corresponding to the True Name (Step S**362**).

(iii) Remove the file indicated by the True File ID and compressed file ID from the True File registry **126**, and remove the True File registry entry record **140** (Step S**364**).

13. Process Audit File Entry

This mechanism performs tasks which are required to maintain information in the local directory extensions table **124** and True File registry **126**, but which can be delayed while the processor is busy doing more time-critical tasks. Entries **142** in the audit file **132** should be processed at a background priority as long as there are entries to be processed. With reference to FIG. **22**, the steps for processing an entry are as follows:

Determine the operation in the entry **142** currently being processed (Step S**365**). If the operation indicates that a file was created or written (Step S**366**), then assimilate the file using the Assimilate Data Item primitive mechanism (Step S**368**), use the New True File primitive mechanism to do additional desired processing (such as cache update, compression, and mirroring) (Step S**369**), and record the newly computed True Name for the file in the audit file record entry (Step S**370**).

Otherwise, if the entry being processed indicates that a compound data item or directory was copied (or deleted) (Step S**376**), then for each component True Name in the compound data item or directory, add (or subtract) one to the use count of the True File registry entry record **140** corresponding to the component True Name (Step S**378**).

In all cases, for each parent directory of the given file, update the size, time of last access, and time of last modification, according to the operation in the audit record (Step S**379**).

Note that the audit record is not removed after processing, but is retained for some reasonable period so that it may be used by the Synchronize Directory extended mechanism to allow a disconnected remote processor to update its representation of the local system.

14. Begin Grooming

This mechanism makes it possible to select a set of files for removal and determine the overall amount of space to be recovered. With reference to FIG. **23**, first verify that the global grooming lock is currently unlocked (Step S**382**). Then set the global grooming lock, set the total amount of space freed during grooming to zero and empty the list of files selected for deletion (Step S**384**). For each True File in the True File registry **126**, set the delete count to zero (Step S**386**).

15. Select For Removal

This grooming mechanism tentatively selects a pathname to allow its corresponding True File to be removed. With reference to FIG. **24**, first find the local directory extensions table entry record **138** corresponding to the given pathname (Step S**388**). Then find the True File registry entry record **140** corresponding to the True File name in the local directory extensions table entry record **138** (Step S**390**). Add one to the grooming delete count in the True File registry entry record **140** and add the pathname to a list of files selected for deletion (Step S**392**). If the grooming delete count of the True File registry entry record **140** is equal to the use count of the True File registry entry record **140**, and if the there are no entries in the dependency list of the True File registry entry record **140**, then add the size of the file indicated by the True File ID and or compressed file ID to the total amount of space freed during grooming (Step S**394**).

16. End Grooming

This grooming mechanism ends the grooming phase and removes all files selected for removal. With reference to FIG. **25**, for each file in the list of files selected for deletion, delete the file (Step S**396**) and then unlock the global grooming lock (Step S**398**).

Operating System Mechanisms

The next of the mechanisms provided by the present invention, operating system mechanisms, are now described.

The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

1. Open File

A mechanism to open a file is described with reference to FIGS. **26**(*a*) and **26**(*b*). This mechanism is given as input a pathname and the type of access required for the file (for example, read, write, read/write, create, etc.) and produces either the File ID of the file to be opened or an indication that no file should be opened. The local directory extensions table record **138** and region table record **142** associated with the opened file are associated with the open file for later use in other processing functions which refer to the file, such as read, write, and close.

First, determine whether or not the named file exists locally by examining the local directory extensions table **124** to determine whether there is an entry corresponding to the given pathname (Step S**400**). If it is determined that the file name does not exist locally, then, using the access type, determine whether or not the file is being created by this opening process (Step S**402**). If the file is not being created, prohibit the open (Step S**404**). If the file is being created, create a zero-length scratch file using an entry in local directory extensions table **124** and produce the scratch file ID of this scratch file as the result (Step S**406**).

If, on the other hand, it is determined in step S**400** that the file name does exist locally, then determine the region in which the file is located by searching the region table **128** to find the record **142** with the longest region path which is a prefix of the file pathname (Step S**408**). This record identifies the region of the specified file.

Next, determine using the access type, whether the file is being opened for writing or whether it is being opened only for reading (Step S**410**). If the file is being opened for reading only, then, if the file is a scratch file (Step S**419**), return the scratch File ID of the file (Step S**424**). Otherwise get the True Name from the local directory extensions table **124** and make a local version of the True File associated with the True Name using the Make True File Local primitive mechanism, and then return the True File ID associated with the True Name (Step S**420**).

If the file is not being opened for reading only (Step S**410**), then, if it is determined by inspecting the region table entry record **142** that the file is in a read-only directory (Step S**416**), then prohibit the opening (Step S**422**).

If it is determined by inspecting the region table **128** that the file is in a cached region (Step S**423**), then send a Lock Cache message to the corresponding cache server, and wait for a return message (Step S**418**). If the return message says the file is already locked, prohibit the opening.

If the access type indicates that the file being modified is being rewritten completely (Step S**419**), so that the original data will not be required, then Delete the File using the Delete File OS mechanism (Step S**421**) and perform step S**406**. Otherwise, make a scratch copy of the file (Step S**417**) and produce the scratch file ID of the scratch file as the result (Step S**424**).

2. Close File

This mechanism takes as input the local directory extensions table entry record **138** of an open file and the data maintained for the open file. To close a file, add an entry to the audit file indicating the time and operation (create, read or write). The audit file processing (using the Process Audit File Entry primitive mechanism) will take care of assimilating the file and thereby updating the other records.

3. Read File

To read a file, a program must provide the offset and length of the data to be read, and the location of a buffer into which to copy the data read.

The file to be read from is identified by an open file descriptor which includes a File ID as computed by the Open File operating system mechanism defined above. The File ID may identify either a scratch file or a True File (or True File segment). If the File ID identifies a True File, it may be either a simple or a compound True File. Reading a file is accomplished by the following steps:

In the case where the File ID identifies a scratch file or a simple True File, use the read capabilities of the underlying operating system.

In the case where the File ID identifies a compound file, break the read operation into one or more read operations on component segments as follows:

A. Identify the segment(s) to be read by dividing the specified file offset and length each by the fixed size of a segment (a system dependent parameter), to determine the segment number and number of segments that must be read.

B. For each segment number computed above, do the following:

i. Read the compound True File index block to determine the True Name of the segment to be read.

ii. Use the Realize True File from Location primitive mechanism to make the True File segment available locally. (If that mechanism fails, the Read File mechanism fails).

iii. Determine the File ID of the True File specified by the True Name corresponding to this segment.

iv. Use the Read File mechanism (recursively) to read from this segment into the corresponding location in the specified buffer.

4. Write File

File writing uses the file ID and data management capabilities of the underlying operating system. File access (Make File Local described above) can be deferred until the first read or write.

5. Delete File or Directory

The process of deleting a file, for a given pathname, is described here with reference to FIGS. **27**(*a*) and **27**(*b*).

First, determine the local directory extensions table entry record **138** and region table entry record **142** for the file (Step S**422**). If the file has no local directory extensions table entry record **138** or is locked or is in a read-only region, prohibit the deletion.

Identify the corresponding True File given the True Name of the file being deleted using the True File registry **126** (Step S**424**). If the file has no True Name, (Step S**426**) then delete the scratch copy of the file based on its scratch file ID in the local directory extensions table **124** (Step S**427**), and continue with step S**428**.

If the file has a True Name and the True File's use count is one (Step S**429**), then delete the True File (Step S**430**), and continue with step S**428**.

If the file has a True Name and the True File's use count is greater than one, reduce its use count by one (Step S**431**). Then proceed with step S**428**.

In Step S**428**, delete the local directory extensions table entry record, and add an entry to the audit file **132** indicating the time and the operation performed (delete).

6. Copy File or Directory

A mechanism is provided to copy a file or directory given a source and destination processor and pathname. The Copy File mechanism does not actually copy the data in the file, only the True Name of the file. This mechanism is performed as follows:

(A) Given the source path, get the True Name from the path. If this step fails, the mechanism fails.

(B) Given the True Name and the destination path, link the destination path to the True Name.

(C) If the source and destination processors have different True File registries, find (or, if necessary, create) an entry for the True Name in the True File registry table **126** of the destination processor. Enter into the source ID field of this new entry the source processor identity.

(D) Add an entry to the audit file **132** indicating the time and operation performed (copy).

This mechanism addresses capability of the system to avoid copying data from a source location to a destination location when the destination already has the data. In addition, because of the ability to freeze a directory, this mechanism also addresses capability of the system immediately to make a copy of any collection of files, thereby to support an efficient version control mechanisms for groups of files.

7. Move File or Directory

A mechanism is described which moves (or renames) a file from a source path to a destination path. The move operation, like the copy operation, requires no actual transfer of data, and is performed as follows:

(A) Copy the file from the source path to the destination path.

(B) If the source path is different from the destination path, delete the source path.

8. Get File Status

This mechanism takes a file pathname and provides information about the pathname. First the local directory extensions table entry record **138** corresponding to the pathname given is found. If no such entry exists, then this mechanism fails, otherwise, gather information about the file and its corresponding True File from the local directory extensions table **124**. The information can include any information shown in the data structures, including the size, type, owner, True Name, sources, time of last access, time of last modification, state (local or not, assimilated or not, compressed or not), use count, expiration date, and reservations.

9. Get Files in Directory

This mechanism enumerates the files in a directory. It is used (implicitly) whenever it is necessary to determine whether a file exists (is present) in a directory. For instance, it is implicitly used in the Open File, Delete File, Copy File or Directory, and Move File operating system mechanisms, because the files operated on are referred to by pathnames containing directory names. The mechanism works as follows:

The local directory extensions table **124** is searched for an entry **138** with the given directory pathname. If no such entry is found, or if the entry found is not a directory, then this mechanism fails.

If there is a corresponding True File field in the local directory extensions table record, then it is assumed that the True File represents a frozen directory. The Expand Frozen Directory primitive mechanism is used to expand the existing True File into directory entries in the local directory extensions table.

Finally, the local directory extensions table **124** is again searched, this time to find each directory subordinate to the given directory. The names found are provided as the result.

Remote Mechanisms

The remote mechanisms provided by the present invention are now described. Recall that remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation.

In a presently preferred embodiment, processors communicate with each other using a remote procedure call (RPC) style interface, running over one of any number of communication protocols such as IPX/SPX or TCP/IP. Each peer processor which provides access to its True File registry **126** or file regions, or which depends on another peer processor, provides a number of mechanisms which can be used by its peers.

The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

1. Locate True File

This mechanism allows a remote processor to determine whether the local processor contains a copy of a specific True File. The mechanism begins with a True Name and a flag indicating whether to forward requests for this file to other servers. This mechanism is now described with reference to FIG. **28**.

First determine if the True File is available locally or if there is some indication of where the True File is located (for example, in the Source IDs field). Look up the requested True Name in the True File registry **126** (Step S**432**).

If a True File registry entry record **140** is not found for this True Name (Step S**434**), and the flag indicates that the request is not to be forwarded (Step S**436**), respond negatively (Step S**438**). That is, respond to the effect that the True File is not available.

One the other hand, if a True File registry entry record **140** is not found (Step S**434**), and the flag indicates that the request for this True File is to be forwarded (Step S**436**), then forward a request for this True File to some other processors in the system (Step S**442**). If the source table for the current processor identifies one or more publishing servers which should have a copy of this True File, then forward the request to each of those publishing servers (Step S**436**).

If a True File registry entry record **140** is found for the required True File (Step S**434**), and if the entry includes a True File ID or Compressed File ID (Step S**440**), respond positively (Step S**444**). If the entry includes a True File ID then this provides the identity or disk location of the actual physical representation of the file or file segment required. If the entry include a Compressed File ID, then a compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.

If the True File registry entry record **140** is found (Step S**434**) but does not include a True File ID (the File ID is absent if the actual file is not currently present at the current location) (Step S**440**), and if the True File registry entry record **140** includes one or more source processors, and if the request can be forwarded, then forward the request for this True File to one or more of the source processors (Step S**444**).

2. Reserve True File

This mechanism allows a remote processor to indicate that it depends on the local processor for access to a specific True File. It takes a True Name as input. This mechanism is described here.

(A) Find the True File registry entry record **140** associated with the given True File. If no entry exists, reply negatively.

(B) If the True File registry entry record **140** does not include a True File ID or compressed File ID, and if the True File registry entry record **140** includes no source IDs for removable storage volumes, then this processor does not have access to a copy of the given file. Reply negatively.

(C) Add the ID of the sending processor to the list of dependent processors for the True File registry entry record **140**. Reply positively, with an indication of whether the reserved True File is on line or off line.

3. Request True File

This mechanism allows a remote processor to request a copy of a True File from the local processor. It requires a True Name and responds positively by sending a True File back to the requesting processor. The mechanism operates as follows:

(A) Find the True File registry entry record **140** associated with the given True Name. If there is no such True File registry entry record **140**, reply negatively.

(B) Make the True File local using the Make True File Local primitive mechanism. If this mechanism fails, the Request True File mechanism also fails.

(C) Send the local True File in either it is uncompressed or compressed form to the requesting remote processor. Note that if the True File is a compound file, the components are not sent.

(D) If the remote file is listed in the dependent process list of the True File registry entry record **140**, remove it.

4. Retire True File

This mechanism allows a remote processor to indicate that it no longer plans to maintain a copy of a given True File. An alternate source of the True File can be specified, if, for instance, the True File is being moved from one server to another. It begins with a True Name, a requesting processor ID, and an optional alternate source. This mechanism operates as follows:

(A) Find a True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Find the requesting processor on the source list and, if it is there, remove it.

(C) If an alternate source is provided, add it to the source list for the True File registry entry record **140**.

(D) If the source list of the True File registry entry record **140** has no items in it, use the Locate Remote File primitive mechanism to search for another copy of the file. If it fails, raise a serious error.

5. Cancel Reservation

This mechanism allows a remote processor to indicate that it no longer requires access to a True File stored on the local processor. It begins with a True Name and a requesting processor ID and proceeds as follows:

(A) Find the True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Remove the identity of the requesting processor from the list of dependent processors, if it appears.

(C) If the list of dependent processors becomes zero and the use count is also zero, delete the True File.

6. Acquire True File

This mechanism allows a remote processor to insist that a local processor make a copy of a specified True File. It is used, for example, when a cache client wants to write through a new version of a file. The Acquire True File mechanism begins with a data item and an optional True Name for the data item and proceeds as follows:

(A) Confirm that the requesting processor has the right to require the local processor to acquire data items. If not, send a negative reply.

(B) Make a local copy of the data item transmitted by the remote processor.

(C) Assimilate the data item into the True File registry of the local processor.

(D) If a True Name was provided with the file, the True Name calculation can be avoided, or the mechanism can verify that the file received matches the True Name sent.

(E) Add an entry in the dependent processor list of the true file registry record indicating that the requesting processor depends on this copy of the given True File.

(F) Send a positive reply.

7. Lock Cache

This mechanism allows a remote cache client to lock a local file so that local users or other cache clients cannot change it while the remote processor is using it. The mechanism begins with a pathname and proceeds as follows:

(A) Find the local directory extensions table entry record **138** of the specified pathname. If no such entry exists, reply negatively.

(B) If an local directory extensions table entry record **138** exists and is already locked, reply negatively that the file is already locked.

(C) If an local directory extensions table entry record **138** exists and is not locked, lock the entry. Reply positively.

8. Update Cache

This mechanism allows a remote cache client to unlock a local file and update it with new contents. It begins with a

pathname and a True Name. The file corresponding to the True Name must be accessible from the remote processor. This mechanism operates as follows:

Find the local directory extensions table entry record **138** corresponding to the given pathname. Reply negatively if no such entry exists or if the entry is not locked.

Link the given pathname to the given True Name using the Link Path to True Name primitive mechanism.

Unlock the local directory extensions table entry record **138** and return positively.

9. Check Expiration Date

Return current or new expiration date and possible alternative source to caller.

Background Processes and Mechanisms

The background processes and mechanisms provided by the present invention are now described. Recall that background mechanisms are intended to run occasionally and at a low priority to provide automated management capabilities with respect to the present invention.

The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links;
4. Verify Region; and
5. Groom Source List.

1. Mirror True File

This mechanism is used to ensure that files are available in alternate locations in mirror groups or archived on archival servers. The mechanism depends on application-specific migration/archival criteria (size, time since last access, number of copies required, number of existing alternative sources) which determine under what conditions a file should be moved. The Mirror True File mechanism operates as follows, using the True File specified, perform the following steps:

(A) Count the number of available locations of the True File by inspecting the source list of the True File registry entry record **140** for the True File. This step determines how many copies of the True. File are available in the system.

(B) If the True File meets the specified migration criteria, select a mirror group server to which a copy of the file should be sent. Use the Acquire True File remote mechanism to copy the True File to the selected mirror group server. Add the identity of the selected system to the source list for the True File.

2. Groom Region

This mechanism is used to automatically free up space in a processor by deleting data items that may be available elsewhere. The mechanism depends on application-specific grooming criteria (for instance, a file may be removed if there is an alternate online source for it, it has not been accessed in a given number of days, and it is larger than a given size). This mechanism operates as follows:

Repeat the following steps (i) to (iii) with more aggressive grooming criteria until sufficient space is freed or until all grooming criteria have been exercised. Use grooming information to determine how much space has been freed. Recall that, while grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

(i) Begin Grooming (using the primitive mechanism).

(ii) For each pathname in the specified region, for the True File corresponding to the pathname, if the True File is present, has at least one alternative source, and meets application specific grooming criteria for the region, select the file for removal (using the primitive mechanism).

(iii) End Grooming (using the primitive mechanism).

If the region is used as a cache, no other processors are dependent on True Files to which it refers, and all such True Files are mirrored elsewhere. In this case, True Files can be removed with impunity. For a cache region, the grooming criteria would ordinarily eliminate the least recently accessed True Files first. This is best done by sorting the True Files in the region by the most recent access time before performing step (ii) above. The application specific criteria would thus be to select for removal every True File encountered (beginning with the least recently used) until the required amount of free space is reached.

3. Check for Expired Links

This mechanism is used to determine whether dependencies on published files should be refreshed. The following steps describe the operation of this mechanism:

For each pathname in the specified region, for each True File corresponding to the pathname, perform the following step:

If the True File registry entry record **140** corresponding to the True File contains at least one source which is a publishing server, and if the expiration date on the dependency is past or close, then perform the following steps:

(A) Determine whether the True File registry entry record contains other sources which have not expired.

(B) Check the True Name expiration of the server. If the expiration date has been extended, or an alternate source is suggested, add the source to the True File registry entry record **140**.

(C) If no acceptable alternate source was found in steps (A) or (B) above, make a local copy of the True File.

(D) Remove the expired source.

4. Verify Region

This mechanism can be used to ensure that the data items in the True File registry **126** have not been damaged accidentally or maliciously. The operation of this mechanism is described by the following steps:

(A) Search the local directory extensions table **124** for each pathname in the specified region and then perform the following steps:

(i) Get the True File name corresponding to the pathname;

(ii) If the True File registry entry **140** for the True File does not have a True File ID or compressed file ID, ignore it.

(iii) Use the Verify True File mechanism (see extended mechanisms below) to confirm that the True File specified is correct.

5. Groom Source List

The source list in a True File entry should be groomed sometimes to ensure there are not too many mirror or archive copies. When a file is deleted or when a region definition or its mirror criteria are changed, it may be necessary to inspect the affected True Files to determine whether there are too many mirror copies. This can be done with the following steps:

For each affected True File,

(A) Search the local directory extensions table to find each region that refers to the True File.

(B) Create a set of "required sources", initially empty.

(C) For each region found,

(a) determine the mirroring criteria for that region,

(b) determine which sources for the True File satisfy the mirroring criteria, and

(c) add these sources to the set of required sources.

(D) For each source in the True File registry entry, if the source identifies a remote processor (as opposed to removable media), and if the source is not a publisher, and if the source is not in the set of required sources, then eliminate the source, and use the Cancel Reservation remote mechanism to eliminate the given processor from the list of dependent processors recorded at the remote processor identified by the source.

Extended Mechanisms

The extended mechanisms provided by the present invention are now described. Recall that extended mechanisms run within application programs over the operating system to provide solutions to specific problems and applications.

The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize Directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at Location;
7. Verify True File;
8. Track for Accounting Purposes; and
9. Track for Licensing Purposes.

1. Inventory Existing Directory

This mechanism determines the True Names of files in an existing on-line directory in the underlying operating system. One purpose of this mechanism is to install True Name mechanisms in an existing file system.

An effect of such an installation is to eliminate immediately all duplicate files from the file system being traversed. If several file systems are inventoried in a single True File registry, duplicates across the volumes are also eliminated.

(A) Traverse the underlying file system in the operating system. For each file encountered, excluding directories, perform the following:

(i) Assimilate the file encountered (using the Assimilate File primitive mechanism). This process computes its True Name and moves its data into the True File registry **126**.

(ii) Create a pathname consisting of the path to the volume directory and the relative path of the file on the media. Link this path to the computed True Name using the Link Path to True Name primitive mechanism.

2. Inventory Removable, Read-only Files

A system with access to removable, read-only media volumes (such as WORM disks and CD-ROMs) can create a usable inventory of the files on these disks without having to make online copies. These objects can then be used for archival purposes, directory overlays, or other needs. An operator must request that an inventory be created for such a volume.

This mechanism allows for maintaining inventories of the contents of files and data items on removable media, such as diskettes and CD-ROMs, independent of other properties of the files such as name, location, and date of creation.

The mechanism creates an online inventory of the files on one or more removable volumes, such as a floppy disk or CD-ROM, when the data on the volume is represented as a directory. The inventory service uses a True Name to identify each file, providing a way to locate the data independent of its name, date of creation, or location.

The inventory can be used for archival of data (making it possible to avoid archiving data when that data is already on a separate volume), for grooming (making it possible to delete infrequently accessed files if they can be retrieved from removable volumes), for version control (making it possible to generate a new version of a CD-ROM without having to copy the old version), and for other purposes.

The inventory is made by creating a volume directory in the media inventory in which each file named identifies the data item on the volume being inventoried. Data items are not copied from the removable volume during the inventory process.

An operator must request that an inventory be created for a specific volume. Once created, the volume directory can be frozen or copied like any other directory. Data items from either the physical volume or the volume directory can be accessed using the Open File operating system mechanism which will cause them to be read from the physical volume using the Realize True File from Location primitive mechanism.

To create an inventory the following steps are taken:

(A) A volume directory in the media inventory is created to correspond to the volume being inventoried. Its contextual name identifies the specific volume.

(B) A source table entry **144** for the volume is created in the source table **130**. This entry **144** identifies the physical source volume and the volume directory created in step (A).

(C) The filesystem on the volume is traversed. For each file encountered, excluding directories, the following steps are taken:

(i) The True Name of the file is computed. An entry is created in the True Name registry **124**, including the True Name of the file using the primitive mechanism. The source field of the True Name registry entry **140** identifies the source table entry **144**.

(ii) A pathname is created consisting of the path to the volume directory and the relative path of the file on the media. This path is linked to the computed True Name using Link Path to True Name primitive mechanism.

(D) After all files have been inventoried, the volume directory is frozen. The volume directory serves as a table of contents for the volume. It can be copied using the Copy File or Directory primitive mechanism to create an "overlay" directory which can then be modified, making it possible to edit a virtual copy of a read-only medium.

3. Synchronize Directories

Given two versions of a directory derived from the same starting point, this mechanism creates a new, synchronized version which includes the changes from each. Where a file is changed in both versions, this mechanism provides a user exit for handling the discrepancy. By using True Names, comparisons are instantaneous, and no copies of files are necessary.

This mechanism lets a local processor synchronize a directory to account for changes made at a remote processor. Its purpose is to bring a local copy of a directory up to date after a period of no communication between the local and remote processor. Such a period might occur if the local processor were a mobile processor detached from its server, or if two distant processors were run independently and updated nightly.

An advantage of the described synchronization process is that it does not depend on synchronizing the clocks of the local and remote processors. However, it does require that the local processor track its position in the remote processor's audit file.

This mechanism does not resolve changes made simultaneously to the same file at several sites. If that occurs, an external resolution mechanism such as, for example, operator intervention, is required.

The mechanism takes as input a start time, a local directory pathname, a remote processor name, and a remote directory pathname name, and it operates by the following steps:

(A) Request a copy of the audit file **132** from the remote processor using the Request True File remote mechanism.

(B) For each entry **146** in the audit file **132** after the start time, if the entry indicates a change to a file in the remote directory, perform the following steps:

(i) Compute the pathname of the corresponding file in the local directory. Determine the True Name of the corresponding file.

(ii) If the True Name of the local file is the same as the old True Name in the audit file, or if there is no local file and the audit entry indicates a new file is being created, link the new True Name in the audit file to the local pathname using the Link Path to True Name primitive mechanism.

(iii) Otherwise, note that there is a problem with the synchronization by sending a message to the operator or to a problem resolution program, indicating the local pathname, remote pathname, remote processor, and time of change.

(C) After synchronization is complete, record the time of the final change. This time is to be used as the new start time the next time this directory is synchronized with the same remote processor.

4. Publish Region

The publish region mechanism allows a processor to offer the files in a region to any client processors for a limited period of time.

The purpose of the service is to eliminate any need for client processors to make reservations with the publishing processor. This in turn makes it possible for the publishing processor to service a much larger number of clients.

When a region is published, an expiration date is defined for all files in the region, and is propagated into the publishing system's True File registry entry record **140** for each file.

When a remote file is copied, for instance using the Copy File operating system mechanism, the expiration date is copied into the source field of the client's True File registry entry record **140**. When the source is a publishing system, no dependency need be created.

The client processor must occasionally and in background, check for expired links, to make sure it still has access to these files. This is described in the background mechanism Check for Expired Links.

5. Retire Directory

This mechanism makes it possible to eliminate safely the True Files in a directory, or at least dependencies on them, after ensuring that any client processors depending on those files remove their dependencies. The files in the directory are not actually deleted by this process. The directory can be deleted with the Delete File operating system mechanism.

The mechanism takes the pathname of a given directory, and optionally, the identification of a preferred alternate source processor for clients to use. The mechanism performs the following steps:

(A) Traverse the directory. For each file in the directory, perform the following steps:

(i) Get the True Name of the file from its path and find the True File registry entry **140** associated with the True Name.

(ii) Determine an alternate source for the True File. If the source IDs field of the TFR entry includes the preferred alternate source, that is the alternate source. If it does not, but includes some other source, that is the alternate source. If it contains no alternate sources, there is no alternate source.

(iii) For each dependent processor in the True File registry entry **140**, ask that processor to retire the True File, specifying an alternate source if one was determined, using the remote mechanism.

6. Realize Directory at Location

This mechanism allows the user or operating system to force copies of files from some source location to the True File registry **126** at a given location. The purpose of the mechanism is to ensure that files are accessible in the event the source location becomes inaccessible. This can happen for instance if the source or given location are on mobile computers, or are on removable media, or if the network connection to the source is expected to become unavailable, or if the source is being retired.

This mechanism is provided in the following steps for each file in the given directory, with the exception of subdirectories:

(A) Get the local directory extensions table entry record **138** given the pathname of the file. Get the True Name of the local directory extensions table entry record **138**. This service assimilates the file if it has not already been assimilated.

(B) Realize the corresponding True File at the given location. This service causes it to be copied to the given location from a remote system or removable media.

7. Verify True File

This mechanism is used to verify that the data item in a True File registry **126** is indeed the correct data item given its True Name. Its purpose is to guard against device errors, malicious changes, or other problems.

If an error is found, the system has the ability to "heal" itself by finding another source for the True File with the given name. It may also be desirable to verify that the error has not propagated to other systems, and to log the problem or indicate it to the computer operator. These details are not described here.

To verify a data item that is not in a True File registry **126**, use the Calculate True Name primitive mechanism described above.

The basic mechanism begins with a True Name, and operates in the following steps:

(A) Find the True File registry entry record **140** corresponding to the given True Name.

(B) If there is a True File ID for the True File registry entry record **140** then use it. Otherwise, indicate that no file exists to verify.

(C) Calculate the True Name of the data item given the file ID of the data item.

(D) Confirm that the calculated True Name is equal to the given True Name.

(E) If the True Names are not equal, there is an error in the True File registry **126**. Remove the True File ID from the True File registry entry record **140** and place it somewhere else. Indicate that the True File registry entry record **140** contained an error.

8. Track for Accounting Purposes

This mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another. The mechanism can be used as a basis for a value-based accounting system in which charges are based on the identity of the data stored or transmitted, rather than simply on the number of bits.

This mechanism allows the system to track possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. True names make it possible to identify each file briefly yet uniquely for this purpose.

Tracking the identities of files requires maintaining an accounting log **134** and processing it for accounting or billing purposes. The mechanism operates in the following steps:

(A) Note every time a file is created or deleted, for instance by monitoring audit entries in the Process Audit File Entry primitive mechanism. When such an event is encountered, create an entry **148** in the accounting log **134** that shows the responsible party and the identity of the file created or deleted.

(B) Every time a file is transmitted, for instance when a file is copied with a Request True File remote mechanism or an Acquire True File remote mechanism, create an entry in the accounting log **134** that shows the responsible party, the identity of the file, and the source and destination processors.

(C) Occasionally run an accounting program to process the accounting log **134**, distributing the events to the account

records of each responsible party. The account records can eventually be summarized for billing purposes.

9. Track for Licensing Purposes

This mechanism ensures that licensed files are not used by unauthorized parties. The True Name provides a safe way to identify licensed material. This service allows proof of possession of specific files according to their contents without disclosing their contents.

Enforcing use of valid licenses can be active (for example, by refusing to provide access to a file without authorization) or passive (for example, by creating a report of users who do not have proper authorization).

One possible way to perform license validation is to perform occasional audits of employee systems. The service described herein relies on True Names to support such an audit, as in the following steps:

(A) For each licensed product, record in the license table **136** the True Name of key files in the product (that is, files which are required in order to use the product, and which do not occur in other products) Typically, for a software product, this would include the main executable image and perhaps other major files such as clip-art, scripts, or online help. Also record the identity of each system which is authorized to have a copy of the file.

(B) Occasionally, compare the contents of each user processor against the license table **136**. For each True Name in the license table do the following:

(i) Unless the user processor is authorized to have a copy of the file, confirm that the user processor does not have a copy of the file using the Locate True File mechanism.

(ii) If the user processor is found to have a file that it is not authorized to have, record the user processor and True Name in a license violation table.

The System in Operation

Given the mechanisms described above, the operation of a typical DP system employing these mechanisms is now described in order to demonstrate how the present invention meets its requirements and capabilities.

In operation, data items (for example, files, database records, messages, data segments, data blocks, directories, instances of object classes, and the like) in a DP system employing the present invention are identified by substantially unique identifiers (True Names), the identifiers depending on all of the data in the data items and only on the data in the data items. The primitive mechanisms Calculate True Name and Assimilate Data Item support this property. For any given data item, using the Calculate True Name primitive mechanism, a substantially unique identifier or True Name for that data item can be determined.

Further, in operation of a DP system incorporating the present invention, multiple copies of data items are avoided (unless they are required for some reason such as backups or mirror copies in a fault-tolerant system). Multiple copies of data items are avoided even when multiple names refer to the same data item. The primitive mechanisms Assimilate Data Items and New True File support this property. Using the Assimilate Data Item primitive mechanism, if a data item already exists in the system, as indicated by an entry in the True File registry **126**, this existence will be discovered by this mechanism, and the duplicate data item (the new data item) will be eliminated (or not added). Thus, for example, if a data file is being copied onto a system from a floppy disk, if, based on the True Name of the data file, it is determined that the data file already exists in the system (by the same or some other name), then the duplicate copy will not be installed. If the data item was being installed on the system by some name other than its current name, then, using the Link Path to True Name primitive mechanism, the other (or new) name can be linked to the already existing data item.

In general, the mechanisms of the present invention operate in such a way as to avoid recreating an actual data item at a location when a copy of that data item is already present at that location. In the case of a copy from a floppy disk, the data item (file) may have to be copied (into a scratch file) before it can be determined that it is a duplicate. This is because only one processor is involved. On the other hand, in a multiprocessor environment or DP system, each processor has a record of the True Names of the data items on that processor. When a data item is to be copied to another location (another processor) in the DP system, all that is necessary is to examine the True Name of the data item prior to the copying. If a data item with the same True Name already exists at the destination location (processor), then there is no need to copy the data item. Note that if a data item which already exists locally at a destination location is still copied to the destination location (for example, because the remote system did not have a True Name for the data item or because it arrives as a stream of un-named data), the Assimilate Data Item primitive mechanism will prevent multiple copies of the data item from being created.

Since the True Name of a large data item (a compound data item) is derived from and based on the True Names of components of the data item, copying of an entire data item can be avoided. Since some (or all) of the components of a large data item may already be present at a destination location, only those components which are not present there need be copied. This property derives from the manner in which True Names are determined.

When a file is copied by the Copy File or Directory operating system mechanism, only the True Name of the file is actually replicated.

When a file is opened (using the Open File operating system mechanism), it uses the Make True File Local primitive mechanism (either directly or indirectly through the Create Scratch File primitive mechanism) to create a local copy of the file. The Open File operating system mechanism uses the Make True File Local primitive mechanism, which uses the Realize True File from Location primitive mechanism, which, in turn uses the Request True File remote mechanism.

The Request True File remote mechanism copies only a single data item from one processor to another. If the data item is a compound file, its component segments are not copied, only the indirect block is copied. The segments are copied only when they are read (or otherwise needed).

The Read File operating system mechanism actually reads data. The Read File mechanism is aware of compound files and indirect blocks, and it uses the Realize True File from Location primitive mechanism to make sure that component segments are locally available, and then uses the operating system file mechanisms to read data from the local file.

Thus, when a compound file is copied from a remote system, only its True Name is copied. When it is opened, only its indirect block is copied. When the corresponding file is read, the required component segments are realized and therefore copied.

In operation data items can be accessed by reference to their identities (True Names) independent of their present location. The actual data item or True File corresponding to a given data identifier or True Name may reside anywhere in the system (that is, locally, remotely, offline, etc). If a required True File is present locally, then the data in the file can be accessed. If the data item is not present locally, there are a number of ways in which it can be obtained from wherever it is present. Using the source IDs field of the True File registry table, the location(s) of copies of the True File corresponding to a given True Name can be determined. The Realize True File from Location primitive mechanism tries to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the

True File. If, on the other hand, for some reason it is not known where there is a copy of the True File, or if the processors identified in the source IDs field do not respond with the required True File, the processor requiring the data item can make a general request for the data item using the Request True File remote mechanism from all processors in the system that it can contact.

As a result, the system provides transparent access to any data item by reference to its data identity, and independent of its present location.

In operation, data items in the system can be verified and have their integrity checked. This is from the manner in which True Names are determined. This can be used for security purposes, for instance, to check for viruses and to verify that data retrieved from another location is the desired, and requested data. For example, the system might store the True Names of all executable applications on the system and then periodically redetermine the True Names of each of these applications to ensure that they match the stored True Names. Any change in a True Name potentially signals corruption in the system and can be further investigated. The Verify Region background mechanism and the Verify True File extended mechanisms provide direct support for this mode of operation. The Verify Region mechanism is used to ensure that the data items in the True File registry have not been damaged accidentally or maliciously. The Verify True File mechanism verifies that a data item in a True File registry is indeed the correct data item given its True Name.

Once a processor has determined where (that is, at which other processor or location) a copy of a data item is in the DP system, that processor might need that other processor or location to keep a copy of that data item. For example, a processor might want to delete local copies of data items to make space available locally while knowing that it can rely on retrieving the data from somewhere else when needed. To this end the system allows a processor to Reserve (and cancel the reservation of) True Files at remote locations (using the remote mechanism). In this way the remote locations are put on notice that another location is relying on the presence of the True File at their location.

A DP system employing the present invention can be made into a fault-tolerant system by providing a certain amount of redundancy of data items at multiple locations in the system. Using the Acquire True File and Reserve True File remote mechanisms, a particular processor can implement its own form of fault-tolerance by copying data items to other processors and then reserving them there. However, the system also provides the Mirror True File background mechanism to mirror (make copies) of the True File available elsewhere in the system. Any degree of redundancy (limited by the number of processors or locations in the system) can be implemented. As a result, this invention maintains a desired degree or level of redundancy in a network of processors, to protect against failure of any particular processor by ensuring that multiple copies of data items exist at different locations.

The data structures used to implement various features and mechanisms of this invention store a variety of useful information which can be used, in conjunction with the various mechanisms, to implement storage schemes and policies in a DP system employing the invention. For example, the size, age and location of a data item (or of groups of data items) is provided. This information can be used to decide how the data items should be treated. For example, a processor may implement a policy of deleting local copies of all data items over a certain age if other copies of those data items are present elsewhere in the system. The age (or variations on the age) can be determined using the time of last access or modification in the local directory extensions table, and the presence of other copies of the data item can be determined either from the Safe Flag or the source IDs, or by checking which other processors in the system have copies of the data item and then reserving at least one of those copies.

In operation, the system can keep track of data items regardless of how those items are named by users (or regardless of whether the data items even have names). The system can also track data items that have different names (in different or the same location) as well as different data items that have the same name. Since a data item is identified by the data in the item, without regard for the context of the data, the problems of inconsistent naming in a DP system are overcome.

In operation, the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of these data items. True Names are globally unique identifiers which can be published simply by copying them. For example, a user might create a textual representation of a file on system A with True Name N (for instance as a hexadecimal string), and post it on a computer bulletin board. Another user on system B could create a directory entry F for this True Name N by using the Link Path to True Name primitive mechanism. (Alternatively, an application could be developed which hides the True Name from the users, but provides the same public transfer service.)

When a program on system B attempts to open pathname F linked to True Name N, the Locate Remote File primitive mechanism would be used, and would use the Locate True File remote mechanism to search for True Name N on one or more remote processors, such as system A. If system B has access to system A, it would be able to realize the True File (using the Realize True File from Location primitive mechanism) and use it locally. Alternatively, system B could find True Name N by accessing any publicly available True Name server, if the server could eventually forward the request to system A.

Clients of a local server can indicate that they depend on a given True File (using the Reserve True File remote mechanism) so that the True File is not deleted from the server registry as long as some client requires access to it. (The Retire True File remote mechanism is used to indicate that a client no longer needs a given True File.)

A publishing server, on the other hand, may want to provide access to many clients, and possibly anonymous ones, without incurring the overhead of tracking dependencies for each client. Therefore, a public server can provide expiration dates for True Files in its registry. This allows client systems to safely maintain references to a True File on the public server. The Check For Expired Links background mechanism allows the client of a publishing server to occasionally confirm that its dependencies on the publishing server are safe.

In a variation of this aspect of the invention, a processor that is newly connected (or reconnected after some absence) to the system can obtain a current version of all (or of needed) data in the system by requesting it from a server processor. Any such processor can send a request to update or resynchronize all of its directories (starting at a root directory), simply by using the Synchronize Directories extended mechanism on the needed directories.

Using the accounting log or some other user provided mechanism, a user can prove the existence of certain data items at certain times. By publishing (in a public place) a list of all True Names in the system on a given day (or at some given time), a user can later refer back to that list to show that a particular data item was present in the system at the time that list was published. Such a mechanism is useful in tracking, for example, laboratory notebooks or the like to prove dates of conception of inventions. Such a mechanism also permits proof of possession of a data item at a particular date and time.

The accounting log file can also track the use of specific data items and files by content for accounting purposes. For instance, an information utility company can determine the data identities of data items that are stored and transmitted through its computer systems, and use these identities to provide bills to its customers based on the identities of the data items being transmitted (as defined by the substantially unique identifier). The assignment of prices for storing and transmitting specific True Files would be made by the information utility and/or its data suppliers; this information would be joined periodically with the information in the accounting log file to produce customer statements.

Backing up data items in a DP system employing the present invention can be done based on the True Names of the data items. By tracking backups using True Names, duplication in the backups is prevented. In operation, the system maintains a backup record of data identifiers of data items already backed up, and invokes the Copy File or Directory operating system mechanism to copy only those data items whose data identifiers are not recorded in the backup record. Once a data item has been backed up, it can be restored by retrieving it from its backup location, based on the identifier of the data item. Using the backup record produced by the backup to identify the data item, the data item can be obtained using, for example, the Make True File Local primitive mechanism.

In operation, the system can be used to cache data items from a server, so that only the most recently accessed data items need be retained. To operate in this way, a cache client is configured to have a local registry (its cache) with a remote Local Directory Extensions table (from the cache server). Whenever a file is opened (or read), the Local Directory Extensions table is used to identify the True Name, and the Make True File Local primitive mechanism inspects the local registry. When the local registry already has a copy, the file is already cached. Otherwise, the Locate True File remote mechanism is used to get a copy of the file. This mechanism consults the cache server and uses the Request True File remote mechanism to make a local copy, effectively loading the cache.

The Groom Cache background mechanism flushes the cache, removing the least-recently-used files from the cache client's True File registry. While a file is being modified on a cache client, the Lock Cache and Update Cache remote mechanisms prevent other clients from trying to modify the same file.

In operation, when the system is being used to cache data items, the problems of maintaining cache consistency are avoided.

To access a cache and to fill it from its server, a key is required to identify the data item desired. Ordinarily, the key is a name or address (in this case, it would be the pathname of a file). If the data associated with such a key is changed, the client's cache becomes inconsistent; when the cache client refers to that name, it will retrieve the wrong data. In order to maintain cache consistency it is necessary to notify every client immediately whenever a change occurs on the server.

By using an embodiment of the present invention, the cache key uniquely identifies the data it represents. When the data associated with a name changes, the key itself changes. Thus, when a cache client wishes to access the modified data associated with a given file name, it will use a new key (the True Name of the new file) rather than the key to the old file contents in its cache. The client will always request the correct data, and the old data in its cache will be eventually aged and flushed by the Groom Cache background mechanism.

Because it is not necessary to immediately notify clients when changes on the cache server occur, the present invention makes it possible for a single server to support a much larger number of clients than is otherwise possible.

In operation, the system automatically archives data items as they are created or modified. After a file is created or modified, the Close File operating system mechanism creates an audit file record, which is eventually processed by the Process Audit File Entry primitive mechanism. This mechanism uses the New True File primitive mechanism for any file which is newly created, which in turn uses the Mirror True File background mechanism if the True File is in a mirrored or archived region. This mechanism causes one or more copies of the new file to be made on remote processors.

In operation, the system can efficiently record and preserve any collection of data items. The Freeze Directory primitive mechanism creates a True File which identifies all of the files in the directory and its subordinates. Because this True File includes the True Names of its constituents, it represents the exact contents of the directory tree at the time it was frozen. The frozen directory can be copied with its components preserved.

The Acquire True File remote mechanism (used in mirroring and archiving) preserves the directory tree structure by ensuring that all of the component segments and True Files in a compound data item are actually copied to a remote system. Of course, no transfer is necessary for data items already in the registry of the remote system.

In operation, the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items.

The Freeze Directory primitive mechanism is used to create a collection of data items. The constituent files and segments referred to by the frozen directory are maintained in the registry, without any need to make copies of the constituents each time the directory is frozen.

Whenever a pathname is traversed, the Get Files in Directory operating system mechanism is used, and when it encounters a frozen directory, it uses the Expand Frozen Directory primitive mechanism.

A frozen directory can be copied from one pathname to another efficiently, merely by copying its True Name. The Copy File operating system mechanism is used to copy a frozen directory.

Thus it is possible to efficiently create copies of different versions of a directory, thereby creating a record of its history (hence a version control system).

In operation, the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM. The inventory is independent of other properties of the data items such as their name, location, and date of creation.

The Inventory Existing Directory extended mechanism provides a way to create True File Registry entries for all of the files in a directory. One use of this inventory is as a way to pre-load a True File registry with backup record information. Those files in the registry (such as previously installed software) which are on the volumes inventoried need not be backed up onto other volumes.

The Inventory Removable, Read-only Files extended mechanism not only determines the True Names for the files on the medium, but also records directory entries for each file in a frozen directory structure. By copying and modifying this directory, it is possible to create an on line patch, or small modification of an existing read-only file. For example, it is possible to create an online representation of a modified CD-ROM, such that the unmodified files are actually on the CD-ROM, and only the modified files are online.

In operation, the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. Using the Track for Accounting Purposes extended

mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another.

True Names in Relational and Object-Oriented Databases

Although the preferred embodiment of this invention has been presented in the context of a file system, the invention of True Names would be equally valuable in a relational or object-oriented database. A relational or object-oriented database system using True Names would have similar benefits to those of the file system employing the invention. For instance, such a database would permit efficient elimination of duplicate records, support a cache for records, simplify the process of maintaining cache consistency, provide location-independent access to records, maintain archives and histories of records, and synchronize with distant or disconnected systems or databases.

The mechanisms described above can be easily modified to serve in such a database environment. The True Name registry would be used as a repository of database records. All references to records would be via the True Name of the record. (The Local Directory Extensions table is an example of a primary index that uses the True Name as the unique identifier of the desired records.)

In such a database, the operations of inserting, updating, and deleting records would be implemented by first assimilating records into the registry, and then updating a primary key index to map the key of the record to its contents by using the True Name as a pointer to the contents.

The mechanisms described in the preferred embodiment, or similar mechanisms, would be employed in such a system. These mechanisms could include, for example, the mechanisms for calculating true names, assimilating, locating, realizing, deleting, copying, and moving True Files, for mirroring True Files, for maintaining a cache of True Files, for grooming True Files, and other mechanisms based on the use of substantially unique identifiers.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiments, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

We claim:

1. A computer-implemented method in a system which includes a network of computers, the method implemented at least in part by hardware comprising at least one processor, the method comprising the steps:

(a) at a first computer, obtaining a content-based name for a particular data item from a second computer distinct from the first computer, the content-based name being based at least in part on a function of at least some of the data which comprise the contents of the particular data item, wherein the function comprises a message digest function or a hash function, and wherein two identical data items will have the same content-based name; and

(b) by hardware in combination with software, a processor at said first computer ascertaining whether or not the content-based name for the particular data item corresponds to an entry in a database comprising a plurality of identifiers; and

(c) based at least in part on said ascertaining in (b), determining whether or not access to the particular data item is authorized.

2. A method as recited in claim **1** further comprising:

(d) based at least in part on said determining in (c), causing access to the particular data item to be denied when it is determined that access to the particular data item is not authorized.

3. A method as recited in claim **1** wherein the content-based name for the particular data item was included in a request to access the particular data item, the request having been received from a particular requestor, and wherein said step (c) of determining comprises: determining whether or not the particular requestor is authorized.

4. A method as recited in claim **3** further comprising:

if it is determined that the particular requestor is not authorized, causing the particular requestor's request to access the particular data item to be denied.

5. A method as recited in claim **1** wherein said database comprises a table or a list comprising said plurality of identifiers.

6. A method as recited in claim **1** wherein the plurality of identifiers in the database are identifiers of licensed content items, and wherein the identifier of each licensed content item is based at least in part on the function of at least some of the data comprising the licensed content item.

7. A method as recited in claim **1** further comprising:

(d) collecting information regarding the particular data item.

8. A method as recited in claim **7**, wherein the information collected includes at least one of: (a) information about which data items have been stored on a computer; (b) information about the content of the particular data item, (c) information about an owner of the particular data item, (d) information about a type of particular data item, (e) information about a contextual name of the particular data item, (f) information about whether the particular data item was copied; (g) the content-based name of the particular data item; (h) information about an identity of a requestor; (i) a timestamp; (j) information about whether the particular data item was created; and (k) information about whether the particular data item was read.

9. A method as recited in claim **7** wherein at least some of the information collected is maintained for accounting or billing purposes.

10. A method as recited in claim **1** further comprising:

(d) tracking identities of data items.

11. A method as recited in claim **1** wherein the content-based name of the particular data item is based, at least in part, on a function of all of the data which comprise the contents of the particular data item.

12. A method as recited in claim **1** wherein the message digest function or hash function is selected from the functions: MD4, MD5, and SHA.

13. A method as recited in claim **1** wherein the content-based name of the particular data item is a True Name of the particular data item.

14. A method as recited in claim **1** wherein the particular data item is selected from the group comprising: a file, a portion of a file, a page in memory, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, and a portion of a software product.

15. A method as recited in claim **1** wherein at least some computers make up part of a peer-to-peer network of computers.

16. A method as recited in claim **1** further comprising:

(d) authorizing access to the particular data item when it is not determined that the data item is unauthorized.

17. A method as recited in claim **16** wherein the authorized access permits copying of the particular data item to or from at least one of the computers in said network of computers.

18. A method as recited in claim **16** wherein the content-based name of the particular data item is received at the first computer and wherein, if it is not determined that said par-

ticular data item is unauthorized, access to the data item is permitted from at least one of a plurality of computers distinct from the first computer.

19. A method as recited in claim **16** wherein, if it is not determined that access to said particular data item is unauthorized, access to the data item is permitted from more than one of a plurality of computers in the network of computers.

20. A computer-implemented method operable in a system which includes a plurality of computers, the method comprising:

controlling distribution of content from a first computer to at least one other computer, in response to a request obtained by a first device in the system from a second device in the system, the first device comprising hardware including at least one processor, the request including at least a content-dependent name of a particular data item, the content-dependent name being based at least in part on a function of at least some of the data comprising the particular data item, wherein the function comprises a message digest function or a hash function, and wherein two identical data items will have the same content-dependent name,

based at least in part on said content-dependent name of said particular data item, the first device (A) permitting the content to be provided to or accessed by the at least one other computer if it is not determined that the content is unauthorized or unlicensed, otherwise, (B) if it is determined that the content is unauthorized or unlicensed, not permitting the content to be provided to or accessed by the at least one other computer.

21. A computer-implemented method implemented at least in part by hardware comprising one or more processors, the method comprising:

(a) obtaining a list of content-dependent names, one for each of a plurality of data items, wherein, for each particular data item of the plurality of data items, the corresponding content-dependent name for that particular data item is based at least in part on a function of at least some of the contents of the particular data item, wherein the function comprises a message digest function or a hash function, and wherein two identical data items have the same content-dependent name on the list of content-dependent names;

(b) receiving at a first location, and from a second location distinct from said first location, a content-dependent identifier corresponding to a particular data item, said content-dependent identifier being based at least in part on at least some of the contents of the particular data item;

(c) at said first location, by a processor, in combination with software, determining, based at least in part on said content-dependent identifier for said particular data item, and using said list of content-dependent names, whether a requestor may access the particular data item; and

(d) based on said determining in (c), if it is determined that the requestor may not access the particular data item, causing access to the particular data item to be denied.

22. A method as recited in claim **21** wherein the list of content-dependent names comprises a list of True Names of authorized data items and wherein the content-dependent identifier of the particular data item is a True Name of the particular data item.

23. A method as recited in claim **21** wherein at least some of said computers make up part of a peer-to-peer network of computers.

24. A computer-implemented method implemented at least in part by hardware comprising one or more processors, the method comprising:

(a) using a processor, receiving at a first computer from a second computer, a request regarding a particular data item, said request including at least a content-dependent name for the particular data item, the content-dependent name being based, at least in part, on at least a function of the data in the particular data item, wherein the data used by the function to determine the content-dependent name comprises at least some of the contents of the particular data item, wherein the function that was used comprises a message digest function or a hash function, and wherein two identical data items will have the same content-dependent name; and

(b) in response to said request:

(i) causing the content-dependent name of the particular data item to be compared to a plurality of values;

(ii) hardware in combination with software determining whether or not access to the particular data item is unauthorized based on whether the content-dependent name of the particular data item corresponds to at least one of said plurality of values, and

(iii) based on said determining in step (ii), not allowing the particular data item to be provided to or accessed by the second computer if it is determined that access to the particular data item is not authorized.

25. A method as recited in claim **24** further comprising:

in response to said request:

(iv) allowing the particular data item to be provided to or accessed by the second computer if it is not determined that access to the particular data item is unauthorized.

26. The method of claim **1** wherein the content-based name of the particular data item is based on a function of only the data which comprise the contents of the particular data item.

27. The method of claim **20** wherein the content-dependent name of the particular data item is based on a function of only the data comprising the particular data item.

28. The method of claim **21** wherein, for each particular data item of the plurality of data items, the corresponding content-dependent name for that particular data item was determined as a function of only the contents of that particular data item.

29. The method of claim **24** wherein the data used by the function to determine the content-dependent name of the particular data item comprises only the contents of the particular data item.

30. The method of claim **20** wherein the content-dependent name of the particular data item is based on a function of all of the data comprising the particular data item.

31. The method of claim **21** wherein, for each particular data item of the plurality of data items, the corresponding content-dependent name for that particular data item was based on a function of all of the contents of that particular data item.

32. The method of claim **24** wherein the data used by the function to determine the content-dependent name of the particular data item comprises of all of the contents of the particular data item.

33. The method of claim **1** wherein the hardware comprises a processor and memory.

34. The method of claim **20** wherein the hardware comprises a processor and memory.

35. The method of claim **21** wherein the hardware comprises a processor and memory.

36. The method of claim **24** wherein the hardware comprises a processor and memory.

37. The method of claim **1** wherein the hardware comprises a computer.

38. The method of claim 20 wherein the hardware comprises a computer.

39. The method of claim 21 wherein the hardware comprises a computer.

40. The method of claim 24 wherein the hardware comprises a computer.

41. The method of claim 1 wherein said content-based name for said particular data item was obtained as part of a request to allow providing of the particular data item to one or more computers in the network of computers, and wherein the determining in (c) whether or not access to the particular data item is authorized comprises:

using at least the content-based name of the particular data item and information in the database to determine whether or not to allow the particular data item to be provided to computers in the network of computers.

42. The method of claim 41 further comprising:

(d) based at least in part on said determining in (c), causing permission to provide the particular data item to computers to be denied when it is determined that access to the particular data item is not authorized.

43. The method as in claim 1 wherein:

the function comprises a message digest function or a hash function applied to at least some of the data in the particular data item to obtain a digital fingerprint of the particular data item.

44. The method of claim 1 wherein the content-based name for the particular data item comprises a digital fingerprint of the particular data item, and wherein each of the plurality of identifiers in the database comprises a digital fingerprint of the corresponding data item.

45. The method of claim 1 wherein, when the particular data item comprises a file or a portion of a file comprising an audio signal, the content-based name of the particular data item is a function of at least some of the data comprising the audio signal.

46. The method as in claim 1 wherein, when the particular data item comprises a file or a portion of a file comprising a video signal, the content-based name of the particular data item is a function of at least some of the data comprising the video signal.

47. A method as recited in claim 1 wherein, when it is not determined in step (c) that access to said particular data item is unauthorized, providing of the particular data item is permitted to or from one or more computers in the network of computers.

48. A method as recited in claim 16 wherein the authorizing access to the particular data item authorizes providing the particular data item to or from one or more computers in the network of computers.

49. The method of claim 21 wherein determining in (c) whether the requestor may access the particular data item comprises:

using at least the content-dependent identifier of the particular data item and the list of content-dependent names to determine whether or not the requestor is authorized to allow copying of the particular data item.

50. The method of claim 1 wherein determining whether or not access to the particular data item is authorized comprises determining whether or not access to the particular data item is not unauthorized.

51. The method of claim 1 wherein the content-based name for the particular data item corresponds to an entry in the database when the content-based name for the particular data item matches one of the plurality of identifiers in the database.

52. The method of claim 1 wherein said database maps each identifier of said plurality of identifiers in said database to information relating to a corresponding data item.

53. A method as recited in claim 1 further comprising:

(d) permitting at least one copy of the particular data item to be distributed to or from at least one computer in the network when it is determined that the particular data item is not unauthorized.

54. The method of claim 1 further comprising:

(d) based at least in part on said determining in (c), causing denial of permission for at least one copy of the particular file to be distributed across the plurality of computers in a network when it is determined that the file is not authorized.

55. The method of claim 1 further comprising the step:

(i) by hardware in combination with software, determining said content-based name for said particular data item.

56. The method of claim 55 further comprising:

obtaining a copy of the particular data item, and wherein the step (i) determines the content-based name using the copy of the particular data item.

57. The method of claim 1 further comprising:

obtaining the database.

58. The method of claim 1 wherein the content-based name for said particular data item is based, at least in part, on a size or length of the particular data item.

59. The method of claim 51 wherein the content-based name for the particular data item corresponds to an entry in the database when the content-based name for the particular data item exactly matches one of the plurality of identifiers in the database.

60. The method of claim 1 wherein the database comprises a table or a list identifying data items which may only be used by licensed users.

61. The method of claim 1 wherein the database comprises a table identifying license information associated with data items, and wherein the table is indexed at least by the identifiers of data items.

62. The method of claim 1 wherein, wherein the database comprises a table containing a plurality of records, and wherein a record of the table records a relationship between a specific data item and users licensed to have access to that specific data item.

63. The method of claim 1, wherein the plurality of identifiers in the database are identifiers of licensed content items.

64. The method of claim 5 wherein the table or list identifies conditions under which certain data items may be used.

65. A method as recited in claim 1 wherein access to said data item comprises one or more of: (a) copying the data item to or from at least one of a plurality of computers; (b) providing the data item to at least one of a plurality of computers; (c) reading the data item; (d) copying the data item; (e) distributing the data item; (f) modifying the data item; (g) storing the data item; (h) opening the data item; (i) publishing the data item; (j) writing the data item; (k) moving the data item; and (1) deleting the data item.

66. A system operable in a network of computers, the system comprising hardware including at least one processor to:

(a) obtain at a first computer, from a second computer distinct from said first computer, a content-based name for a particular data item, the content-based name being based at least in part on a function of at least some of the data which comprise the contents of the particular data item, wherein the function comprises a message digest function or a hash function, and wherein two identical data items will have the same content-based name; and to

(b) ascertain whether or not the content-based name for the particular data item corresponds to an entry in a database comprising a plurality of identifiers; and to

(c) determine, based at least in part on whether or not the particular data item corresponds to an entry in a data-

base, whether or not access to the data item is unauthorized at or by one or more computers distinct from the first computer.

67. A system operable in a network of computers, the system comprising hardware including at least one processor to:

control distribution of content from a first computer in said network, at a first computer, in response to a request from a second computer distinct from the first computer, the request including at least a content-dependent name of a particular data item, the content-dependent name being based at least in part on a function of at least some of the data comprising the particular data item, wherein the function comprises a message digest function or a hash function, and wherein two identical data items will have the same content-dependent name, and to

based at least in part on said content-dependent name of said particular data item, selectively permit the content to be provided to or from at least one other computer if providing of the content is not determined to unauthorized or unlicensed.

68. A device operable in a network of computers, the device comprising hardware including at least one processor, and software, in combination with said hardware:

(a) to obtain a list of content-dependent names, one content-dependent name for each of a plurality of data items, wherein, for each of the plurality of data items, the corresponding content-dependent name for that data item is based at least in part on a function of at least some of the contents of that data item, wherein the function is a message digest function or a hash function, and wherein two identical data items have the same content-dependent name;

(b) to receive at a first location, from a second location distinct from said first location, an identifier for a particular data item;

(c) to determine, based at least in part on said identifier for said particular data item, and using said list of content-dependent names, whether a requestor may access the particular data item; and

(d) based at least in part on said determining, if it is determined that requestor may not access the particular data item, to cause access to the particular data item to be denied.

69. A system operable in a network of computers, the system comprising hardware including at least a processor, and software, in combination with said hardware:

(a) to receive at a first computer, from a second computer, a request regarding a data item, said request including at least a content-dependent name for the data item, the content-dependent name being based at least in part on a function of the data in the data item, wherein the data used by the function to determine the content-dependent name comprises at least some of the contents of the data item, wherein the function that was used is a message digest function or a hash function, and wherein two identical data items will have the same content-dependent name; and

(b) in response to said request:

(i) to cause the content-dependent name of the data item to be compared to a plurality of values; and

(ii) to determine if access to the data item is authorized or unauthorized based on whether or not the content-dependent name corresponds to at least one of said plurality of values, and

(iii) based on whether or not it is determined that access to the data item is authorized or unauthorized, to allow the data item to be provided to or accessed by the second computer if it is not determined that access to the data item is unauthorized.

70. A computer-implemented method operable in a system which includes a network of computers, the system implemented at least in part by hardware including at least one processor, the method comprising the steps of:

in response to a request at a first computer, from another computer, said request comprising at least a content-based identifier for a particular data item, the content-based identifier for the particular data item being based at least in part on a given function of at least some data which comprise the contents of the particular data item, wherein the given function comprises a message digest or a hash function, and wherein two identical data items will have the same content-based identifier:

(A) hardware in combination with software, determining whether the content-based identifier for the particular data item corresponds to an entry in a database comprising a plurality of content-based identifiers; and

(B) based at least in part on said determining in step (A), selectively permitting the particular data item to be accessed at or by one or more computers in the network of computers, said one or more computers being distinct from said first computer.

71. A computer-implemented method implemented at least in part by hardware comprising at least one processor and software, in combination with said hardware, the method comprising the steps:

at a first location, by a first computer,

(A) for a particular data item, said particular data item comprising a plurality of segments, for at least some of said plurality of segments, obtaining a corresponding content-dependent segment identifier from another computer at another location, each said corresponding content-dependent segment identifier being based, at least in part, on a given function of at least some of the data comprising the corresponding segment, wherein said given function for content-dependent segment identifier of at least one of the segments comprises at least a message digest function or hash function, and wherein two identical segments will have the same content-dependent segment identifier; and

(B) hardware in combination with software, ascertaining whether or not at least some of said content-dependent segment identifiers have corresponding entries in a database comprising a plurality of content-dependent identifiers; and

(C) based at least in part on said ascertaining in (B), selectively permitting access to the particular data item at one or more locations distinct from the first location,

wherein, when the particular data item comprises a file or a portion of a file comprising an audio signal, the content-dependent segment identifier of at least one of the plurality of segments for particular data item is a function of at least some of the data comprising the audio signal; and

wherein, when the particular data item comprises a file or a portion of a file comprising an video signal, the content-dependent segment identifier of at least one of the plurality of segments for particular data item is a function of at least some of the data comprising the video signal, and

wherein said selectively permitting access to said particular data item in step (C) comprises one or more of: (a) selectively permitting copying of the data item to or from at least one of a plurality of computers; (b) selectively permitting providing of the data item to at least one of a plurality of computers; (c) selectively permitting reading of the data item; (d) selectively permitting copying of the data item; (e) selectively permitting distribution of the data item; (f) selectively permitting modification of the data item; (g) selectively permitting storage of the data item; (h) selectively permitting open-

ing of the data item; (i) selectively permitting publishing of the data item; (j) selectively permitting writing the data item; (k) selectively permitting moving the data item; and (l) selectively permitting deleting the data item.

72. The method of claim **1** wherein step (c) comprises:

determining whether or not access to the particular data item is authorized at or by one or more computers distinct from the first computer.

73. The method of claim **20** wherein the first location is the first computer.

74. The method of claim **20** wherein the second location is the second computer.

75. The method of claim **21** wherein the step (c) of determining takes place at said first location.

76. The method of claim **21** wherein the first location receives the content-dependent identifier from the requestor.

77. The system as in claim **67** wherein the at least one other computer is distinct from the first computer.

78. The system as in claim **68** wherein the identifier is received a the first location from the requestor.

79. The system of claim **68** wherein access to the particular data item comprises one or more of: (a) copying the data item to or from at least one of a plurality of computers; (b) providing the data item to at least one of a plurality of computers; (c) reading the data item; (d) copying the data item; (e) distributing the data item; (f) modifying the data item; (g) storing the data item; (h) opening the data item; (i) publishing the data item; (j) writing the data item; (k) moving the data item; and (l) deleting the data item.

80. The system of claim **69** wherein

based on whether or not it is determined that access to the data item is authorized or unauthorized, the system allows the data item to be provided to the second computer from the first computer if it is not determined that access to the data item is unauthorized.

81. A device operable in a network of computers, the device comprising hardware including at least one processor and memory, to:

(a) receive, at said device, from another device in the network, a content-based identifier for a particular sequence of bits, the content-based identifier being based at least in part on a function of at least some of the particular sequence of bits, wherein the function comprises a message digest function or a hash function, and wherein two identical sequences of bits will have the same content-based identifier; and to

(b) compare the content-based identifier of the particular sequence of bits to a plurality of values; and to

(c) selectively allow said particular sequence of bits to be provided to or accessed by other devices depending on whether or not said content-dependent identifier corresponds to one of the plurality of values.

82. The device of claim **81** wherein the particular sequence of bits represent data selected from the group comprising: a file, a portion of a file, a page in memory, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, and a portion of a software product.

83. The method of claim **1** wherein the content-based name for the particular data item comprises a request for the particular data item.

84. The method of claim **24** wherein the request regarding the particular data item is a request for the particular data item.

85. The system as in claim **69** wherein the request regarding the data item is a request for the data item.

86. A device operable in a network of computers, the device comprising hardware, including at least one processor and memory, to:

(a) receive at said device, from another device in the network, a digital identifier for a particular sequence of bits, the digital identifier being based, at least in part, on a given function of at least some of the bits in the particular sequence of bits, wherein the given function comprises a message digest function or a hash function, and wherein two identical sequences of bits will have the same digital identifier; and

(b) selectively allow the particular sequence of bits to be provided to or accessed by other devices in the system, based at least in part on whether or not the digital identifier for the particular sequence of bits corresponds to a value in a plurality of values, each of the plurality of values being based, at least in part, on the given function of at least some of the bits in a corresponding sequence of bits.

87. A device as in claim **86** wherein the device is a computer.

* * * * *

US007945544B2

(12) **United States Patent**
Farber et al.

(10) **Patent No.: US 7,945,544 B2**
(45) **Date of Patent: May 17, 2011**

(54) **SIMILARITY-BASED ACCESS CONTROL OF DATA IN A DATA PROCESSING SYSTEM**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignees: **Kinetech, Inc.**, Studio City, CA (US); **Level 3 Communications, LLC**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 624 days.

(21) Appl. No.: **11/980,688**

(22) Filed: **Oct. 31, 2007**

(65) **Prior Publication Data**

US 2008/0065635 A1 Mar. 13, 2008

Related U.S. Application Data

(60) Continuation of application No. 11/724,232, filed on Mar. 15, 2007, which is a continuation of application No. 11/017,650, filed on Dec. 22, 2004, which is a continuation of application No. 09/987,723, filed on Nov. 15, 2001, now Pat. No. 6,928,442, which is a continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned, application No. 11/980,688, which is a continuation of application No. 10/742,972, filed on Dec. 23, 2003, which is a division of application No. 09/987,723, which is a continuation of application No. 09/283,160, which is a division of application No. 08/960,079, which is a continuation of application No. 08/425,160.

(51) **Int. Cl.**
G06F 17/00 (2006.01)

(52) **U.S. Cl.** **707/698**; 707/821; 707/822

(58) **Field of Classification Search** 707/609, 707/821–828, 697–698
See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,668,647 A 6/1972 Evangelisti et al.
3,835,260 A 9/1974 Prescher et al.
(Continued)

FOREIGN PATENT DOCUMENTS

EP 0 268 069 A2 5/1988
(Continued)

OTHER PUBLICATIONS

Fowler, et al. "A User-Level Replicated File System," AT&T Bell Laboratories Technical Memorandum 0112670-930414-05, Apr. 1993, and USENIX 1993 Summer Conference Proceedings, Cincinnati, OH, Jun. 1993.

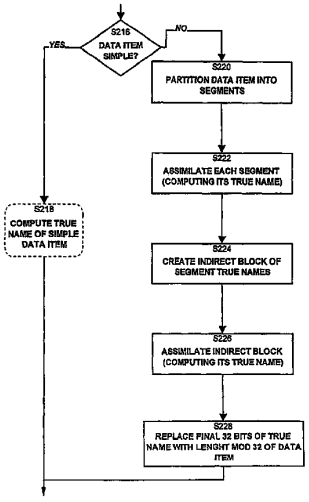
(Continued)

Primary Examiner — Khanh B Pham
(74) *Attorney, Agent, or Firm* — Davidson Berquist Jackson & Gowdey, LLP; Brian Siritzky

(57) **ABSTRACT**

Similarity of data items is determined by analyzing corresponding segments of the data items. A function is applied to each segment of a data item and the output of that function is compared to the output of the same function applied to a corresponding segment of another data item. A function may be applied to the output of the functions. The functions may be hash or message digest functions.

56 Claims, 31 Drawing Sheets



U.S. PATENT DOCUMENTS

4,096,568	A		6/1978	Bennett et al.
4,215,402	A		7/1980	Mitchell et al.
4,221,003	A		9/1980	Chang et al.
4,290,105	A		9/1981	Cichelli et al.
4,376,299	A		3/1983	Rivest
4,405,829	A		9/1983	Rivest et al.
4,412,285	A		10/1983	Neches et al.
4,414,624	A		11/1983	Summer, Jr. et al.
4,441,155	A		4/1984	Fletcher et al.
4,464,713	A		8/1984	Benhase et al.
4,490,782	A		12/1984	Dixon et al.
4,558,413	A		12/1985	Schmidt et al.
4,571,700	A		2/1986	Emry, Jr. et al.
4,577,293	A		3/1986	Matick et al.
4,642,764	A		2/1987	Auslander
4,642,793	A		2/1987	Meaden
4,658,093	A		4/1987	Hellman
4,675,810	A		6/1987	Gruner et al.
4,691,299	A		9/1987	Rivest et al.
4,725,945	A		2/1988	Kronstadt et al.
4,773,039	A		9/1988	Zamora
4,821,184	A		4/1989	Clancy et al.
4,887,235	A		12/1989	Holloway et al.
4,888,681	A		12/1989	Barnes et al.
4,914,571	A		4/1990	Baratz et al.
4,914,586	A		4/1990	Swinehart et al.
4,922,414	A		5/1990	Holloway et al.
4,922,417	A		5/1990	Churm et al.
4,937,863	A		6/1990	Robert et al.
4,949,302	A		8/1990	Arnold et al.
4,953,209	A		8/1990	Ryder, Sr. et al.
4,972,367	A		11/1990	Burke
5,014,192	A		5/1991	Mansfield et al.
5,025,421	A		6/1991	Cho
5,032,979	A		7/1991	Hecht et al.
5,047,918	A		9/1991	Schwartz et al.
5,050,074	A		9/1991	Marca
5,050,212	A		9/1991	Dyson
5,057,837	A		10/1991	Colwell et al.
5,077,658	A		12/1991	Bendert et al.
5,084,815	A		1/1992	Mazzario
5,117,351	A		5/1992	Miller
5,129,081	A		7/1992	Kobayashi et al.
5,129,082	A		7/1992	Tirfing et al.
5,144,667	A		9/1992	Pogue, Jr. et al.
5,163,147	A		11/1992	Orita
5,179,680	A		1/1993	Colwell et al.
5,182,799	A		1/1993	Tamura et al.
5,199,073	A		3/1993	Scott
5,202,982	A		4/1993	Gramlich et al.
5,204,897	A		4/1993	Wyman
5,204,958	A		4/1993	Cheng et al.
5,204,966	A		4/1993	Wittenberg et al.
5,208,858	A		5/1993	Vollert et al.
5,222,134	A		6/1993	Waite et al.
5,230,051	A		7/1993	Quan
5,239,648	A		8/1993	Nukui
5,241,671	A		8/1993	Reed et al.
5,247,620	A		9/1993	Fukuzawa et al.
5,260,999	A		11/1993	Wyman
5,276,869	A		1/1994	Forrest et al.
5,276,901	A		1/1994	Howell et al.
5,287,499	A		2/1994	Nemes
5,287,514	A		2/1994	Gram
5,297,279	A		3/1994	Bannon et al.
5,301,286	A		4/1994	Rajani
5,301,316	A		4/1994	Hamilton et al.
5,317,693	A		5/1994	Cuenod et al.
5,321,841	A		6/1994	East et al.
5,339,403	A		8/1994	Parker
5,341,477	A		8/1994	Pitkin et al.
5,343,527	A		8/1994	Moore
5,347,653	A		9/1994	Flynn et al.
5,351,302	A		9/1994	Leighton et al.
5,357,440	A		10/1994	Talbott et al.
5,357,623	A		10/1994	Megory-Cohen
5,357,630	A		10/1994	Oprescu et al.
5,359,523	A		10/1994	Talbott et al.
5,361,356	A		11/1994	Clark et al.
5,371,897	A		12/1994	Brown et al.
5,375,206	A		12/1994	Hunter et al.
5,384,565	A		1/1995	Cannon
5,394,555	A		2/1995	Hunter et al.
5,403,639	A		4/1995	Belsan et al.
5,404,508	A		4/1995	Konrad et al.
5,438,508	A		8/1995	Wyman
5,442,343	A		8/1995	Cato et al.
5,448,668	A	*	9/1995	Perelson et al. 714/21
5,448,718	A		9/1995	Cohn et al.
5,452,447	A		9/1995	Nelson et al.
5,454,000	A	*	9/1995	Dorfman 714/54
5,454,039	A		9/1995	Coppersmith et al.
5,459,860	A		10/1995	Burnett
5,465,365	A		11/1995	Winterbottom
5,467,471	A		11/1995	Bader
5,475,826	A		12/1995	Fischer
5,479,654	A	*	12/1995	Squibb 707/695
5,491,817	A		2/1996	Gopal et al.
5,499,294	A		3/1996	Friedman
5,504,879	A		4/1996	Eisenberg et al.
5,530,757	A	*	6/1996	Krawczyk 713/188
5,537,585	A		7/1996	Blickenstaff et al.
5,542,087	A		7/1996	Neimat et al.
5,548,724	A		8/1996	Akizawa et al.
5,553,143	A		9/1996	Ross et al.
5,568,181	A		10/1996	Greenwood et al.
5,581,615	A		12/1996	Stern
5,581,758	A		12/1996	Burnett
5,581,764	A		12/1996	Fitzgerald et al.
5,583,995	A		12/1996	Gardner et al.
5,588,147	A		12/1996	Neeman et al.
5,596,744	A		1/1997	Dao et al.
5,600,834	A		2/1997	Howard
5,604,803	A		2/1997	Aziz
5,604,892	A		2/1997	Nuttall et al.
5,630,067	A		5/1997	Kindell et al.
5,632,031	A		5/1997	Velissaropoulos et al.
5,638,443	A		6/1997	Stefik et al.
5,640,564	A		6/1997	Hamilton et al.
5,649,196	A		7/1997	Woodhill et al.
5,677,952	A		10/1997	Blakley, III et al.
5,678,038	A		10/1997	Dockter et al.
5,678,046	A		10/1997	Cahill et al.
5,694,472	A	*	12/1997	Johnson et al. 713/189
5,694,596	A		12/1997	Campbell
5,701,316	A		12/1997	Alferness et al.
5,710,922	A		1/1998	Alley et al.
5,724,425	A		3/1998	Chang et al.
5,724,552	A		3/1998	Taoda
5,742,807	A		4/1998	Masinter
5,745,879	A		4/1998	Wyman
5,757,913	A		5/1998	Bellare et al.
5,757,915	A		5/1998	Aucsmith et al.
5,781,629	A		7/1998	Haber et al.
5,802,291	A		9/1998	Balick et al.
5,809,494	A		9/1998	Nguyen
5,826,049	A		10/1998	Ogata et al.
5,835,087	A		11/1998	Herz et al.
5,864,683	A		1/1999	Boebert et al.
5,907,619	A		5/1999	Davis
5,907,704	A		5/1999	Gudmundson et al.
5,940,504	A		8/1999	Griswold
5,978,791	A		11/1999	Farber et al.
5,991,414	A		11/1999	Garay et al.
6,006,018	A		12/1999	Burnett et al.
6,134,603	A		10/2000	Jones et al.
6,135,646	A		10/2000	Kahn et al.
6,415,280	B1		7/2002	Farber et al.
6,732,180	B1		5/2004	Hale et al.
6,816,872	B1	*	11/2004	Squibb 1/1
6,928,442	B2		8/2005	Farber et al.
2002/0052884	A1		6/2002	Farber et al.
2002/0082999	A1		6/2002	Lee et al.
2003/0078888	A1		4/2003	Lee et al.
2003/0078889	A1		4/2003	Lee et al.
2003/0095660	A1		5/2003	Lee et al.
2004/0139097	A1		7/2004	Farber et al.

2005/0010792	A1	1/2005	Carpentier et al.
2005/0114296	A1	5/2005	Farber et al.
2007/0185848	A1	8/2007	Farber et al.
2008/0065635	A1	3/2008	Farber et al.
2008/0066191	A1	3/2008	Farber et al.
2008/0071855	A1	3/2008	Farber et al.
2008/0082551	A1	3/2008	Farber et al.

FOREIGN PATENT DOCUMENTS

EP	0 315 425		5/1989
EP	0 558 945	A2	9/1993
EP	0 566 967	A2	10/1993
EP	0592045		4/1994
EP	0631 226	A1	12/1994
EP	0 654 920	A2	5/1995
EP	0 658 022	A2	6/1995
GB	2294132	A	4/1996
JP	59058564		4/1984
JP	63-106048		5/1988
JP	63-273961		11/1988
JP	2-127755		5/1990
JP	05162529		6/1993
JP	06187384	A2	7/1994
JP	06348558	A	12/1994
WO	WO 92/20021		11/1992
WO	WO 94/06087		3/1994
WO	WO 94/20913		9/1994
WO	WO 95/01599		1/1995
WO	WO 97/43717		11/1997

OTHER PUBLICATIONS

Greene, D., et al., "Multi-Index Hashing for Information Retrieval", Nov. 20-22, 1994, Proceedings, 35th Annual Symp on Foundations of Computer Science, IEEE, pp. 722-731.

Hirano, et al, "Extendible hashing for concurrent insertions and retrievals," in Proc 4th Euromicro Workshop on Parallel and Distributed Processing, 1996 (PDP '96), Jan. 24, 1996 to Jan. 26, 1996, pp. 235-242, Braga , Portugal.

Preneel et al., "The Cryptographic Hash Function RIPEMD-160", appeared in CryptoBytes RSA Laboratories, vol. 3, No. 2, pp. 9-14, Fall, 1997 (also Bosselaers et al., "The RIPEMD-160 Cryptographic Hash Function", Jan. 1997, Dr. Dobb's Journal, pp. 24-28).

Prusker et al., "The Siphon: Managing Distant Replicated Repositories" Nov. 8-9, 1990, Proc. Management of Replicated Data IEEE.

Reply to Examination Report, Munich, Nov. 18, 2009, in Application No. EP 96 910 762.2 [19 pgs.].

Rich, K. et al, "Hobgoblin: A File and Directory Auditor", Sep. 30-Oct. 3, 1991, Lisa V., San Diego, CA.

USPTO Final Office Action in U.S. Appl. No. 10/742,972, Dec. 22, 2009.

USPTO, Advisory Action, Mar. 23, 2010, in U.S. Appl. No. 11/980,679.

USPTO, Final Office Action in U.S. Reexam Control No. 90/010,260, Jan. 29, 2010.

USPTO, Final Office Action mailed Jan. 12, 2010 in U.S. Appl. No. 11/980,679.

USPTO, Final Office Action mailed Aug. 18, 2009 in U.S. Appl. No. 11/017,650.

USPTO, Final Office Action mailed Sep. 30, 2009 in U.S. Appl. No. 11/724,232.

USPTO, Final Office Action, Mar. 5, 2010 in U.S. Appl. No. 11/980,687.

Cheriton, David R. and Mann, Timothy P., "Decentralizing a global naming service for improved performance and fault tolerance", ACM Transactions on Computer Systems, vol. 7, No. 2, May 1989, pp. 147-183.

Request for Reexamination of U.S. Patent No. 6,928,442: Reexam Control No. 90/010,260, filed on Aug. 29, 2008.

[Proposed] Order Regarding Construction of Terms, filed Mar. 29, 2007 in C.D. Cal. case no. CV 06-5086 SJO (Ex) [9 pgs.].

Analysis of Plaintiffs' Claim Chart for the '280 Patent As Against Defendant Media Sentry, Inc. 11 pages.

Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Media Sentry, Inc. (11916.001.0150.A) pp. 1-48.

Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Overpeer pp. 1-40.

Barbara, D., et al., "Exploiting symmetries for low-cost comparison of file copies," 8th Int'l Conf. on Distributed Computing Systems, Jun. 1988, pp. 471-479, San Jose, CA.

Campbell, M., "The Design of Text Signatures for Text Retrieval Systems," Tech. Report, Sep. 5, 1994, Deakin University, School of Computing & Math., Geelong, Australia.

Chang, W. W. et al., "A signature access method for the Starburst database system," in Proc. 15th Int'l Conf. on Very Large Data Bases (Amsterdam, The Netherlands), pp. 145-153.

Changes to Mar. 23, 2007 Deposition of Robert B. K. Dewar, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [3 pgs + cover letter.].

Communication from EPO in European Application No. 96 910 762.2—1225 dated May 8, 2009 [4 pgs.].

Communication pursuant to Article 96(2) EPC from EPO (Examination Report), Jan. 17, 2007, in Application No. EP 96 910 762.2-1225 [1 pg. with 5 pg. annex].

Complaint for Patent Infringement, Permanent Injunction and Damages, Aug. 8, 2006, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].

Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [10 pgs.].

Declaration of Charles S. Baker in Support of Defendant Lime Wire's Motion to Stay Pending Reexamination of Patent and Request for Extension of Deadlines, Aug. 29, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].

Defendant Lime Wire, LLC's First Amended Answer, Affirmative Defenses and Counterclaims, Oct. 2, 2008, C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].

Defendant Lime Wire, LLC's Second Amended Answer, Affirmative Defenses and Counterclaims, Oct. 27, 2008, from C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].

Defendant Michael Weiss's Answer to Plaintiffs Complaint for Patent Infringement, Permanent Injunction and Damages, Demand for Jury Trial, Sep. 15, 2006, case No. CV 06-5086 SJO (Ex) [10 pgs.].

Defendant Recording Industry Association of America's Amended Notice of Motion and Motion for Partial Summary Judgment on Plaintiffs' Claims for Patent Infringement and Inducing Patent Infringement; Memorandum of Points and Authorities, May 22, 2006, redacted, original confidential, filed under seal, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [19 pgs.].

Defendant Recording Industry Association of America's and Mediasentry, Inc.'s Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, in the Alternative, Based on Patent Misuse and Unclean Hands, May 22, 2006, Redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [21 pgs.].

Defendant Recording Industry Association of America's and Mediasentry, Inc's Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, in the Alternative, Based on Patent Misuse and Unclean Hands, May 8, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [20 pgs.].

Defendant StreamCast Networks Inc.'s Answer to Plaintiff's Complaint for Patent Infringement, Permanent Injunction and Damages; Demand for Jury Trial, Sep. 5, 2006, C.D. Cal. case No. CV 065086 SJO (Ex) [10 pgs.].

Defendants' Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 7, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].

Defendant's Second Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 9, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].

Dewar, Rebuttal Expert Report of Robert B.K. Dewar, in C.D. Cal. case No. CV 04-7456 JFW (CTx), Apr. 10, 2006 [87 pgs].

Faloutsos, C. "Access methods for text," ACM Comput. Surv. 17, 1 (Mar. 1985), 49-74.

Faloutsos, C. et al., "Description and performance analysis of signature file methods for office filing," ACM Trans. Inf. Syst. 5, 3 (Jul. 1987), 237-257.

Faloutsos, C. et al., "Signature files: an access method for documents and its analytical performance evaluation," ACM Trans. Inf. Syst. 2, 4 (Oct. 1984), 267-288.
Federal Information Processing Standards (FIPS) Publication 180-1; Secure Hash Standard, Apr. 17, 1995 [17 pgs.].
Feigenbaum, J. et al., "Cryptographic protection of databases and software," in Distributed Computing and Cryptography: Proc. DIMACS Workshop, Apr. 1991, pp. 161-172, American Mathematical Society, Boston, Mass.
First Amended Answer of Defendant Mediasentry to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [29 pgs.].
First Amended Answer of Defendant RIAA to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [27 pgs.].
First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [10 pgs.].
Harrison, M. C., "Implementation of the substring test by hashing," Commun. ACM 14, 12 (Dec. 1971), 777-779.
IEEE, The Authoritative Dictionary of IEEE Standards Terms, 7th ed., Copyright 2000, pp. 107, 176, 209, 240, 241, 432, 468, 505, 506, 682, 1016, 1113, 1266, and 1267.
Ishikawa, Y., et al., "Evaluation of signature files as set access facilities in OODBs," In Proc. of the 1993 ACM SIGMOD Inter. Conf. on Management of Data (Washington, D.C., U.S., May 1993). P. Buneman & S. Jajodia, Eds. SIGMOD '93. ACM, NY, NY, 247-256.
Joint Claim Construction and Prehearing Statement, N.D. Cal. Rule 4-3, Feb. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [20 pgs.].
Karp, R. M. and Rabin, M. O., "Efficient randomized pattern-matching algorithms," IBM J. Res. Dev. 31, 2 (Mar. 1987), 249-260.
List of Asserted Claims and Infringement Chart for Each Asserted Claim, Jul. 28, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [31 pgs.].
McGregor D. R. And Mariani, J. A. "Fingerprinting—A technique for file identification and maintenance," Software: Practice and Experience, vol. 12, No. 12, Dec. 1982, pp. 1165-1166.
Notice of Interested Parties, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Notice of Motion and Motion of Defendant Lime Wire to Stay Litigation Pending Reexamination of Patent and Request for Extension of Deadlines, Sep. 22, 2008, C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [11 pgs.].
Notice of Related Cases, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Panagopoulos, G., et al., "Bit-sliced signature files for very large text databases on a parallel machine architecture," In Proc. of the 4th Inter. Conf. on Extending Database Technology (EDBT), Cambridge, U.K., Mar. 1994, pp. 379-392 (Proc. LNCS 779 Springer 1994, ISBN 3-540-57818-8) [14 pgs.].
Patent Abstract, "Management System for Plural Versions," Pub. No. 63273961 A, published Nov. 11, 1988, NEC Corp.
Patent Abstracts of Japan, "Data Processor," Appln. No. 05135620, filed Jun. 7, 1993, Toshiba Corp.
Plaintiff Kinetech, Inc.'s Responses to Defendant Mediasentry's First set of Interrogatories, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [14 pgs.].
Plaintiff-Counterclaim Defendant Altnet, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Brilliant Digital Entertainment, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Kinetech, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiffs Altnet, Inc., Brilliant Digital, Inc., and Kinetech, Inc.'s Responses to Defendant Recording Industry Association of America's First Set of Requests for Admissions, Jan. 6, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [26 pgs.].
Plaintiffs' Claim Construction Opening Brief and Exhibits A-D, F, G; May 7, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [112 pgs.].
Plaintiffs' Preliminary Claim Constructions and Extrinsic Evidence, Feb. 6, 2006, in case CV 06-5086 SJO (Ex) [20 pgs.].
Plaintiff's Reply to Defendant Mediasentry's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiff's Reply to Defendant RIAA's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiffs' Reply to Defendants' Claim Construction Brief, filed Apr. 23, 2007 in C.D. Cal. case No. CV 06-5086 ODW (Ex) [15 pgs.].
Reply to Examination Report, Jul. 19, 2007, in Application No. EP 96 910 762.2-1225 [7 pgs.].
Response to Non-Final Office Action filed May 19, 2009 in U.S. Appl. No. 11/017,650 [19 pgs.].
Rivest, R., RFC 1320, "The MD4 Message-Digest Algorithm," The Internet Engineering Task Force (IETF), Apr. 1992.
Sacks-Davis, R., et al., "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst. 12, 4 (Nov. 1987), 655-696.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Proc. Workshop on the Management of Replicated Data, Houston, TX, pp. 15-17, Nov. 8-9, 1990.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Technical Report, TR89-1042, Cornell University, Nov. 1989.
Stipulation and Proposed order to (1) Amend the Complaint, (2) Amend pretrial Schedule, and (3) Withdraw Motion to Stay, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF(PLAx) [6 pgs.].
Streamcast Networks Inc.'s Supplemental Responses to Certain of Plaintiffs' First Set of Interrogatories, Apr. 16, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
StreamCast's Brief Re Claim Construction, Apr. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].
Transcript of Deposition of David Farber, Feb. 16, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [94 pgs.].
Transcript of Deposition of Robert B. K. Dewar, Mar. 23, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
Transcript of Deposition of Ronald Lachman, Feb. 1, 2006, C.D. Cal. case No. CV 04-7456 JFW (CTx) [96 pgs.].
USPTO, Non-Final Office Action mailed May 6, 2009 in U.S. Appl. No. 11/980,679.
USPTO, Non-Final Office action mailed Jun. 15, 2009 in U.S. Appl. No. 11/980,687.
USPTO, Non-Final Office action mailed Jun. 18, 2009 in Reexam No. 90/010,260.
Advances in Cryptology-AUSCRYPT '92—Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia Dec. 13-16, 1992 Proceedings.
Advances in Cryptology-EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23-27, 1993 Proceedings.
Affidavit of Timothy P. Walker in Support of CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.
Akamai and MIT's Memorandum in Support of Their Claim Construction of USPAT 5,978,791, dated Aug. 31, 2001, from Civil Action No. 00-cv-11851RWZ.
Akamai's Answer, Affirmative Defenses and Counterclaims to Amended Complaint, filed Dec. 6, 2002, in Civil Action No. 02-CV-11430RWZ.
Akamai's Brief on Claim Construction, dated Aug. 8, 2003, from Civil Action No. 02-11430 RWZ.
Albert Langer (cmf851@anu.oz.au), http://groups.google.com/groups?selm=1991Aug7.225159.786%40newshost.anu. edu.au &oe=UTF-8&output=gplain, Aug. 7, 1991.
Alexander Dupuy (dupuy@smarts.com), "MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0081.html, Apr. 17, 1994.

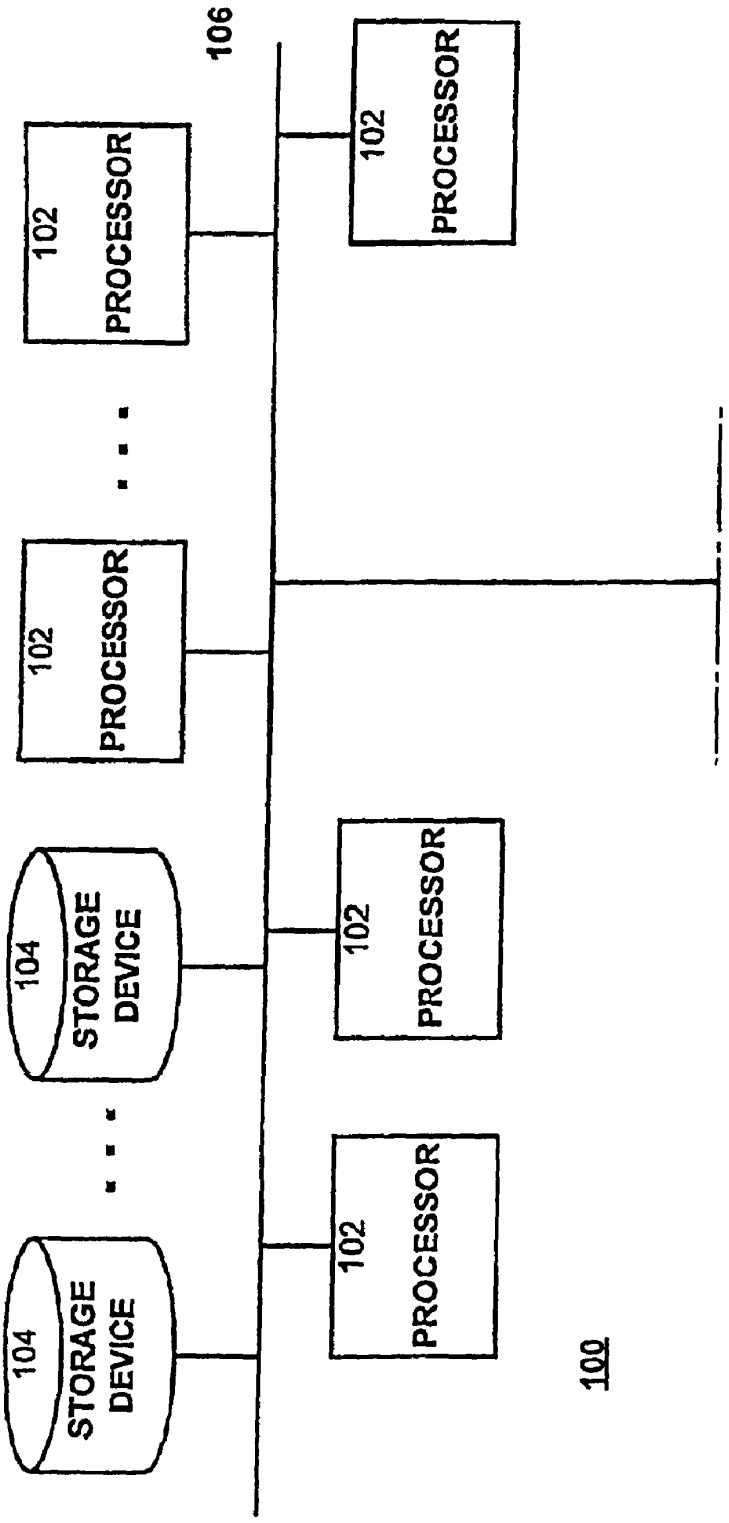
Alexander Dupuy (dupuy@smarts.com), "Re: MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0113.html, Apr. 26, 1994.
Answer of Defendant RIAA to First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).
Berners-Lee, T. et al., "Hypertext Transfer Protocol—HTTP/1.0," May 1996, pp. 1-54.
Berners-Lee, T. et al., "Uniform Resource Locators (URL)," pp. 1-25, Dec. 1994.
Berners-Lee, T., "Universal Resource Identifiers in WWW," Jun. 1994, pp. 1-25.
Bert dem Boer, et al., Collisions for the compression function of MD.sub.5 pp. 292-304, 1994.
Birgit Pfitzman, Sorting Out Signature Schemes, Nov. 1993, 1.sup.st Conf. Computer & Comm. Security '93, p. 74-85.
Birgit Pfitzmann, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93 pp. 74-85.
Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Aug. 4, 1994, pp. 1-27.
Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Mar. 12, 1995, pp. 1-29.
Brisco, T., "DNS Support for Load Balancing," Apr. 1995, pp. 1-7.
Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, 7 pages.
Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, printed from http:/www.netlib.org/utk/papers/lifn/main.html on Mar. 22, 2006, 18 pages.
Carter, J. Lawrence, et al. "Universal Classes of Hash Functions." Journal of Computer and System Sciences, vol. 18, No. 2, Apr. 1979, pp. 143-154.
Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, pp. 156-164, 1993.
Civil Minutes General dated Jan. 25, 2005, from Civil Action No. CV 04-7456-JFW (CTx).
Clifford Lynch (Calur@uccmvsa.bitnet), "ietf url/uri overview draft paper (long)", www.acl.lanl.gov/URI/archive/uri-93q1.messages/0015.html, Mar. 25, 1993.
Complaint for Patent Infringement, Permanent Injunction, and Damages, dated Sep. 8, 2004, from Civil Action No. CV 04-7456 JFW (AJWx).
Cormen, Thomas H., et al. Introduction to Algorithms, The MIT Press, Cambridge, Massachusetts, 1994, pp. 219-243, 991-993.
CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.
CWIS' Reply Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Aug. 15, 2003, from Civil Action No. 02-11430 RWZ.
Danzig, P.B., et al., ""Distributed Indexing: A Scalable Mechanism for Distributed Information Retrieval,"" Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 220-229, Oct. 13-16, 1991.
Davis, James R., "A Server for a Distributed Digital Technical Report Library," Jan. 15, 1994, pp. 1-8.
Declaration of Robert B.K. Dewar in Support of CWIS' Construction of the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-CV-11430RWZ.
Deering, Stephen, et al. "Multicast Routing in Datagram Internetworks and Extended LANs." ACM Transactions on Computer Systems, vol. 8, No. 2, May 1990, pp. 85-110.
Defendant Digital Island's Opening Brief on Claim Construction Issues dated Aug. 17, 2001, from Civil Action No. 00-cv-11851-RWZ.
Defendant Lime Wire, LLC's Answer, Affirmative Defenses and Counterclaims dated Nov. 15, 2007, from Civil Action No. 07-06161 VBF (PLAx).
Defendant Media Sentry, Inc.'s Reply Memorandum of Points and Authorities in Further Support of Its Motion to Dismiss, dated Nov. 15, 2004, from Civil Action No. CV04-7456 JFW (CTx).
Defendant MediaSentry Inc.'s Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW.
Defendant MediaSentry, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaims, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).
Defendant RIAA's Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW (CTx).
Defendants Loudeye Corp.'s and Overpeer, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. 04-7456 JFW (AJWx).
Defendants' Preliminary Invalidity Contentions dated Dec. 14, 2006, from Civil Action No. CV 06-5086 SJO (Ex).
Devine, Robert. "Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm." In Proc. of 4th International Conference on Foundations of Data Organizations and Algorithms, 1993, pp. 101-114.
European Search Report issued Dec. 23, 2004 in correpsonding European Application No. 96910762.2-2201.
Expert Report of Professor Ellis Horowitz, dated Mar. 6, 2006, from Civil Action No. 04-7456 JFW (CTx).
Expert Report of the Honorable Gerald J. Mossinghoff, dated Mar. 13, 2006, from Civil Action No. 04-7456 JFW (CTx).
Faltstrom, P. et al., "How to Interact with a Whois++ Mesh," Feb. 1996, pp. 1-9.
Feeley, Michael, et al. "Implementing Global Memory Management in a Workstation Cluster." In Proc. of the 15th ACM Symp. on Operating Systems Principles, 1995, pp. 201-212.
Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jan. 1997, pp. 1-163.
Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jun. 1999, pp. 1-157.
First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, dated Nov. 24, 2004, from Civil Action No. CV 04-7456 JFW (CTx).
Floyd, Sally, et al. "A reliable Multicast Framework for Light-Weight Sessions and Application Level Framing." In Proceeding of ACM SIGCOMM '95, pp. 342-356.
Fredman, Michael, et al. "Storing a Sparse Table with 0(1) Worst Case Access Time." Journal of the Association for Computing Machinery, vol. 31, No. 3, Jul. 1984, pp. 538-544.
G. L. Friedman, Digital Camera With Apparatus For Authentication of Images Produced From an Image File, NASA Case No. NPO-19108-1-CU, U.S. Appl. No. 08/159,980, Nov. 24, 1993.
Grigni, Michelangelo, et al. "Tight Bounds on Minimum Broadcasts Networks." SIAM Journal of Discrete Mathematics, vol. 4, No. 2, May 1991, pp. 207-222.
Gwertzman, James, et al. "The Case for Geographical Push-Caching." Technical Report HU TR 34-94 (excerpt), Harvard University, DAS, Cambridge, MA 02138, 1994, 2 pgs.
Goodman, Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Structures and File Management Report Documentation p. AD-A275 385-94-04277.
H. Goodman, Feb. 9, 1994 Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Sructures and File Management Report Documentation P. AD-A275 385—94-04277.
Hauzeur, B. M., "A Model for Naming, Addressing, and Routing," ACM Trans. Inf. Syst. 4, Oct. 4, 1986), 293-311.
International Search Report dated Jun. 24, 1996 in corresponding international application PCT/US1996/004733.
K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names", www.w3.org/Addressing/rfc1737.txt, Dec. 1994, pp. 1-7.
Khare, R. and Lawrence, S., "Upgrading to TLS Within HTTP/1.1," May 2000, pp. 1-12.
Khoshafian, S. N. et al. 1986. Object identity. In Conf. Proc. on Object-Oriented Programming Systems, Languages and Applications (Portland, Oregon, United States, Sep. 29-Oct. 2, 1986). N. Meyrowitz, Ed. OOPLSA '86. ACM Press, New York, NY, 406-416.

Kim et al., "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1-12.
Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 23, 1995, pp. 1-18.
Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1-21.
Kim, Gene H., and Spafford, Eugene H., "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool." Coast Labs. Dept. of Computer Sciences Purdue University, Mar. 12, 1994, pp. 1-23.
Knuth, Donald E., "The Art of Computer Programming," 1973, vol. 3, Ch. 6.4, pp. 506-549.
Lantz, K. A., et al., "Towards a universal directory service." In Proc. 4th Annual ACM Symp. on Principles of Distributed Computing (Minaki, Ontario, Canada). PODC '85. ACM Press, New York, NY, 250-260.
Leach, P. J., et al.. The file system of an integrated local network. In Proc. 1985 ACM 13th Annual Conf. on Comp. Sci. CSC '85. ACM Press, NY, NY, 309-324.
Leach, P.J., et al., "UIDs as Internal Names in a Distributed File System," In Proc. 1st ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (Ottawa, Canada, Aug. 18-20, 1982). PODC '82. ACM Press, New York, NY, 34-41.
Ma, C. 1992. On building very large naming systems. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring (France, Sep. 21-23, 1992). EW 5. ACM Press, New York, NY, 1-5.
Memorandum of Points and Authorities in Support of Loudeye's and Overpeer's Motion to Dismiss the First Amended Complaint for Failure to State a Claim or, in the Alternative, for a More Definitive Statement, dated Dec. 13, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).
Ming-Ling Lo et al., On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69-78, May 1993.
Moats, R., "URN Syntax," May 1997, pp. 1-8.
Murlidhar Koushik, Dynamic Hashing With Distributed Overflow Space: A File Organization With Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299-317.
Myers, J. and Rose, M., "The Content-MD5 Header Field," Oct. 1995, pp. 1-4.
Naor, Moni, et al. "The Load, Capacity and Availability of Quorum Systems." In Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Nov. 1994, pp. 214-225.
Nisan, Noam. "Psuedorandom Generators for Space-Bounded Computation." In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, May 1990, pp. 204-212.
Office Action in corresponding Japanese Application No. 531,073/1996 mailed on Apr. 25, 2006.
Office Communication in corresponding European Application No. 96910762.2-1225 dated Jan. 17, 2007.
Order Re Claim Construction dated Nov. 8, 2001, from Civil Action No. 00-11851-RWZ.
Palmer, Mark, et al. "Fido: A Cache that Learns to Fetch." In Proceedings of the 17th International Conference on Very Large Data Bases, Sep. 1991, pp. 255-264.
Patent Abstracts of Japan, "Device for Generating Database and Method for the Same," Application No. 03-080504, Sun Microsyst. Inc., published Jun. 1993, 38 pages.
Patent Abstracts of Japan, "Electronic Mail Multiplexing System and Communication Control Method in the System." Jun. 30, 1993, JP 051625293.
Patent Abstracts of Japan, "Method for Registering and Retrieving Data Base," Application No. 03-187303, Nippon Telegr. & Teleph. Corp., published Feb. 1993, 11 pages.
Peleg, David, et al. "The Availability of Quorum Systems." Information and Computation 123, 1995, 210-223.
Peter Deutsch (peterd@bunyip.com), "Re: MD5 and LiFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0106.html, Apr. 26, 1994.
Peterson, L. L. 1988. A yellow-pages service for a local-area network. In Proc. ACM Workshop on Frontiers in Computer Communications Technology (Vermont, 1987). J. J. Garcia-Luna-Aceves, Ed. SIGCOMM '87. ACM Press, New York, NY, 235-242.
Plaintiffs' Memorandum of Points and Authorities in Opposition to Loudeye Defendants' Motion to Dismiss, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).
Plaintiffs' Opposition to Media Sentry's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV 04-7456 JFW (CTx).
Plaintiff's Opposition to Recording Industry Association of America's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (CTx).
Plaintiff's Reply to Defendant Loudeye Corp.'s and Overpeer, Inc.'s Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).
Plaintiff's Reply to Defendant MediaSentry's Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).
Plaintiff's Reply to Defendant RIAA's Counterclaims, dated Mar. 3, 2005, from Civil Action No. 04-7456 JFW (CTx).
Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.
Rabin, Michael. "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." Journal of the ACM, vol. 36, No. 2, Apr. 1989, pp. 335-348.
Ravi, R., "Rapid Rumor Ramification: Approximating the Minimum Broadcast Time." in Proc. of the 35th IEEE Symp. on Foundation of Computer Science, Nov. 1994, pp. 202-213.
Ravindran, K. and Ramakrishnan, K. K. 1991. A naming system for feature-based service specification in distributed operating systems. SIGSMALL/PC Notes 17, 3-4 (Sep. 1991), 12-21.
Reed Wade (wade@cs.utk.edu), "re: Dienst and BFD/LIFN document," Aug. 8, 1994, printed from http://www.webhistory.org/www.lists/www-talk1994q3/0416.html on Mar. 22, 2006, (7 pages).
Rivest, R., "The MD5 Message-Digest Algorithm," Apr. 1992, pp. 1-19 and errata sheet (1 page).
Rose, M., "The Content-MD5 Header Field," Nov. 1993, pp. 1-3.
Ross, K., "Hash-Routing for Collections of Shared Web Caches," IEEE Network Magazine, pp. 37-44, Nov.-Dec. 1997.
Sakti Pramanik et al., Multi-Directory Hasing, 1993, Info. Sys., vol. 18, No. 1, pp. 63-74.
Schmidt, Jeanette, et al. "Chernoff-Hoeffding Bounds for Applications with Limited Independence." In Proceedings of the 4th ASC-SIAM Symposium on Discrete Algorithms, 1993, pp. 331-340.
Schneier, Bruce, "One-Way Hash Functions, Using Crypographic Algorithms for Hashing," 1991, printed from http://202.179135.4/data/DDJ/articles/1991/9109/91909g/9109g.htm on Mar. 22, 2006.
Schwartz, M., et al. 1987. A name service for evolving heterogeneous systems. In Proc. 11th ACM Symp. on OS Principles (Texas, Nov. 8-11, 1987). SOSP '87. ACM Press, NY, NY, 52-62.
Search Report dated Jun. 24, 1996.
Shaheen-Gouda, A. and Loucks, L. 1992. Name borders. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring (Mont Saint-Michel, France, Sep. 21-23, 1992). EW 5. ACM Press, NY, NY, 1-6.
Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," Mar. 1989, pp. 1-25.
Tarjan, Robert Endre, et al. "Storing a Sparse Table." Communications of the ACM, vol. 22, No. 11, Nov. 1979, pp. 606-611.
Terry, D. B. 1984. An analysis of naming conventions for distributed computer systems. In Proc. ACM SIGCOMM Symp. on Communications Architectures and Protocols: Tutorials & Symp. SIGCOMM '84. ACM Press, NY, NY, 218-224.
Thomas A. Berson, Differential Cryptanalysis Mod 2.sup.32 with Applications to MD5, pp. 69-81, 1992.
Vijay Kumar, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109-113.
Vijay Kumar, A concurrency Control Mechanism based on Extendible Hashing for Main Memory Database Systems, pp. 109-113, ACM, vol. 3, 1989.

Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth }USENIX} Security Symposium, Santa Clara, CA, 1993, 11 pages.
Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, undated, printed from http://www.ja.net/CERI/Vincenzetti_and_Cotrozzi/ATP_Anti_Tamp on Mar. 22, 2006, 8 pages.
Vitter, Jeffrey Scott, et al. "Optimal Prefetching via Data Compression." In Proceedings of 32nd IEEE Symposium on Foundations of Computer Science, Nov. 1991, pp. 121-130.
W3C:ID, HTTP: A protocol for networked information, "Basic HTTP as defined in 1992", www.w3.org/Protocols/HTTP2.html, 1992.
Wegman, Mark, et al. "New Hash Functions and Their Use in Authentication and Set Equality." Journal of Computer and System Sciences vol. 22, Jun. 1981, pp. 265-279.
William Perrizo, et al., Distributed Join Processing Performance Evaluation, 1994. Twenty-Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236-244.
Witold Litwin et al., LH.sup.*-Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21, Jun. 1993, pp. 1-22.
Witold Litwin et al., Linear Hashing for Distributed Files, ACM SIGMOD, May 1993, pp. 327-336.
Witold Litwin, et al., LH*-Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21 Jun. 1993 pp. 1-22.
Yao, Andrew Chi-Chih. "Should Tables be Sorted?" Journal of the Association for Computing Machinery, vol. 28, No. 3, Jul. 1981, pp. 615-628.
Yuliang Zheng et al., HAVAL—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83-105.
Yuliang Zheng, et al., HAVAL—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83-105, Advances in Cryptology, AUSCRIPT '92, 1992.
Zhiyu Tian, et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3-13, SIGIR Forum, 1993.
Zhiyu Tian, et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3-13, SIGIR Forum, Spring 1993.
USPTO, U.S. Reexam Control No. 90/010,260, Notice of Intent to Issue Ex Parte Reexamination Certificate, Apr. 8, 2010.
WIPO, International Preliminary Examination Report (IPER), Jul. 1997, PCT/US96/04733 [5 pgs.].
Birrell et al., A Universal File Server, IEEE Trans. on Software Engineering, vol. SE-6, No. 5, Sep. 1980.
Filing in EPO in related application (EP1996910762), Amended claims with annotations [10 pgs.], Annex [12 pgs.], Letter filed in EPO [2 pgs.], claims [9 pgs.], Nov. 29, 2010.
Guy, R. G., Ficus: A Very Large Scale Reliable Distributed File System, Jun. 3, 1991, Technical Report CSD-910018, Computer Science Department, University of California Los Angeles, CA.
Guy, R.G.,et al, Implementation of the Ficus Replicated File System, Proc. of the Summer USENIX Conference, Anaheim, CA, Jun. 1990, pp. 63-71.
Heidemann, J.S., "Stackable Layers: An Architecture for File System Development," UCLA, Aug. 1991 (available as Technical Report CSD-910056 of the UCLA Computer Science Department).
Needham, R.M., et al., The Cap Filing System, Proc. Sixth ACM Symp. on Operating System Principles, Nov. 1977, 11-16.
Page, Thomas W. Jr., et al., Management of Replicated Volume Location Data in the Ficus Replicated File System, Proc. of the Summer USENIX Conference, Jun. 1991, pp. 17-29.
Popek, Gerald J. et al., Replication in Ficus Distributed File Systems, Proc. of the Workshop on Management of Replicated Data, Nov. 1990, pp. 20-25.
Reiher, P. et al., Resolving File Conflicts in the Ficus File System, in Proc. of the Summer USENIX Conference, Jun. 1994, pp. 183-195.
Sturgis, H. et al, Issues in the design and use of a distributed file system, SIGOPS Oper. Syst. Rev., vol. 14, No. 3. (1980), pp. 55-69.
Swinehart, D., et al, WFS: A Simple Shared File System for a Distributed Environment, XEROX, Palo Alto Research Center, Palo Alto, CA, CSL-79-13, Oct. 1979.
Bowman, C.M., et al., "Scalable Internet Resource Discovery: Research Problems and Approaches," University of Colorado, Dept. of Comp. Sci. Technical Report# CU-CS-679-93 (Oct. 1993).
Bowman, C.M., et al., "Research Problems for Scalable Internet Resource Discovery," CU-CS-643-93, Mar. 1993, University of Colorado at Boulder, Dept. of Comp. Sci.
Cate, V., "Alex—a Global Filesystem," Proceedings of the Usenix File Systems Workshop, pp. 1-11, May 1992.
Dewitt, et al. (Jun. 1984). "Implementation techniques for main memory database systems". Proc. ACM SIGMOD Conf 14 (4): 1-8.
Fagin, R., et al. Extendible hashing—a fast access method for dynamic files. ACM Trans. Database Syst. 4, 3 (Sep. 1979), 315-344.
Griswold, William G.; Townsend, Gregg M. (Apr. 1993), "The Design and Implementation of Dynamic Hashing for Sets and Tables in Icon", Software—Practice and Experience 23 (4): 351-367 (Apr. 1993).
Hardy D. R., et al., "Essence: A resource discovery system based on semantic file indexing," USENIX Winter 1993 Technical Conf., San Diego, CA (Jan. 1993), pp. 361-374.
Kahn, R.E., "Deposit, Registration and Recordation in an Electronic Copyright Management System," Tech. Report, Corporation for National Research Initiatives, Reston, Virginia, Aug. 1992 (downloaded from http://archive.ifla.org/documents/infopol/copyright/kahn.txt on Nov. 10, 2010).
Kitsuregawa, M.; Tanaka, H.; Moto-Oka, T. (Mar. 1983). "Application of Hash to Data Base Machine and Its Architecture". New Generation Computing 1 (1): 63-74, OHMSHA, Ltd. and Springer-Verlag.
Litwin, W., "Linear Hashing: a New Tool for File and Table Addressing," Proc. 6th Int. Conf. on Very Large Databases, IEEE 1980, 212-223.
Manber, U., Finding similar files in a large file system, Dept. of Computer Science TR 93-33, Oct. 1993., U. of Arizona, Tucson, AZ.
Manber, U., Finding similar files in a large file system, USENIX, pp. 1-10, San Francisco, CA, Jan. 1994. (WTEC'94, Proc. USENIX Winter 1994 Technical Conf.).
USPTO, Non-Final Office Action in U.S. Appl. No. 11/980,677, Jun. 4, 2010.
USPTO, Notice of Allowance mailed Apr. 30, 2010 in U.S. Appl. No. 11/980,687.
USPTO, Notice of Allowance mailed Jun. 24, 2010 in U.S. Appl. No. 11/980,687.
USPTO, Supplemental Notice of Allowability mailed May 27, 2010 in U.S. Appl. No. 11/980,687.

* cited by examiner

FIG. 1(a)


106
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
104
STORAGE DEVICE
104
STORAGE DEVICE
100

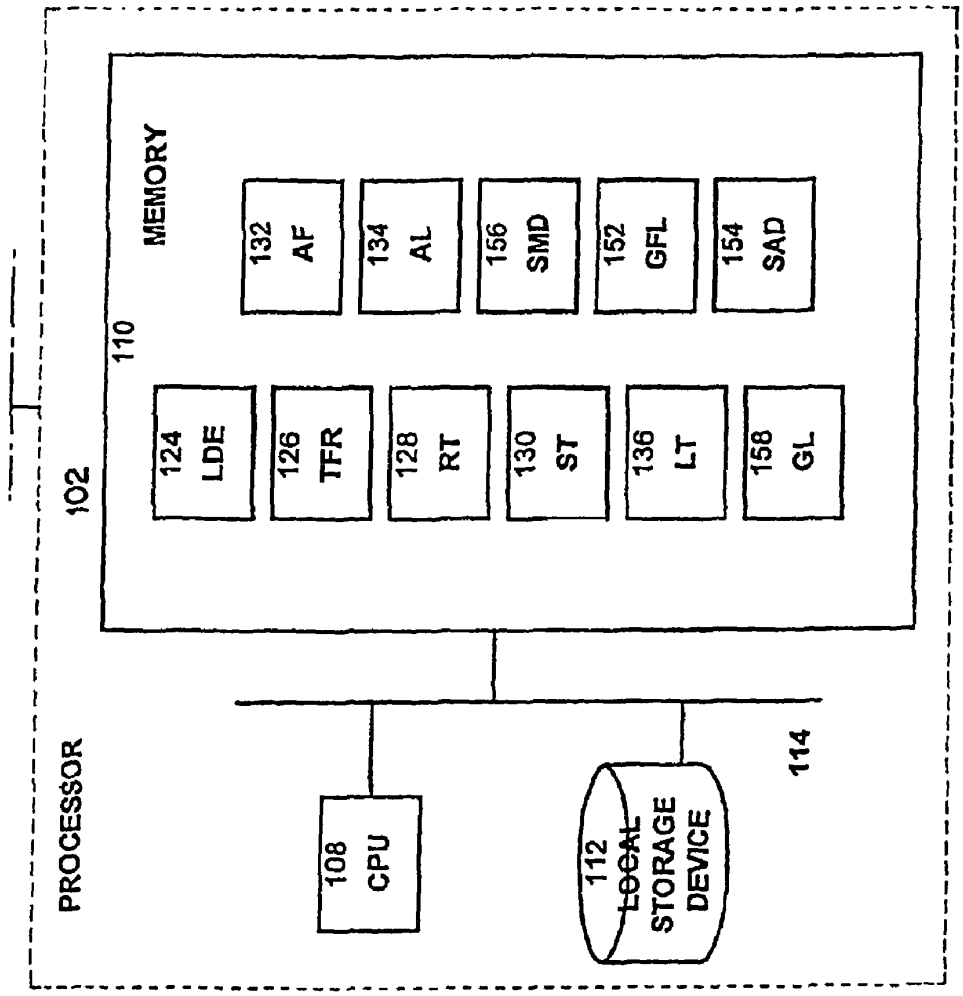

PROCESSOR
102
MEMORY
110
132 AF
134 AL
156 SMD
152 GFL
154 SAD
124 LDE
126 TFR
128 RT
130 ST
136 LT
158 GL
108 CPU
112 LOCAL STORAGE DEVICE
114

FIG. 1(b)

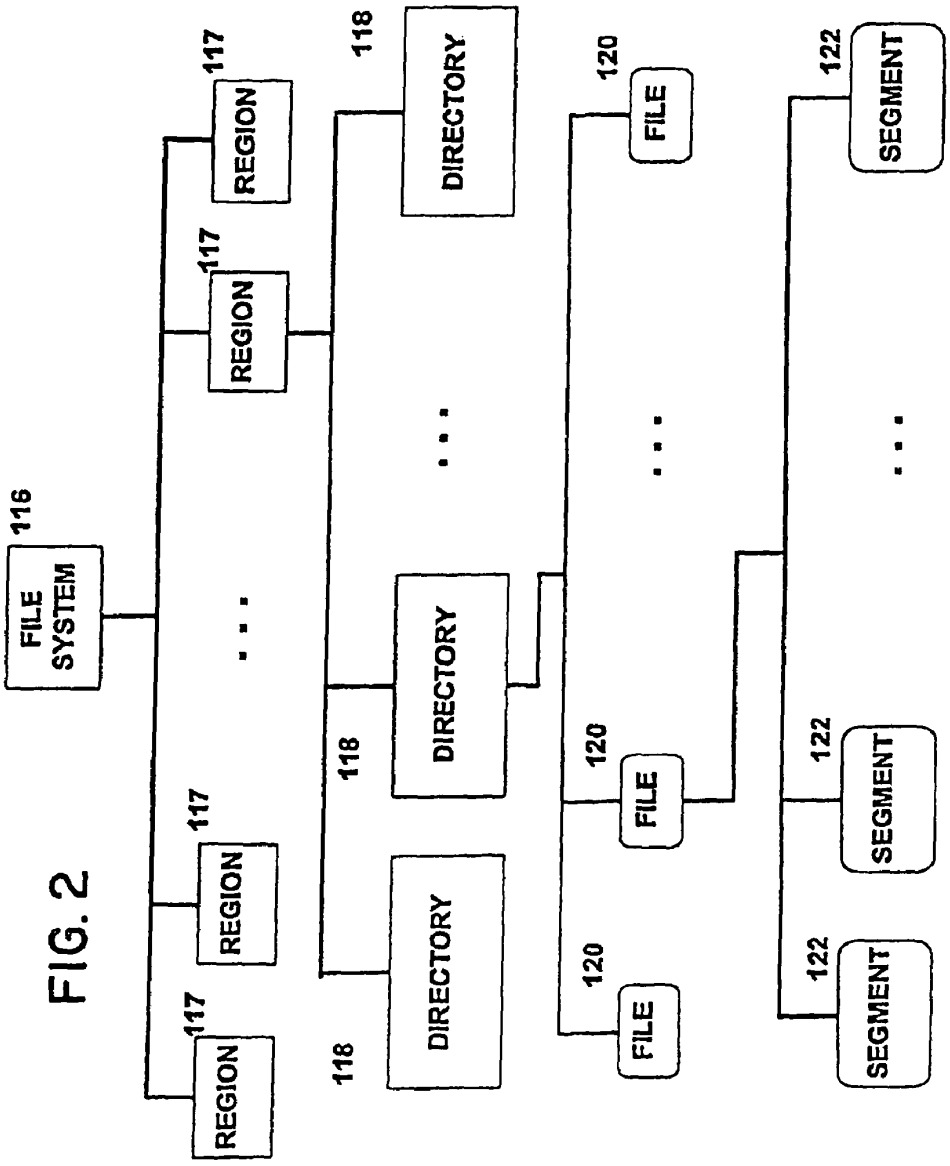

116
FILE SYSTEM
117
REGION
117
REGION
117
REGION
118
DIRECTORY
118
DIRECTORY
118
DIRECTORY
120
FILE
120
FILE
120
FILE
122
SEGMENT
122
SEGMENT
122
SEGMENT

FIG. 2

FIG. 3

138

Region ID
Pathname
True Name
Type
File ID
Time of last access
Time of last modification
Safe flag
Lock flag
Size
Owner

FIG. 4

140

True Name
File ID
Compressed File ID
Source IDs
Dependent processors
Use count
Time of last access
Expiration
Grooming delete count

142

Region ID
Region file system
Region pathname
Region status
Mirror processor(s)
Mirror duplication count
Policy

FIG. 5

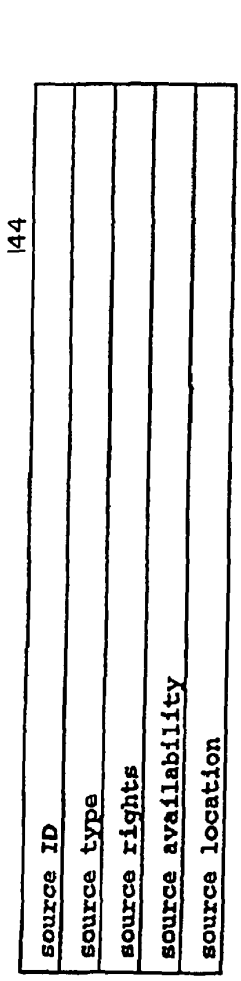


FIG. 6

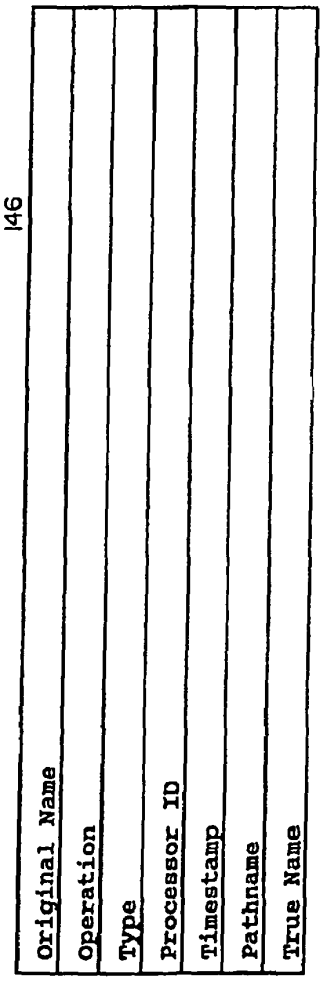


FIG. 7

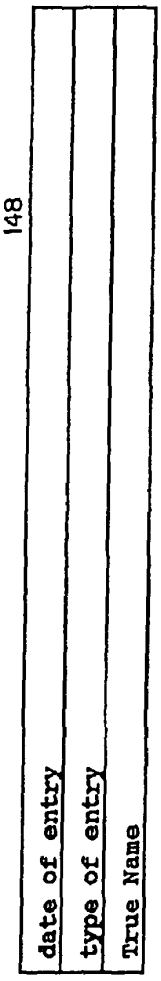
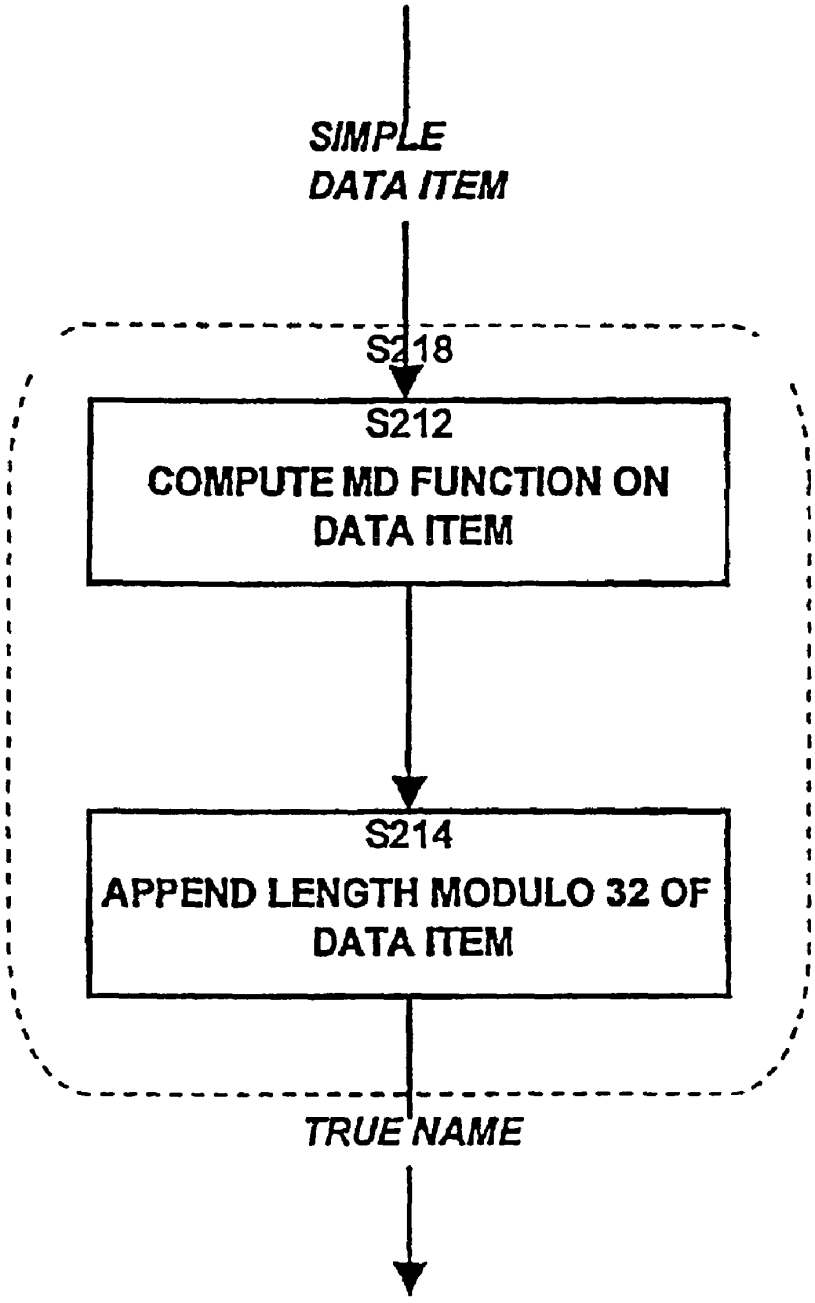


FIG. 8



FIG. 9

FIG. 10(a)


SIMPLE DATA ITEM
S218
S212
COMPUTE MD FUNCTION ON DATA ITEM
S214
APPEND LENGTH MODULO 32 OF DATA ITEM
TRUE NAME

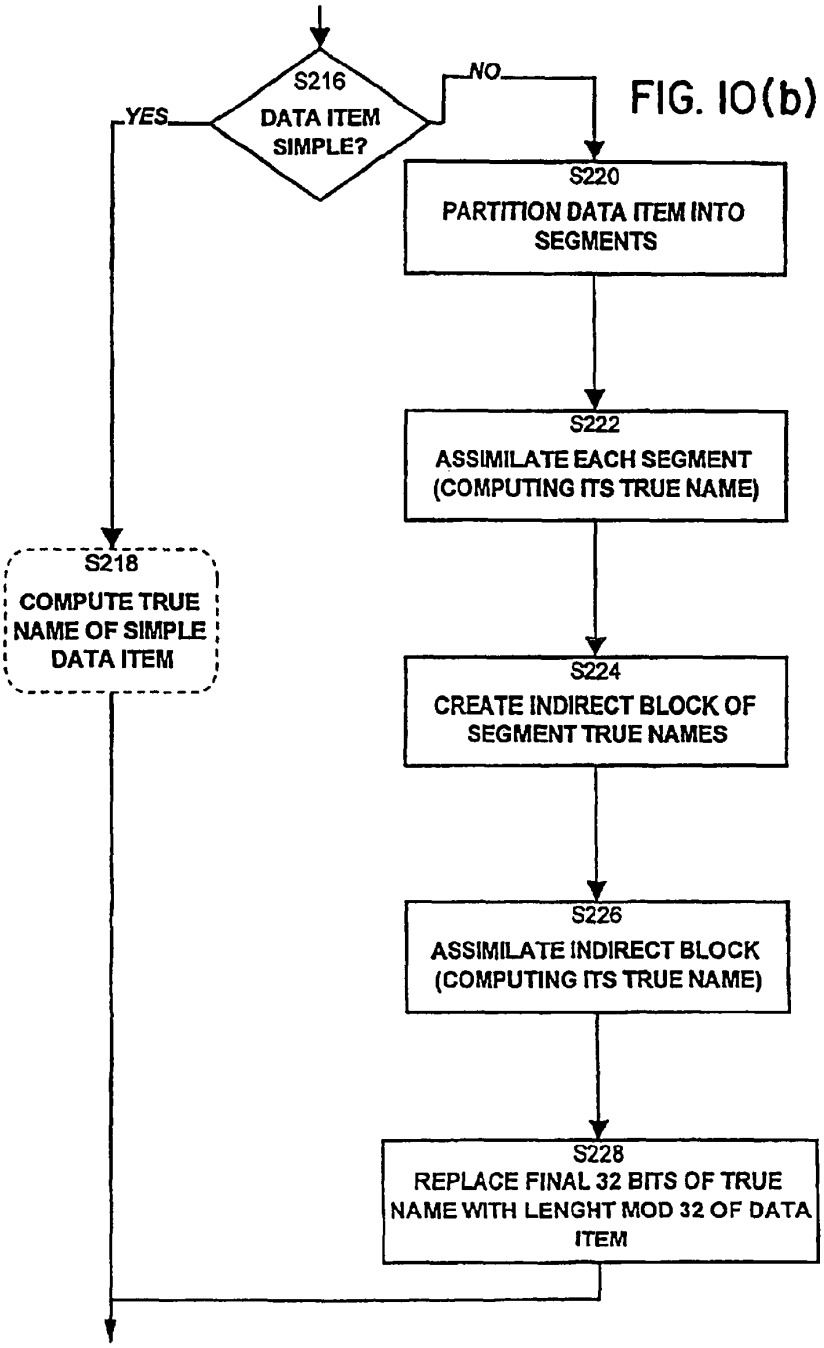

S216
DATA ITEM SIMPLE?
YES
NO
S218
COMPUTE TRUE NAME OF SIMPLE DATA ITEM
S220
PARTITION DATA ITEM INTO SEGMENTS
S222
ASSIMILATE EACH SEGMENT (COMPUTING ITS TRUE NAME)
S224
CREATE INDIRECT BLOCK OF SEGMENT TRUE NAMES
S226
ASSIMILATE INDIRECT BLOCK (COMPUTING ITS TRUE NAME)
S228
REPLACE FINAL 32 BITS OF TRUE NAME WITH LENGHT MOD 32 OF DATA ITEM

FIG. 10(b)

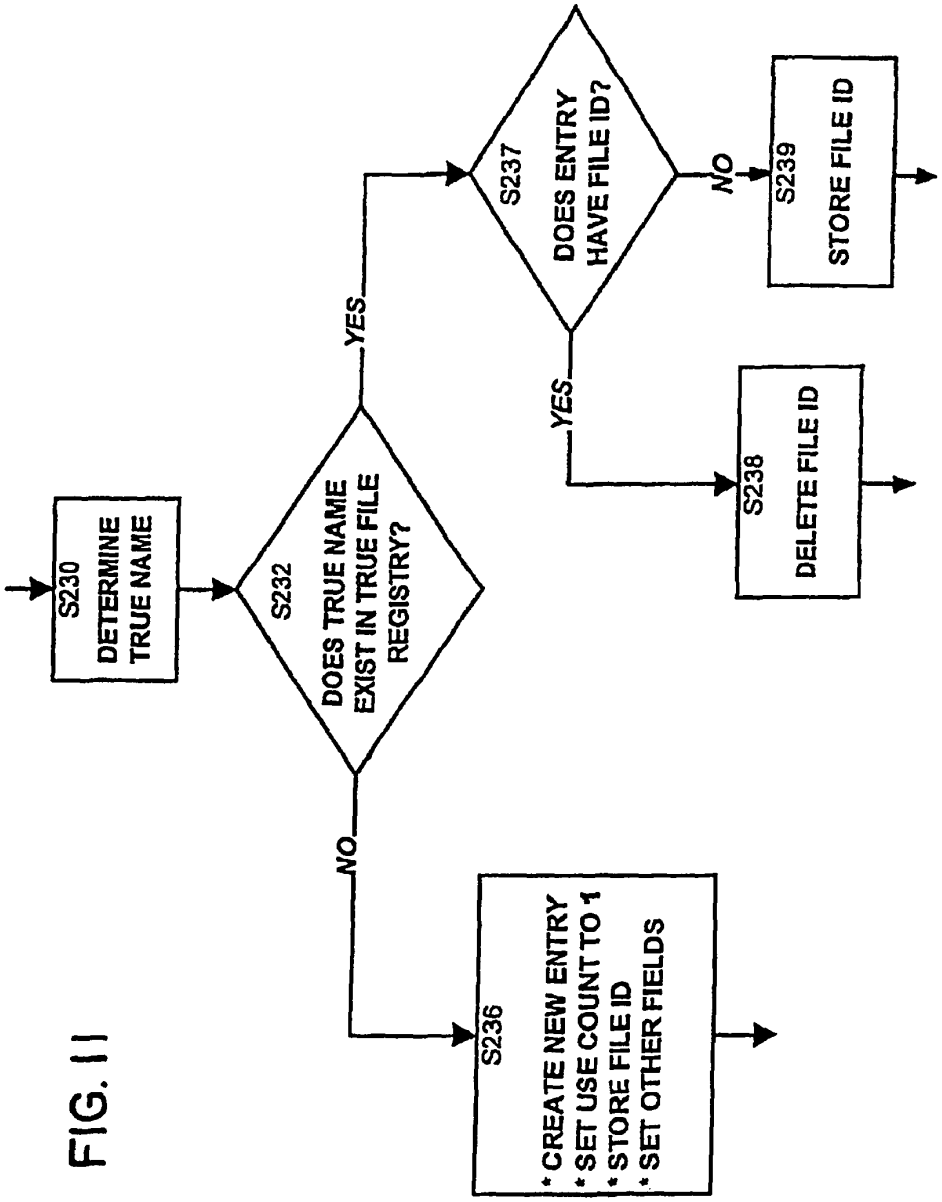

S230
DETERMINE TRUE NAME
S232
DOES TRUE NAME EXIST IN TRUE FILE REGISTRY?
YES
NO
S237
DOES ENTRY HAVE FILE ID?
YES
NO
S238
DELETE FILE ID
S239
STORE FILE ID
S236
* CREATE NEW ENTRY
* SET USE COUNT TO 1
* STORE FILE ID
* SET OTHER FIELDS

FIG. 11

FIG. 12

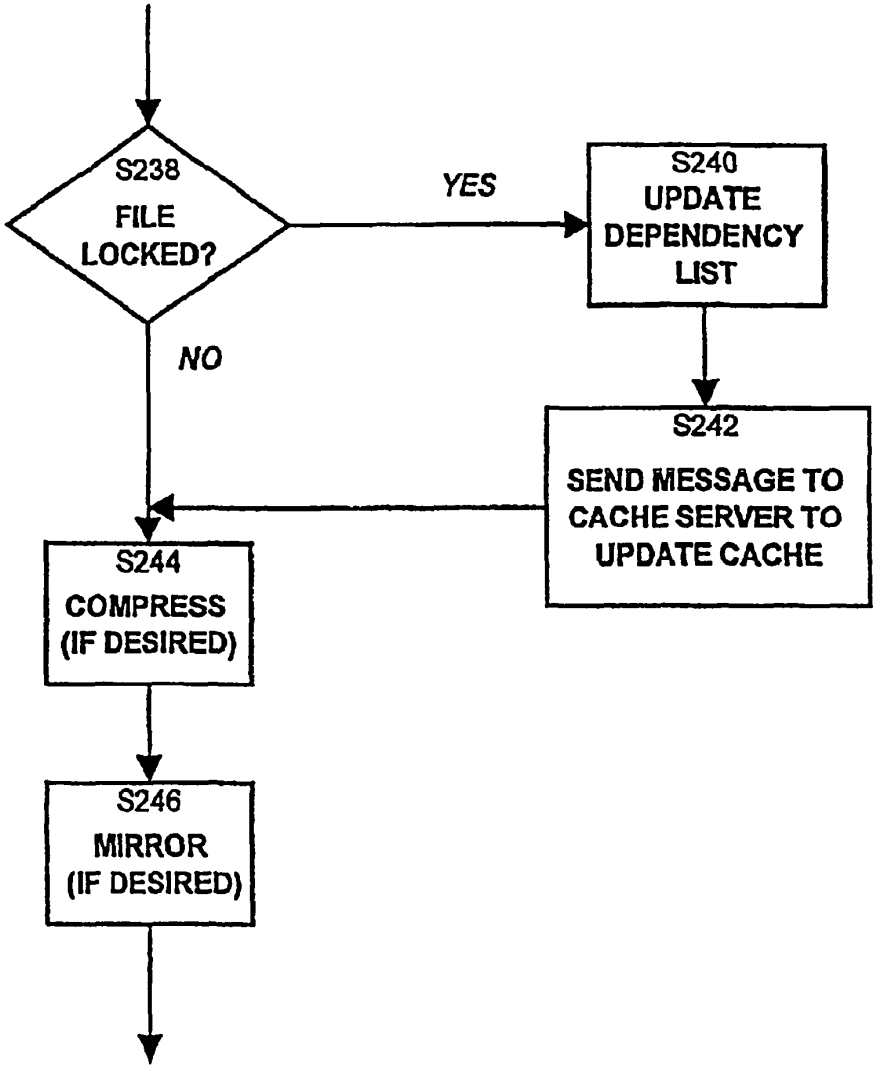
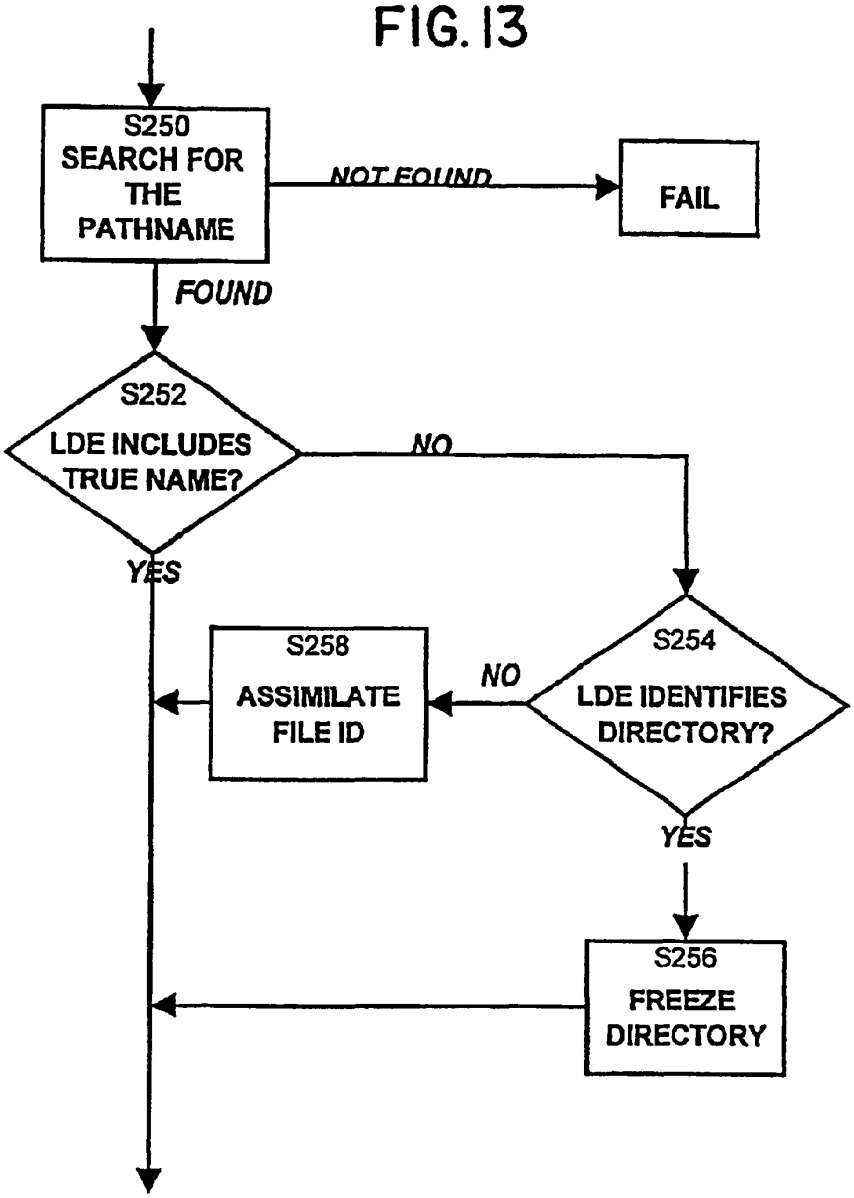

S238
FILE LOCKED?
YES
S240
UPDATE DEPENDENCY LIST
NO
S242
SEND MESSAGE TO CACHE SERVER TO UPDATE CACHE
S244
COMPRESS (IF DESIRED)
S246
MIRROR (IF DESIRED)

FIG. 13


S250
SEARCH FOR THE PATHNAME
NOT FOUND
FAIL
FOUND
S252
LDE INCLUDES TRUE NAME?
NO
YES
S254
LDE IDENTIFIES DIRECTORY?
NO
S258
ASSIMILATE FILE ID
YES
S256
FREEZE DIRECTORY

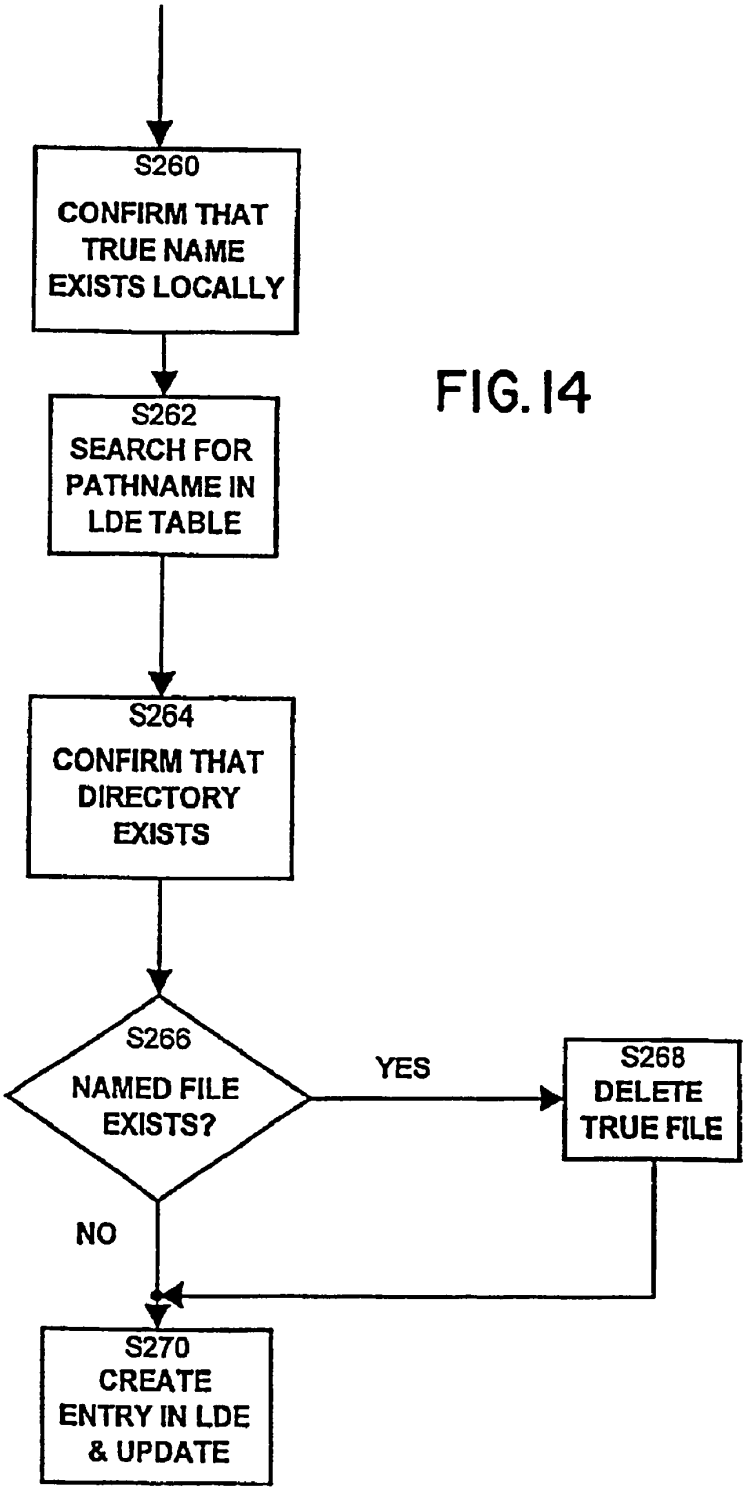

S260
CONFIRM THAT TRUE NAME EXISTS LOCALLY
S262
SEARCH FOR PATHNAME IN LDE TABLE
S264
CONFIRM THAT DIRECTORY EXISTS
S266
NAMED FILE EXISTS?
YES
S268
DELETE TRUE FILE
NO
S270
CREATE ENTRY IN LDE & UPDATE

FIG. 14

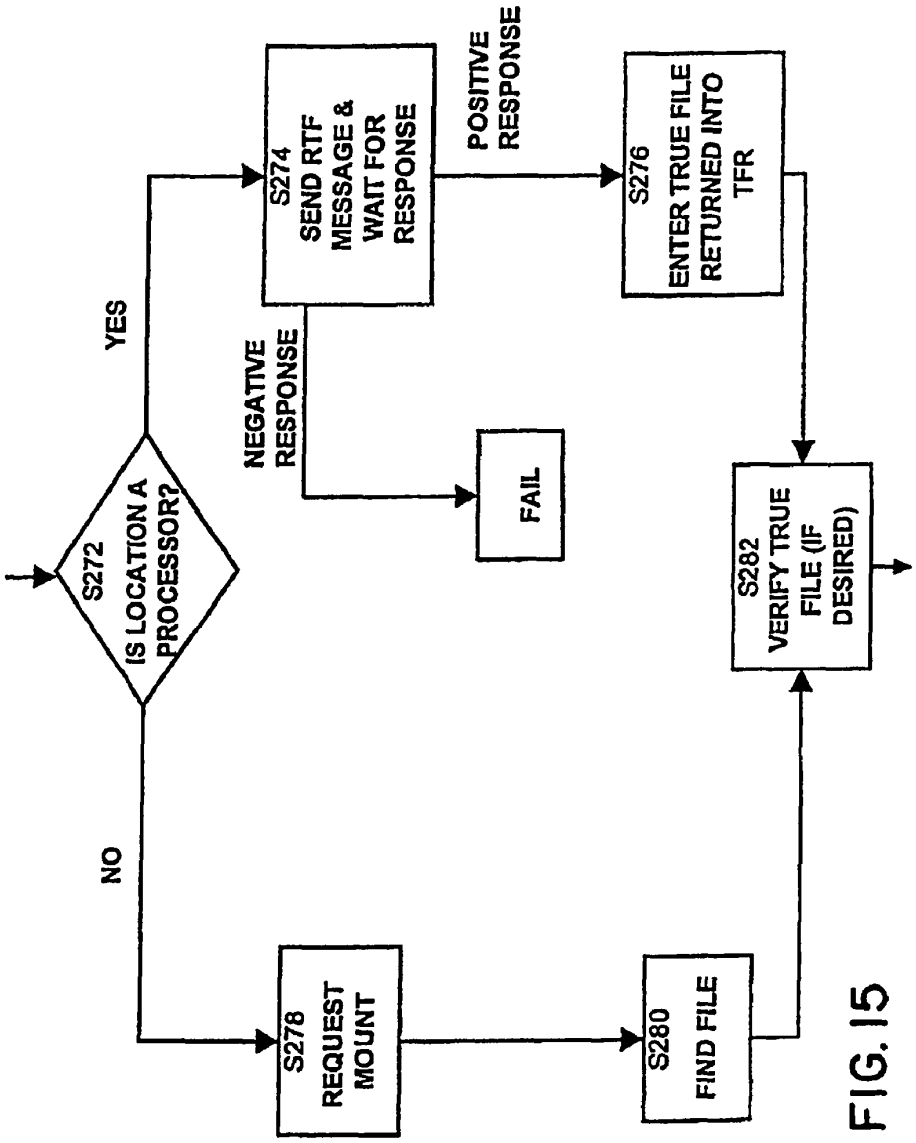

S272
IS LOCATION A PROCESSOR?
YES
NO
S274
SEND RTF MESSAGE & WAIT FOR RESPONSE
POSITIVE RESPONSE
NEGATIVE RESPONSE
FAIL
S276
ENTER TRUE FILE RETURNED INTO TFR
S278
REQUEST MOUNT
S280
FIND FILE
S282
VERIFY TRUE FILE (IF DESIRED)

FIG. 15

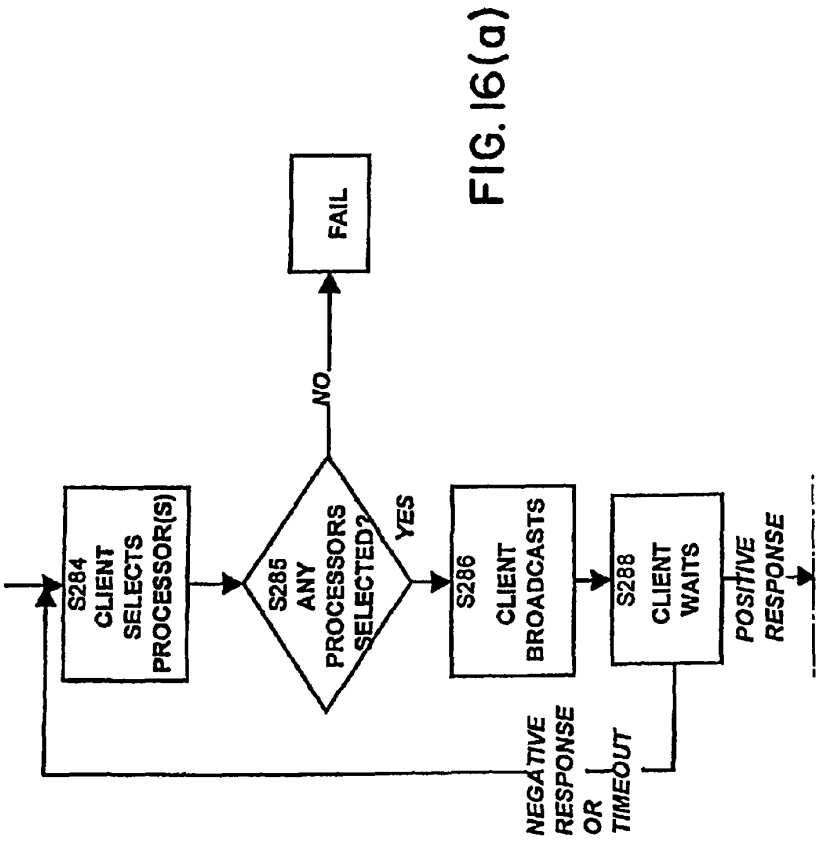

S284
CLIENT
SELECTS
PROCESSOR(S)
S285
ANY
PROCESSORS
SELECTED?
NO
FAIL
YES
S286
CLIENT
BROADCASTS
S288
CLIENT
WAITS
POSITIVE
RESPONSE
NEGATIVE
RESPONSE
OR
TIMEOUT

FIG. 16(a)

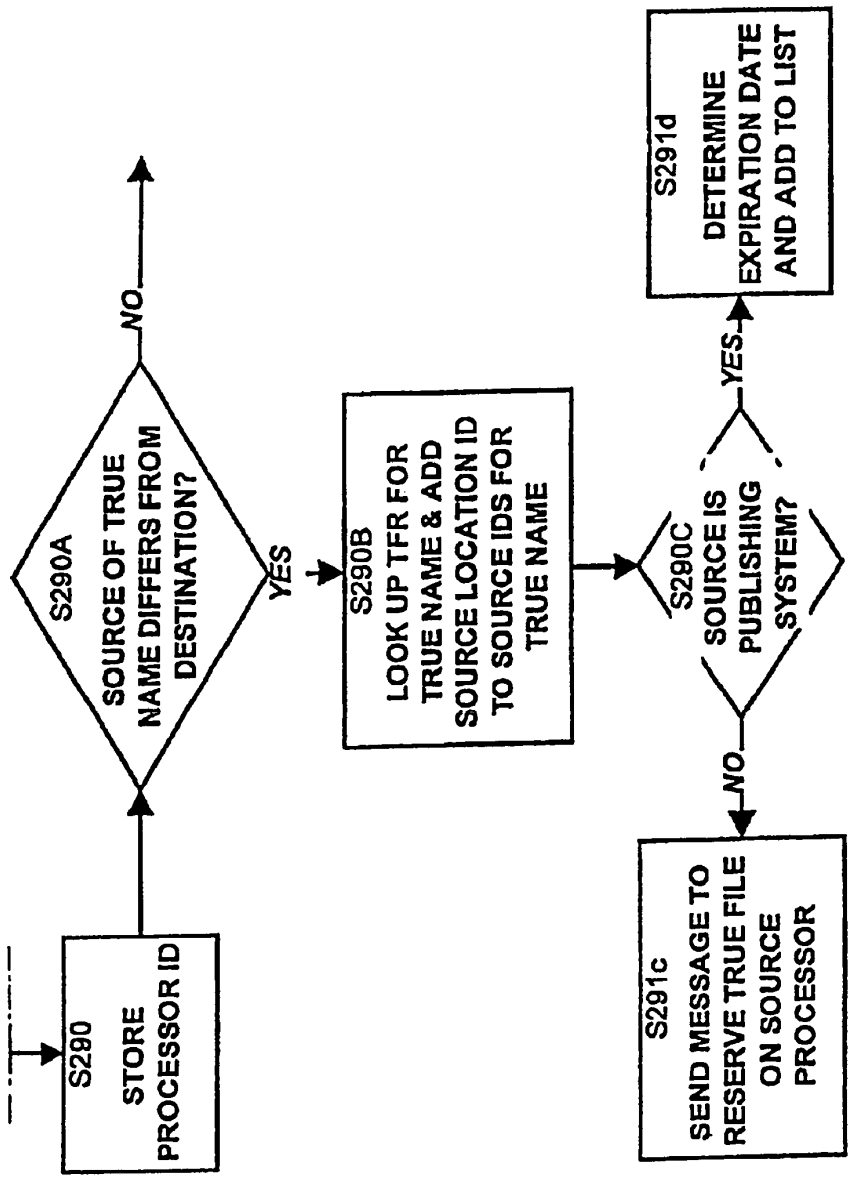

S290
STORE PROCESSOR ID
S290A
SOURCE OF TRUE NAME DIFFERS FROM DESTINATION?
NO
YES
S290B
LOOK UP TFR FOR TRUE NAME & ADD SOURCE LOCATION ID TO SOURCE IDS FOR TRUE NAME
S290C
SOURCE IS PUBLISHING SYSTEM?
YES
NO
S291d
DETERMINE EXPIRATION DATE AND ADD TO LIST
S291c
SEND MESSAGE TO RESERVE TRUE FILE ON SOURCE PROCESSOR

FIG. 16(b)

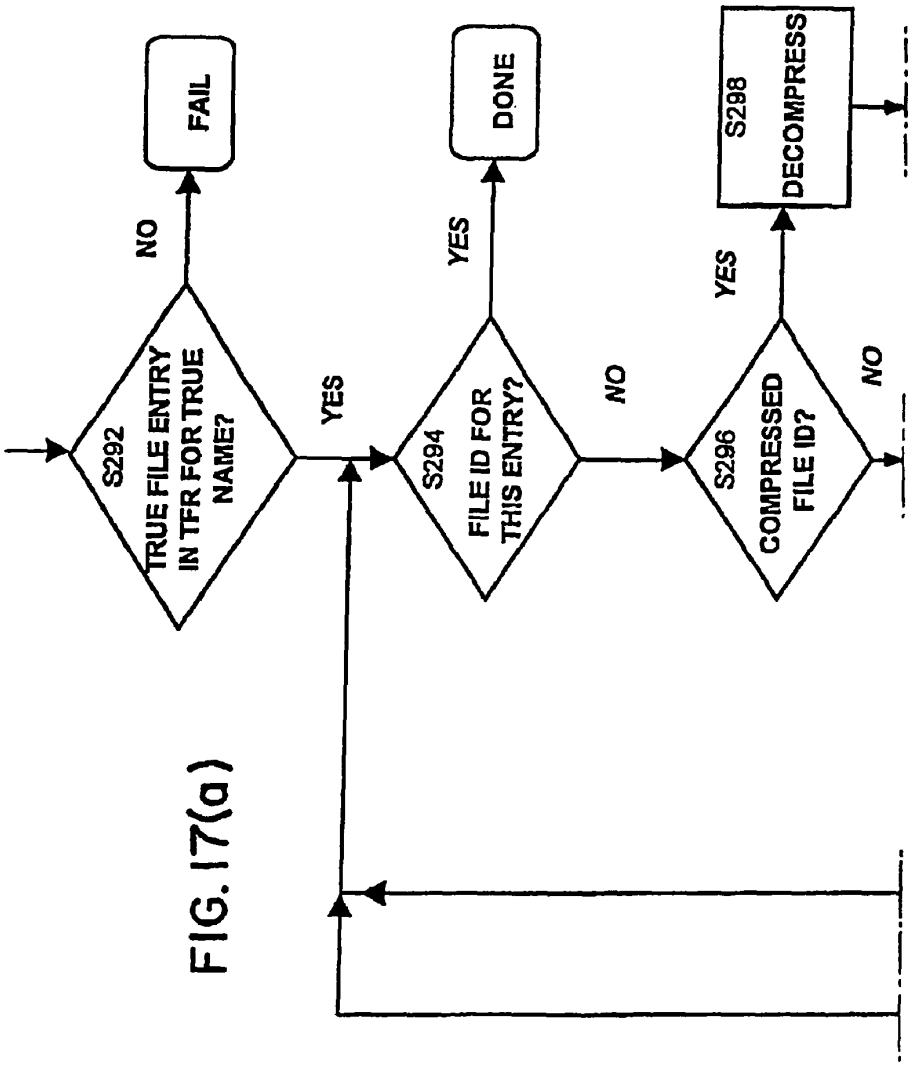

S292
TRUE FILE ENTRY IN TFR FOR TRUE NAME?
NO
FAIL
YES
S294
FILE ID FOR THIS ENTRY?
YES
DONE
NO
S296
COMPRESSED FILE ID?
YES
S298
DECOMPRESS
NO

FIG. 17(a)

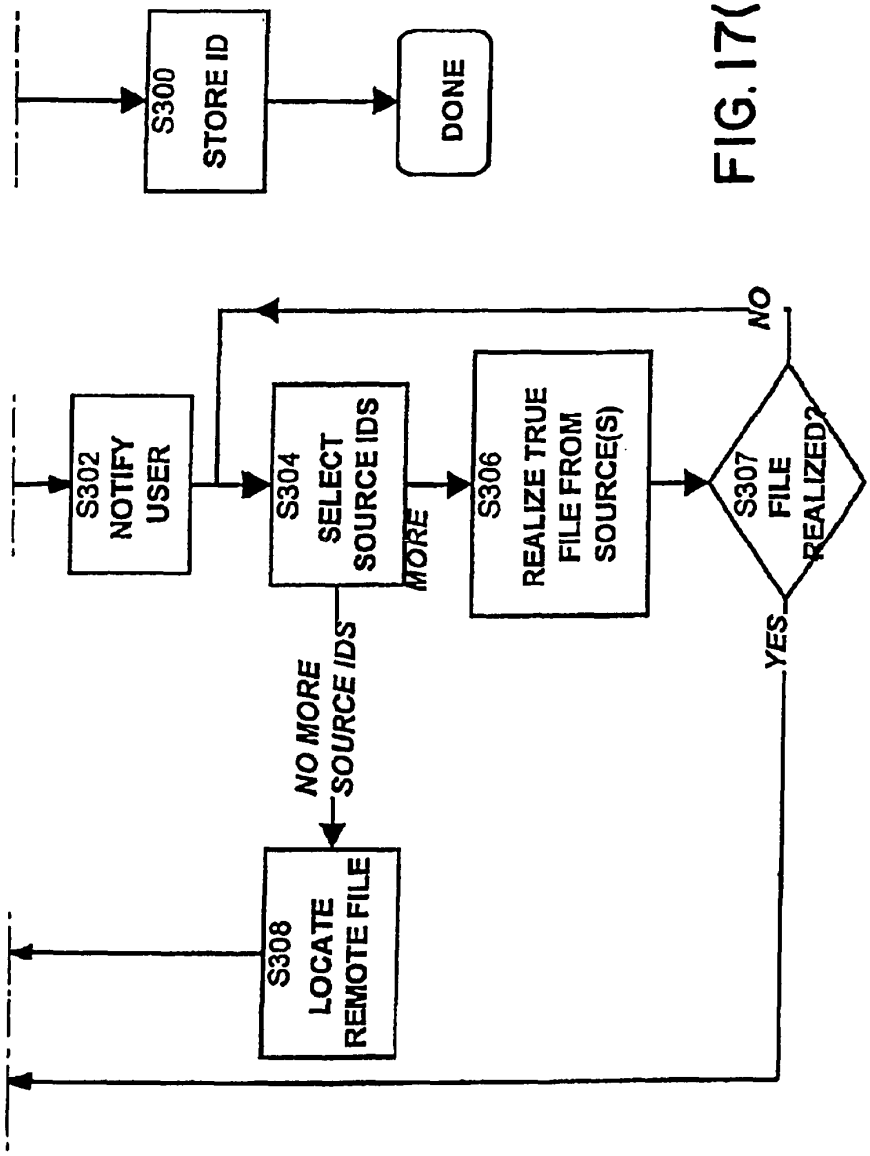
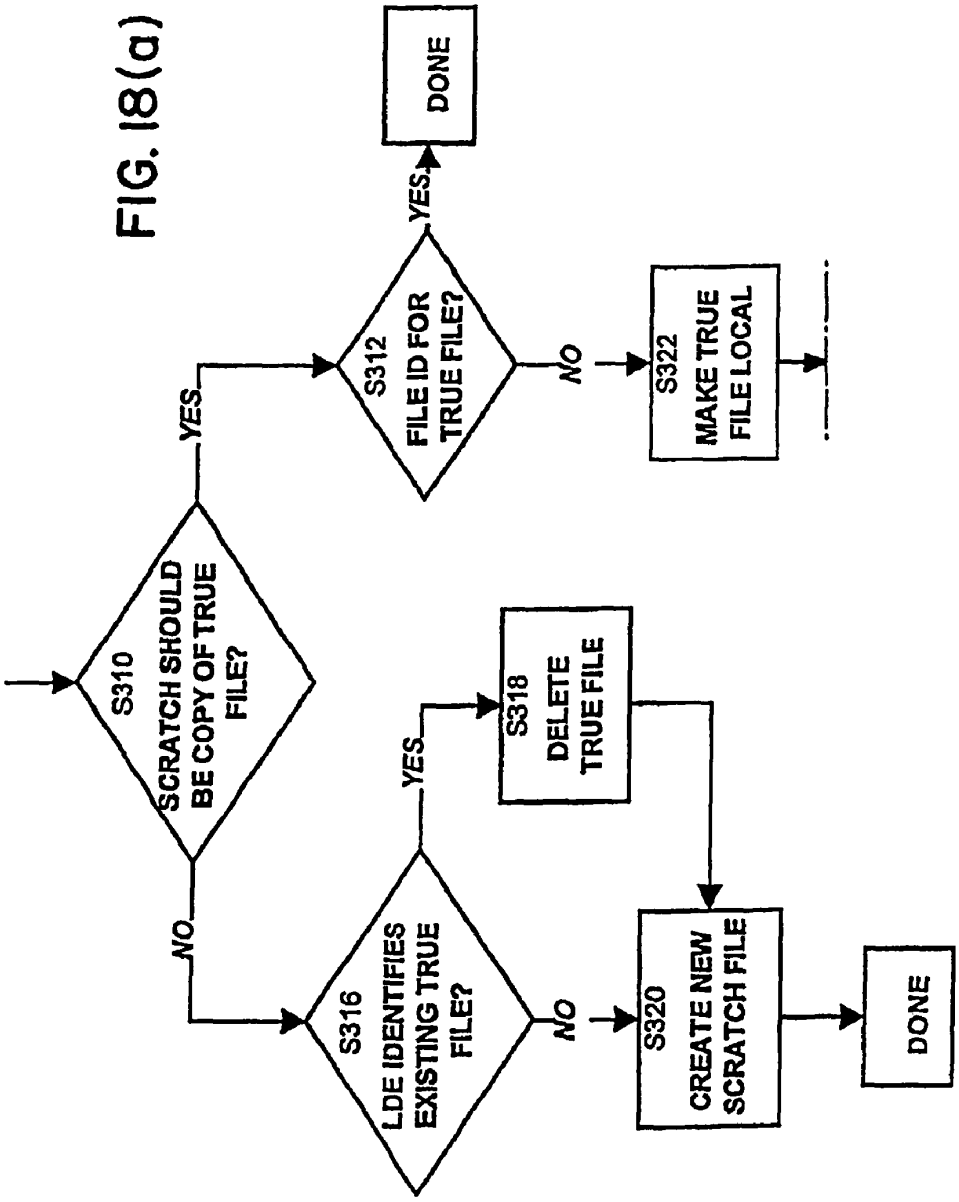

S300
STORE ID
DONE
S302
NOTIFY USER
S304
SELECT SOURCE IDS
MORE
NO MORE SOURCE IDS
S306
REALIZE TRUE FILE FROM SOURCE(S)
S307
FILE REALIZED?
NO
YES
S308
LOCATE REMOTE FILE

FIG. 17(b)

FIG. 18(a)


S310
SCRATCH SHOULD BE COPY OF TRUE FILE?
YES
NO
S312
FILE ID FOR TRUE FILE?
YES
NO
DONE
S322
MAKE TRUE FILE LOCAL
S316
LDE IDENTIFIES EXISTING TRUE FILE?
YES
NO
S318
DELETE TRUE FILE
S320
CREATE NEW SCRATCH FILE
DONE

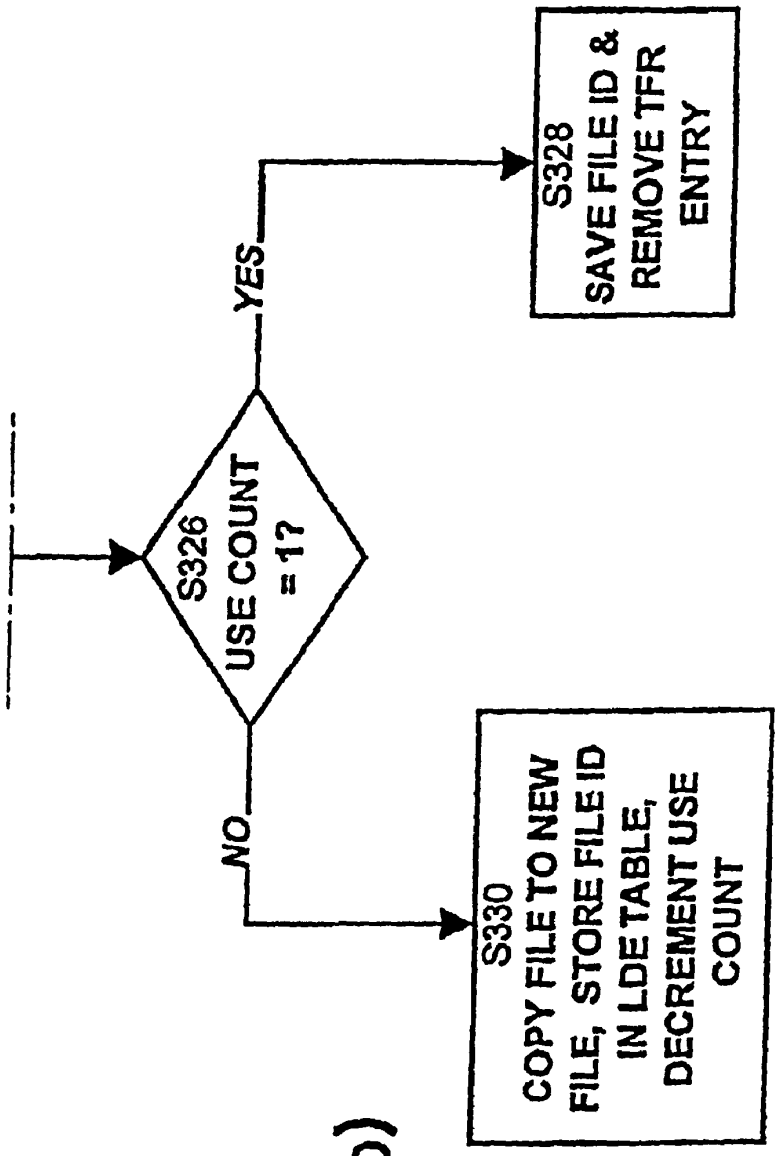
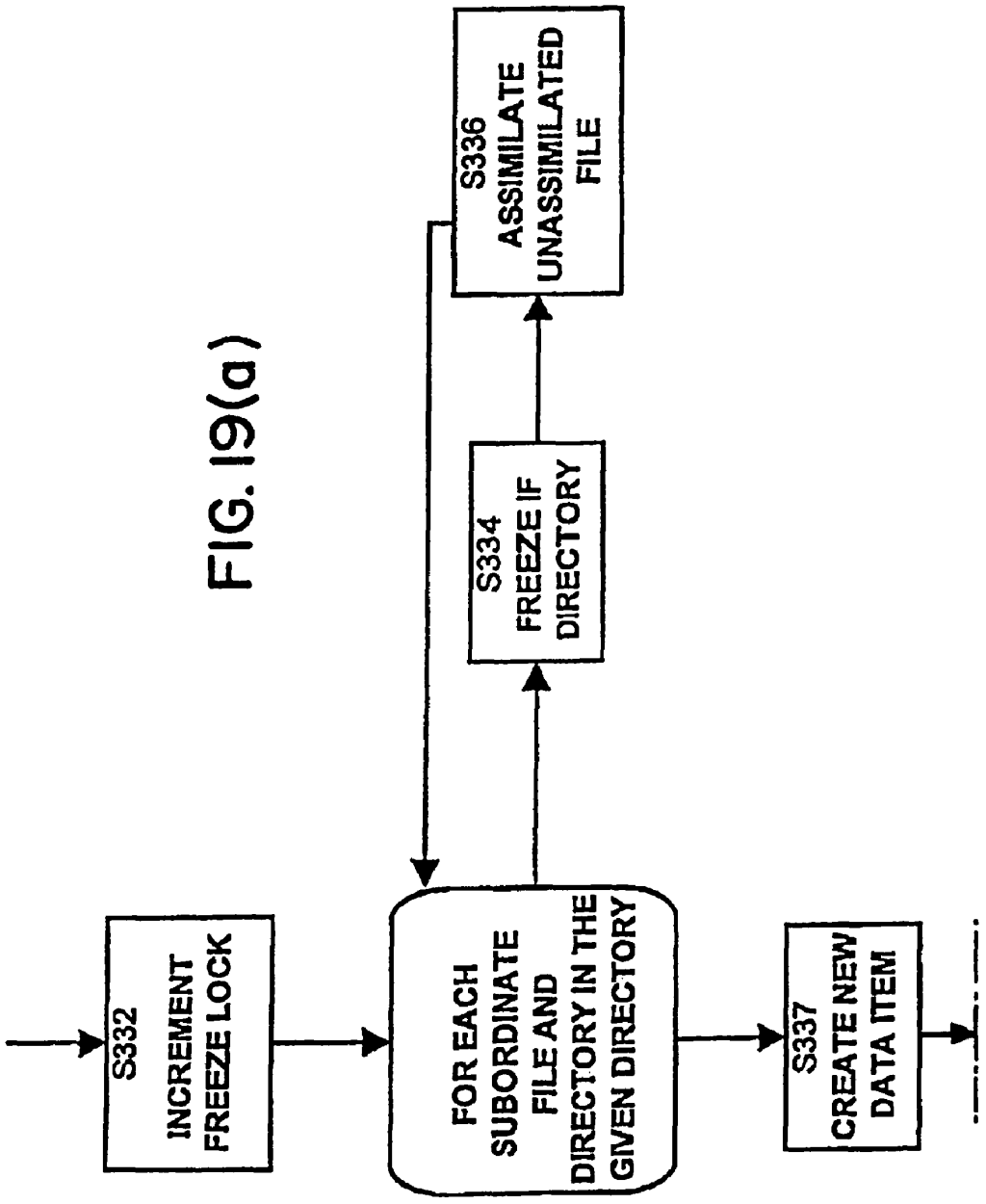

S326
USE COUNT
= 1?
YES
NO
S328
SAVE FILE ID &
REMOVE TFR
ENTRY
S330
COPY FILE TO NEW
FILE, STORE FILE ID
IN LDE TABLE,
DECREMENT USE
COUNT

FIG. 18(b)

FIG. 19(a)


S332
INCREMENT FREEZE LOCK
FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S334
FREEZE IF DIRECTORY
S336
ASSIMILATE UNASSIMILATED FILE
S337
CREATE NEW DATA ITEM

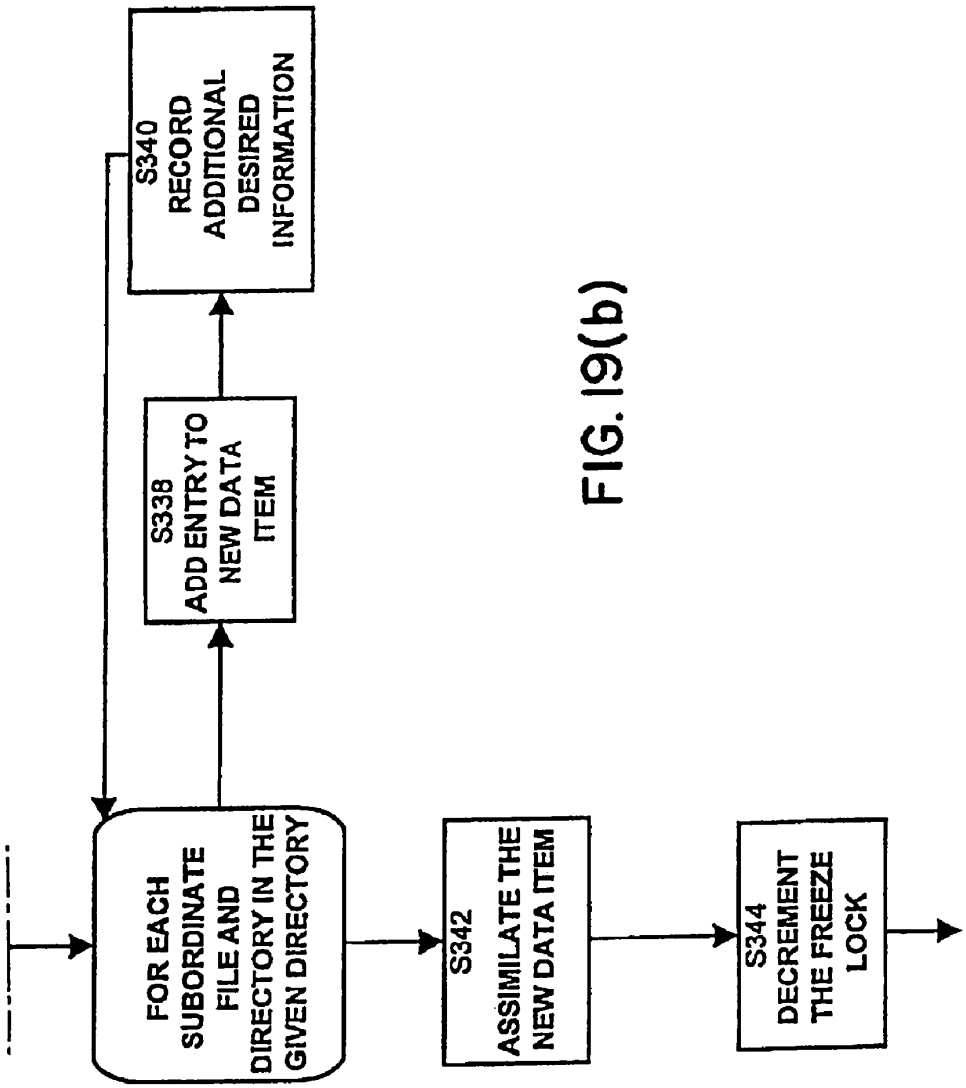

FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S338 ADD ENTRY TO NEW DATA ITEM
S340 RECORD ADDITIONAL DESIRED INFORMATION
S342 ASSIMILATE THE NEW DATA ITEM
S344 DECREMENT THE FREEZE LOCK

FIG. 19(b)

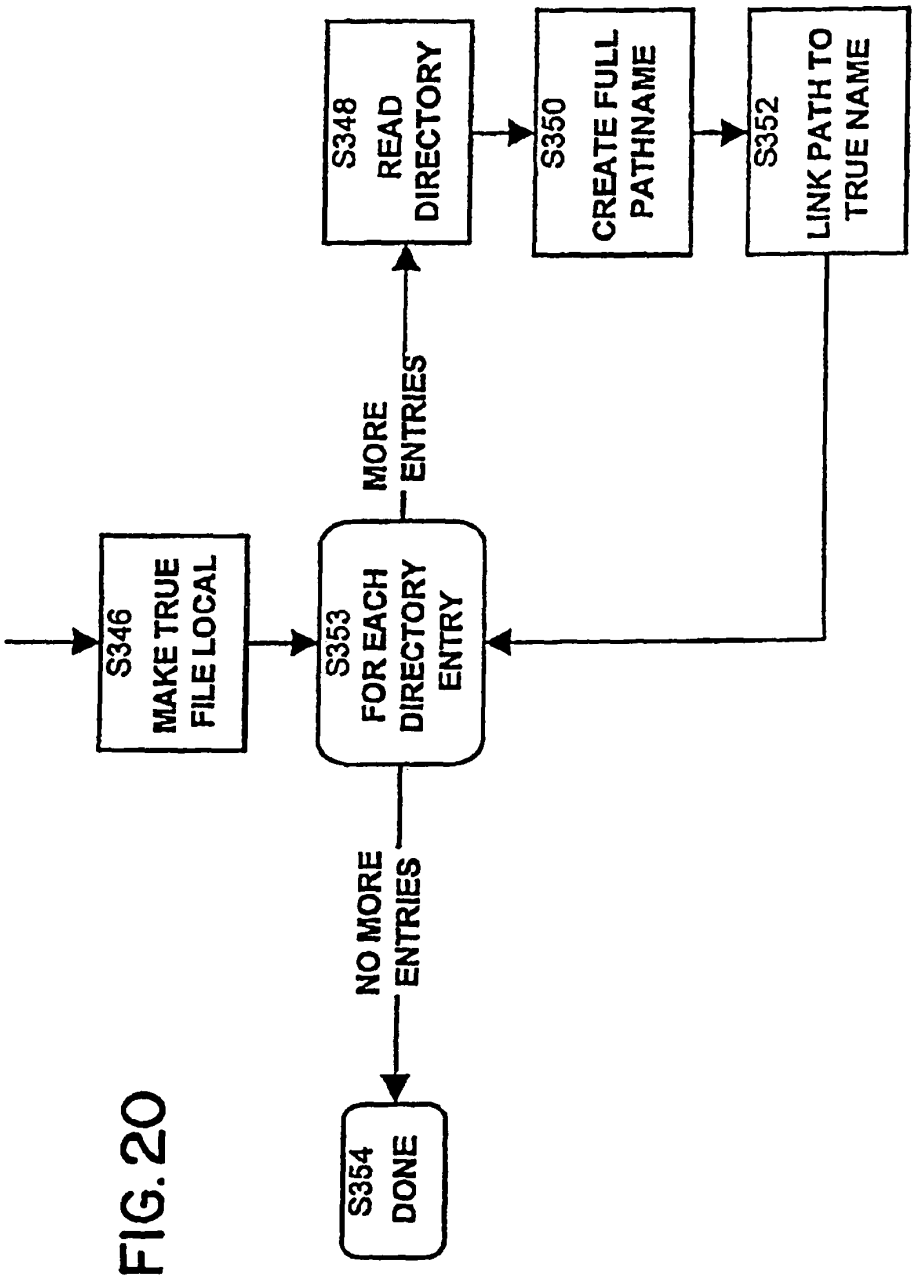

S346
MAKE TRUE FILE LOCAL
S353
FOR EACH DIRECTORY ENTRY
MORE ENTRIES
S348
READ DIRECTORY
S350
CREATE FULL PATHNAME
S352
LINK PATH TO TRUE NAME
NO MORE ENTRIES
S354
DONE

FIG. 20

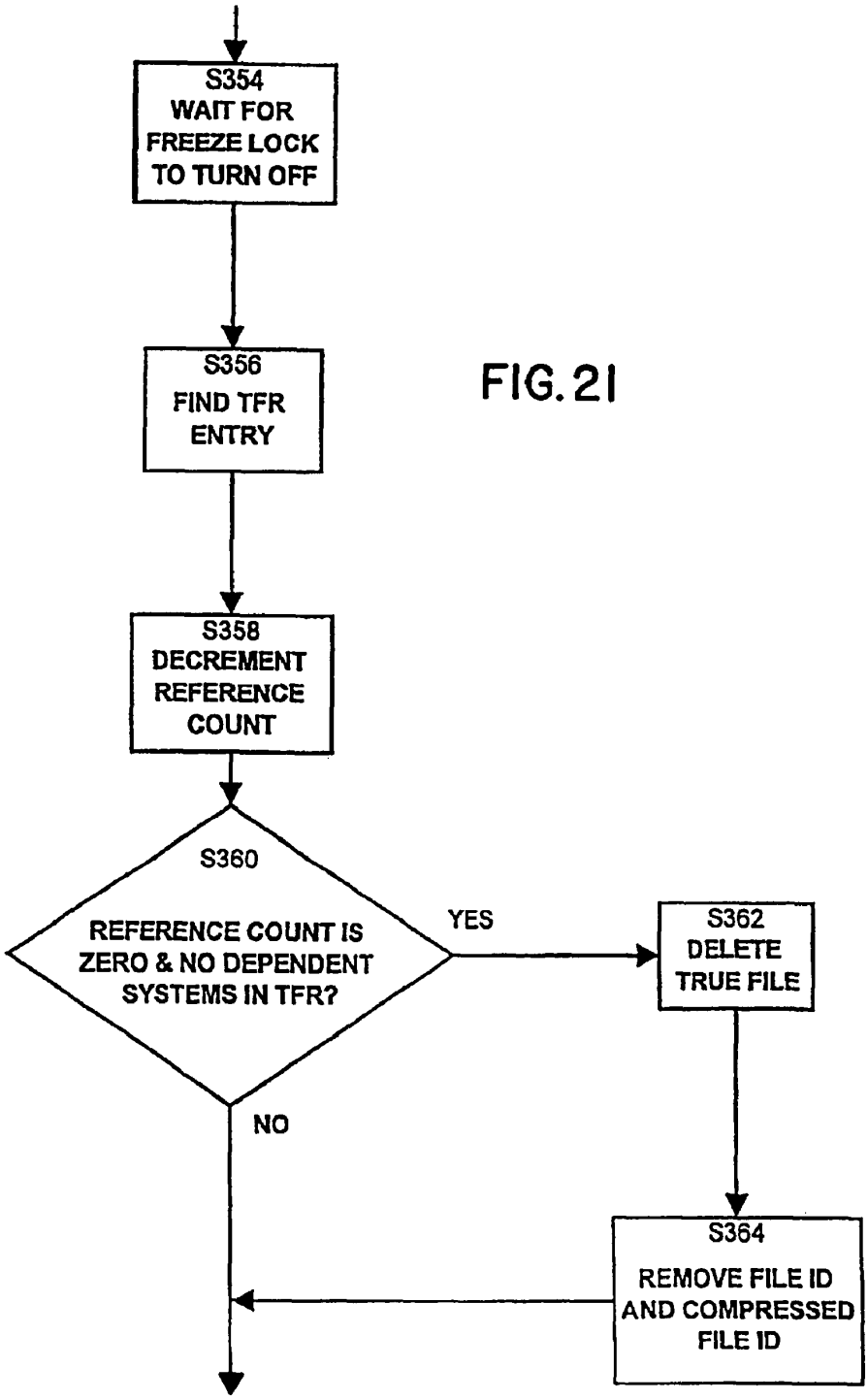

S354
WAIT FOR FREEZE LOCK TO TURN OFF
S356
FIND TFR ENTRY
S358
DECREMENT REFERENCE COUNT
S360
REFERENCE COUNT IS ZERO & NO DEPENDENT SYSTEMS IN TFR?
YES
S362
DELETE TRUE FILE
NO
S364
REMOVE FILE ID AND COMPRESSED FILE ID

FIG. 21

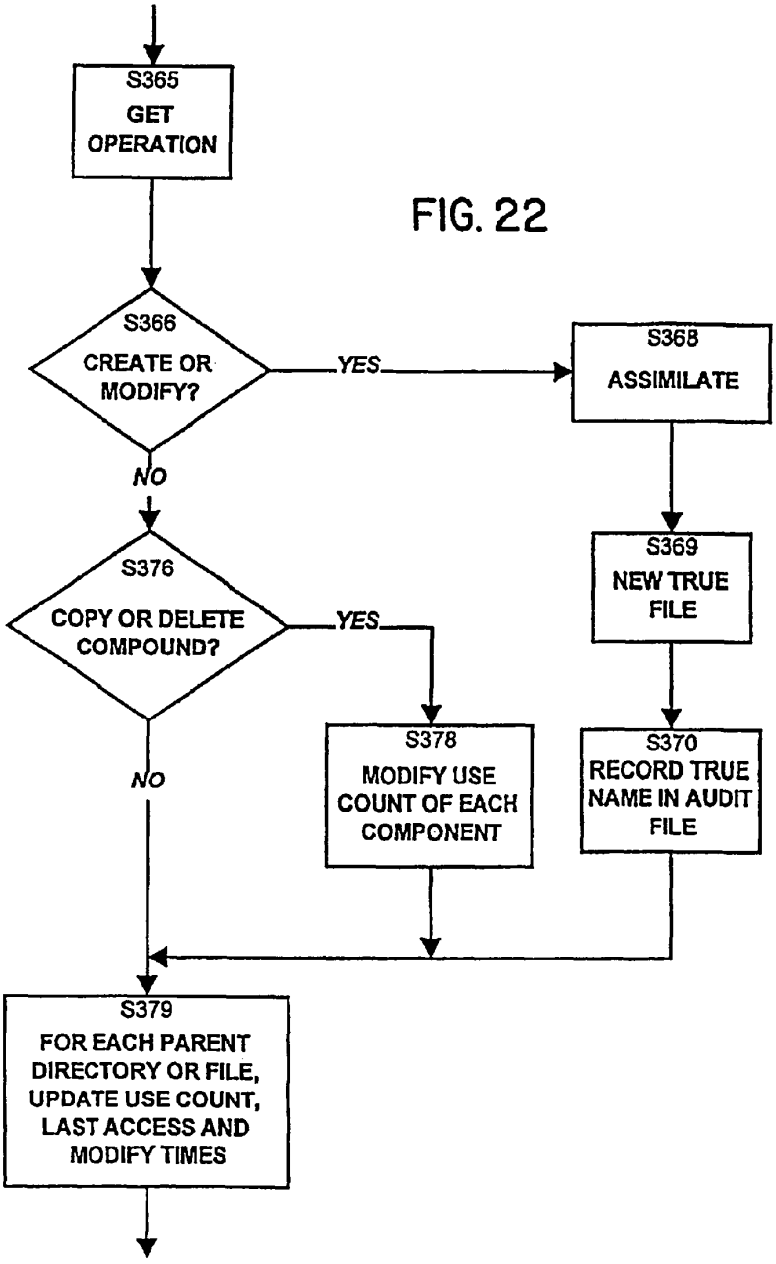
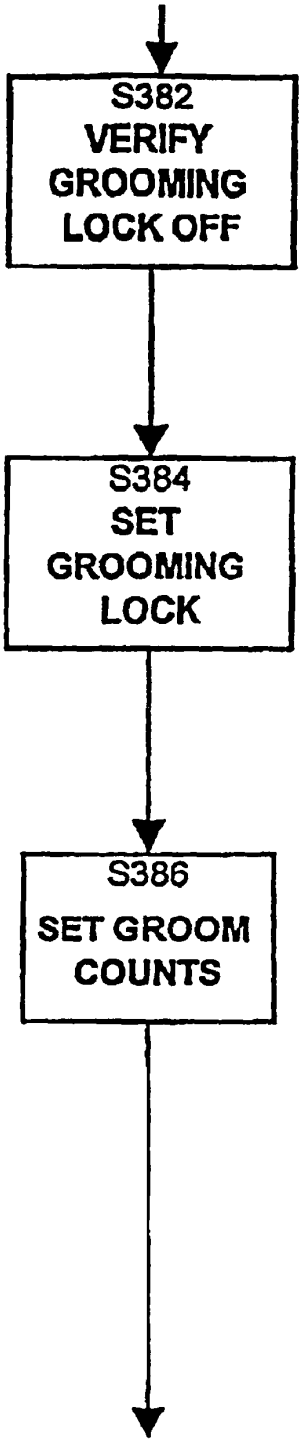

S365
GET OPERATION
S366
CREATE OR MODIFY?
YES
S368
ASSIMILATE
NO
S369
NEW TRUE FILE
S376
COPY OR DELETE COMPOUND?
YES
S378
MODIFY USE COUNT OF EACH COMPONENT
S370
RECORD TRUE NAME IN AUDIT FILE
NO
S379
FOR EACH PARENT DIRECTORY OR FILE, UPDATE USE COUNT, LAST ACCESS AND MODIFY TIMES

FIG. 22

FIG. 23


S382
VERIFY GROOMING LOCK OFF
S384
SET GROOMING LOCK
S386
SET GROOM COUNTS

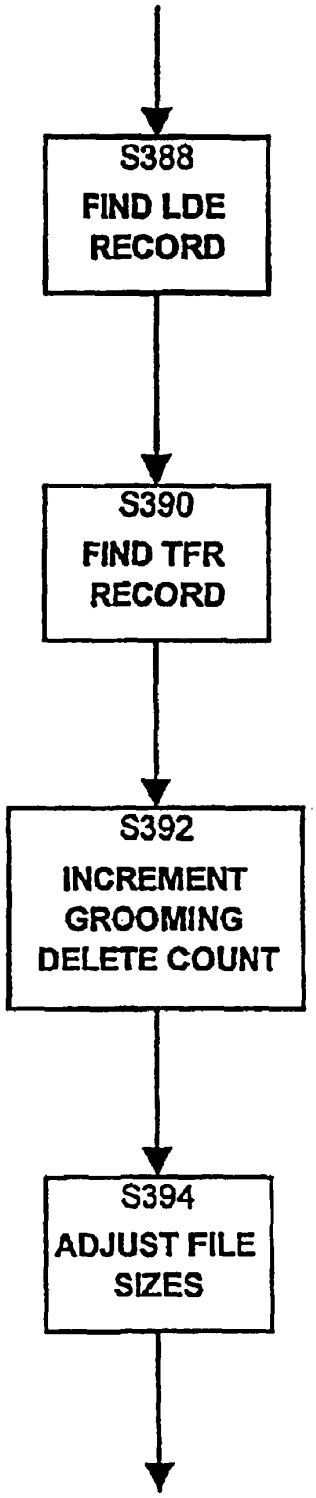

S388
FIND LDE RECORD
S390
FIND TFR RECORD
S392
INCREMENT GROOMING DELETE COUNT
S394
ADJUST FILE SIZES

FIG. 24

FIG. 25

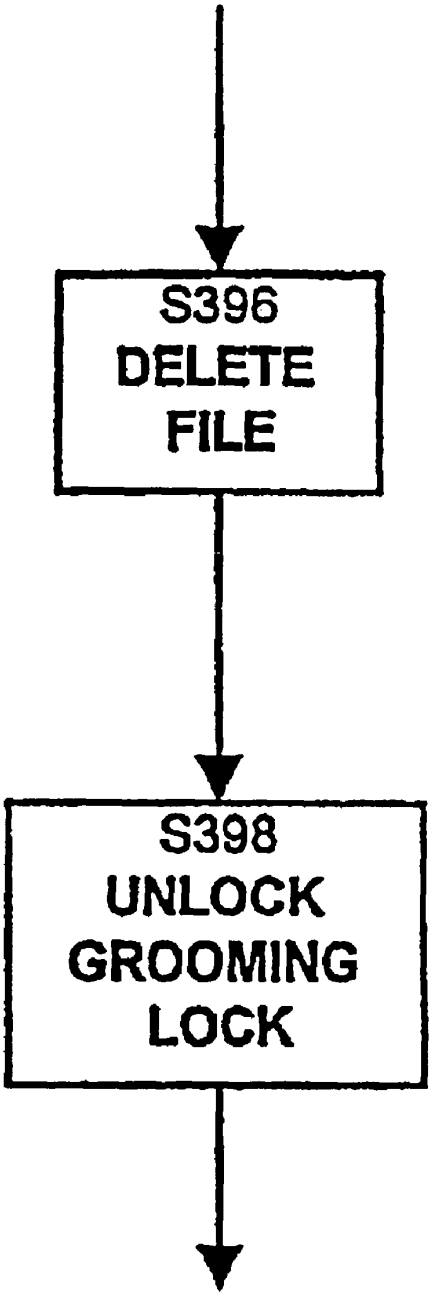
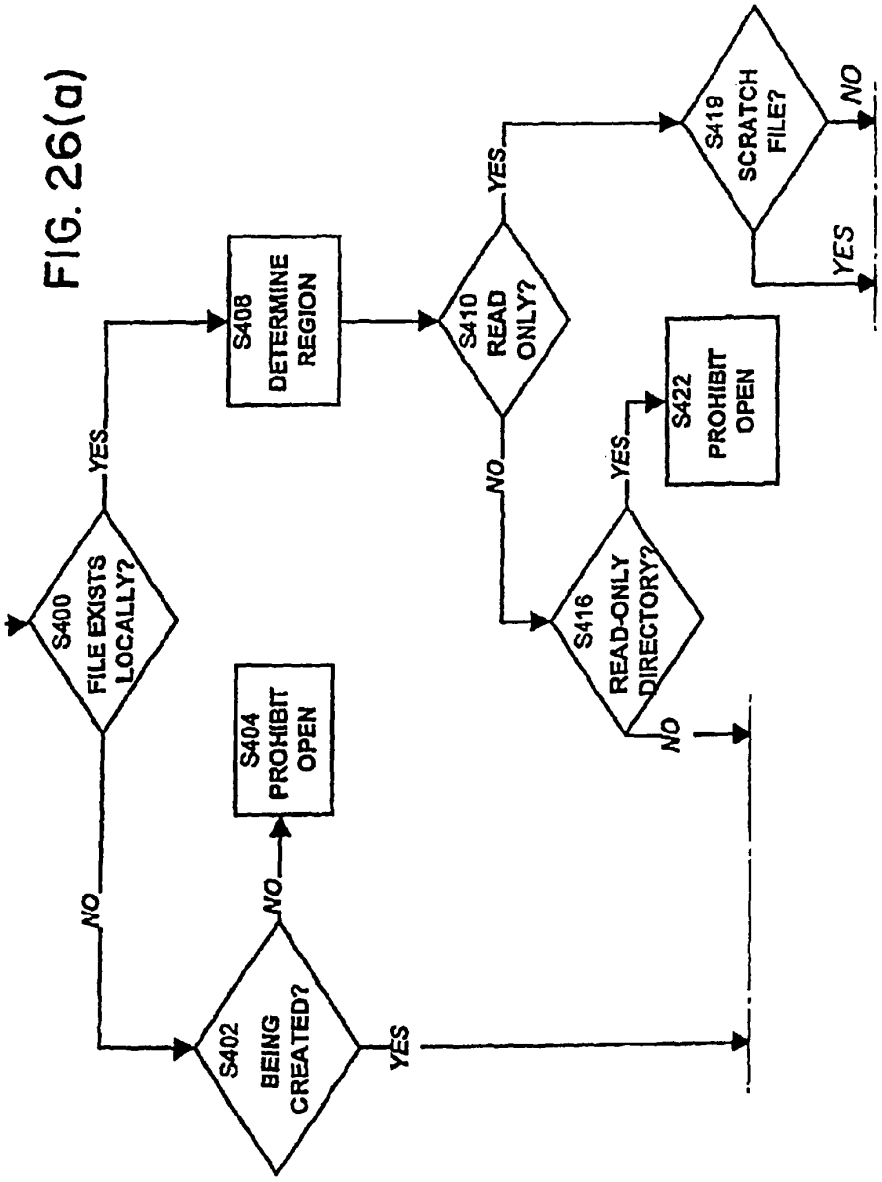


FIG. 26(a)


S400
FILE EXISTS LOCALLY?
YES
NO
S402
BEING CREATED?
YES
NO
S404
PROHIBIT OPEN
S408
DETERMINE REGION
S410
READ ONLY?
YES
NO
S416
READ-ONLY DIRECTORY?
YES
NO
S418
SCRATCH FILE?
YES
NO
S422
PROHIBIT OPEN

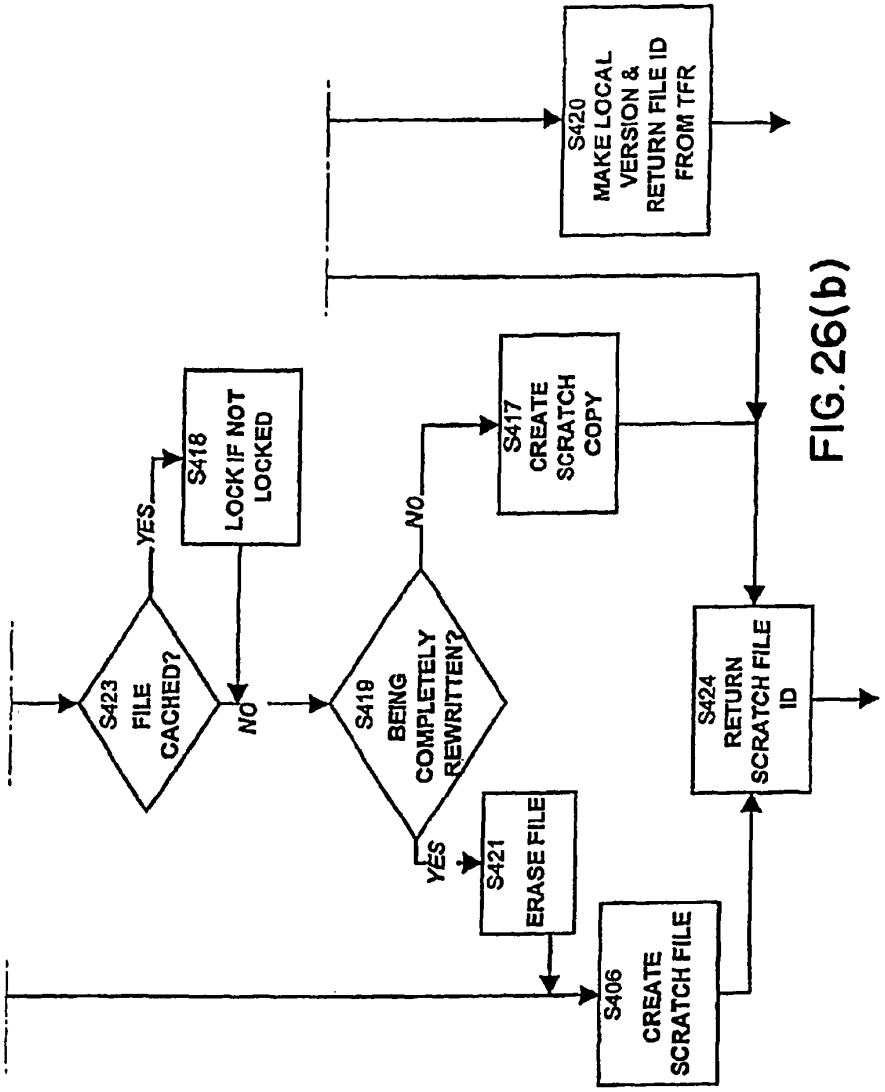

S420
MAKE LOCAL VERSION & RETURN FILE ID FROM TFR
S417
CREATE SCRATCH COPY
S418
LOCK IF NOT LOCKED
YES
NO
S423
FILE CACHED?
NO
S419
BEING COMPLETELY REWRITTEN?
S424
RETURN SCRATCH FILE ID
YES
S421
ERASE FILE
S406
CREATE SCRATCH FILE

FIG. 26(b)

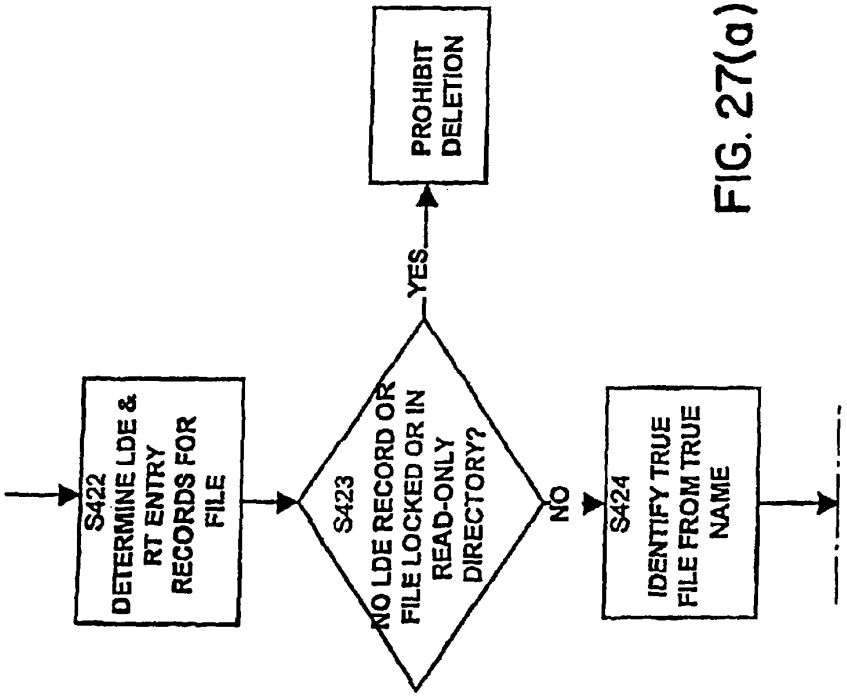

S422
DETERMINE LDE & RT ENTRY RECORDS FOR FILE
S423
NO LDE RECORD OR FILE LOCKED OR IN READ-ONLY DIRECTORY?
YES
PROHIBIT DELETION
NO
S424
IDENTIFY TRUE FILE FROM TRUE NAME

FIG. 27(a)

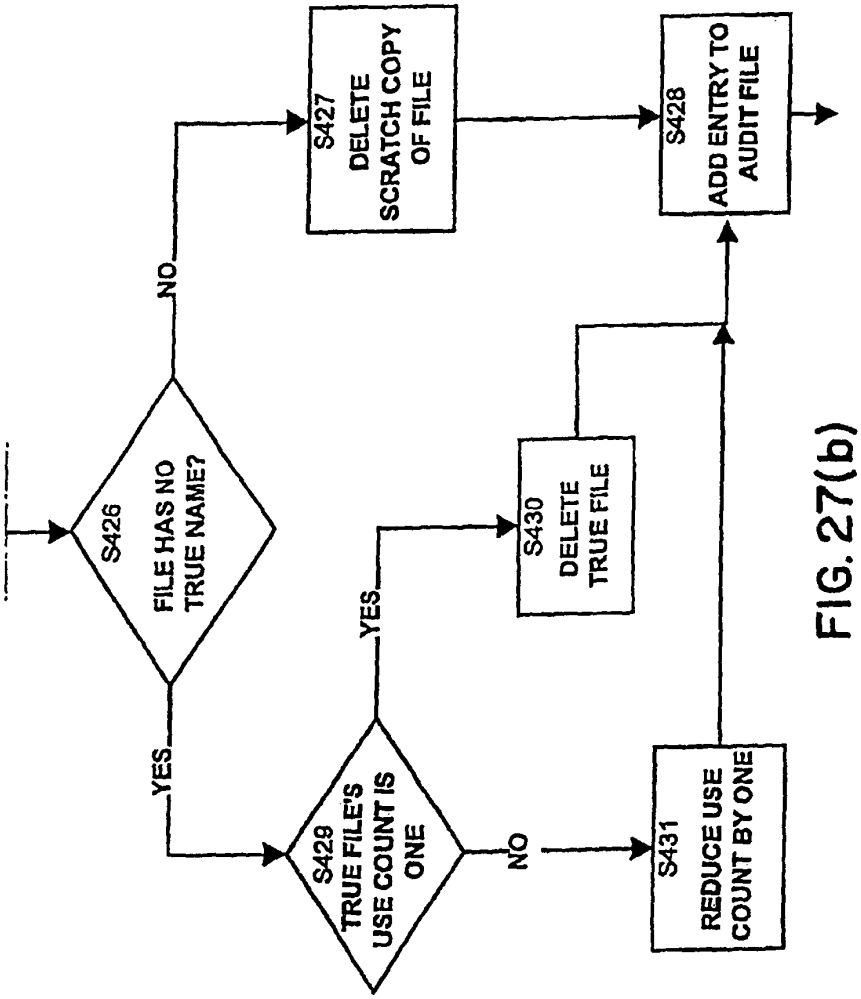
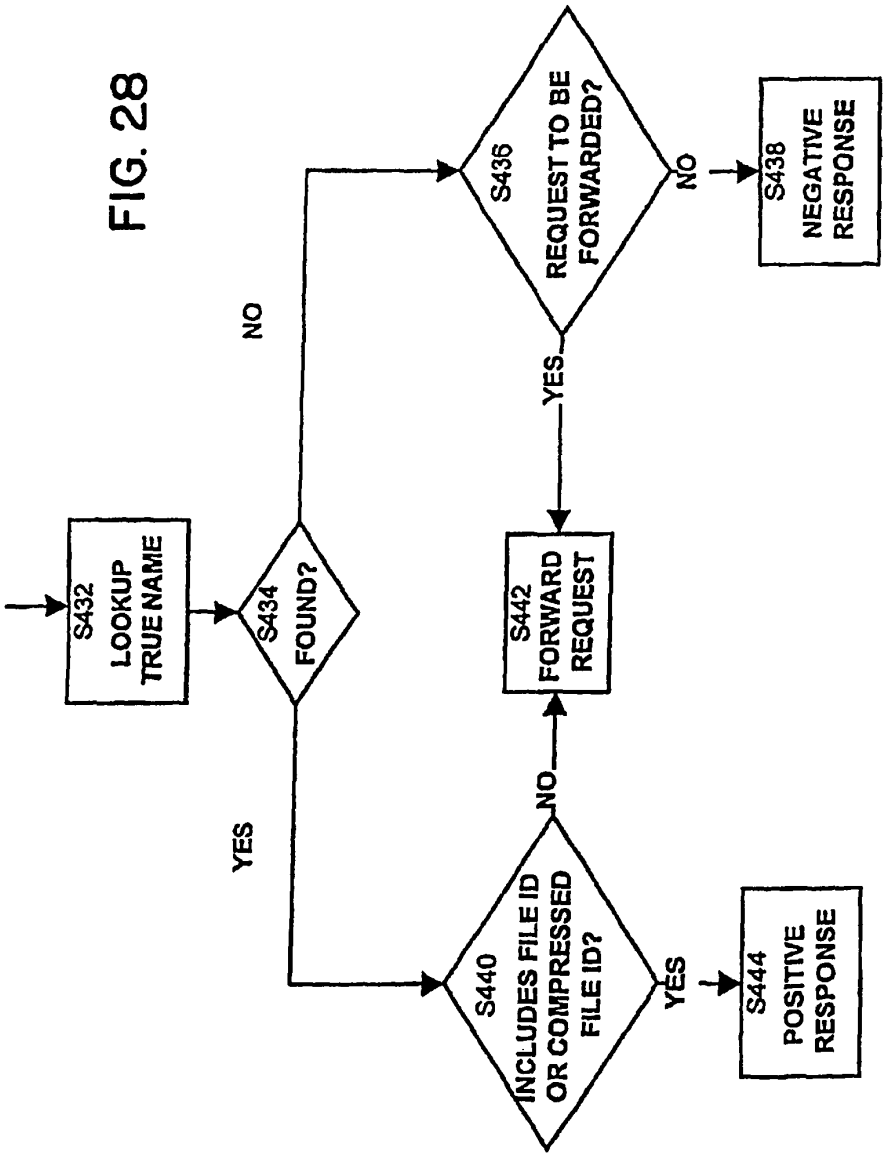

S426
FILE HAS NO TRUE NAME?
NO
YES
S427
DELETE SCRATCH COPY OF FILE
S428
ADD ENTRY TO AUDIT FILE
S429
TRUE FILE'S USE COUNT IS ONE
YES
NO
S430
DELETE TRUE FILE
S431
REDUCE USE COUNT BY ONE

FIG. 27(b)

FIG. 28


S432
LOOKUP TRUE NAME
S434
FOUND?
NO
YES
S436
REQUEST TO BE FORWARDED?
NO
S438
NEGATIVE RESPONSE
YES
S442
FORWARD REQUEST
NO
S440
INCLUDES FILE ID OR COMPRESSED FILE ID?
YES
S444
POSITIVE RESPONSE

SIMILARITY-BASED ACCESS CONTROL OF DATA IN A DATA PROCESSING SYSTEM

RELATED APPLICATIONS

This application is a continuation of and claims priority to co-pending U.S. patent application Ser. No. 11/724,232 filed Mar. 15, 2007, which is a continuation of co-pending application Ser. No. 11/017,650, filed Dec. 22, 2004, which is a continuation of pending application Ser. No. 10/742,972, filed Dec. 23, 2003, which is a continuation of Ser. No. 09/987,723, filed Nov. 15, 2001, patented as U.S. Pat. No. 6,928,442; which is a which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference. This application is a continuation of and claims priority to co-pending application Ser. No. 11/017,650, filed Dec. 22, 2004, which is a continuation of application Ser. No. 09/987,723, filed Nov. 15, 2001, now U.S. Pat. No. 6,928,442, which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference. This is also a continuation of and claims priority to co-pending application Ser. No. 10/742,972, filed Dec. 23, 2003, which is a division of application Ser. No. 09/987,723, filed Nov. 15, 2001, now U.S. Pat. No. 6,928,442, which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791, which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned, the contents of which each of these applications are hereby incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to data processing systems and, more particularly, to data processing systems wherein data items are identified by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items.

2. Background of the Invention

Data processing (DP) systems, computers, networks of computers, or the like, typically offer users and programs various ways to identify the data in the systems.

Users typically identify data in the data processing system by giving the data some form of name. For example, a typical operating system (OS) on a computer provides a file system in which data items are named by alphanumeric identifiers. Programs typically identify data in the data processing system using a location or address. For example, a program may identify a record in a file or database by using a record number which serves to locate that record.

In all but the most primitive operating systems, users and programs are able to create and use collections of named data items, these collections themselves being named by identifiers. These named collections can then, themselves, be made part of other named collections. For example, an OS may provide mechanisms to group files (data items) into directories (collections). These directories can then, themselves be made part of other directories. A data item may thus be identified relative to these nested directories using a sequence of names, or a so-called pathname, which defines a path through the directories to a particular data item (file or directory).

As another example, a database management system may group data records (data items) into tables and then group these tables into database files (collections). The complete address of any data record can then be specified using the database file name, the table name, and the record number of that data record.

Other examples of identifying data items include: identifying files in a network file system, identifying objects in an object-oriented database, identifying images in an image database, and identifying articles in a text database.

In general, the terms "data" and "data item" as used herein refer to sequences of bits. Thus a data item may be the contents of a file, a portion of a file, a page in memory, an object in an object-oriented program, a digital message, a digital scanned image, a part of a video or audio signal, or any other entity which can be represented by a sequence of bits. The term "data processing" herein refers to the processing of data items, and is sometimes dependent on the type of data item being processed. For example, a data processor for a digital image may differ from a data processor for an audio signal.

In all of the prior data processing systems the names or identifiers provided to identify data items (the data items being files, directories, records in the database, objects in object-oriented programming, locations in memory or on a physical device, or the like) are always defined relative to a specific context. For instance, the file identified by a particular file name can only be determined when the directory containing the file (the context) is known. The file identified by a pathname can be determined only when the file system (context) is known. Similarly, the addresses in a process address space, the keys in a database table, or domain names on a global computer network such as the Internet are meaningful only because they are specified relative to a context.

In prior art systems for identifying data items there is no direct relationship between the data names and the data item. The same data name in two different contexts may refer to different data items, and two different data names in the same context may refer to the same data item.

In addition, because there is no correlation between a data name and the data it refers to, there is no a priori way to confirm that a given data item is in fact the one named by a data name. For instance, in a DP system, if one processor requests that another processor deliver a data item with a given data name, the requesting processor cannot, in general, verify that the data delivered is the correct data (given only the name). Therefore it may require further processing, typically on the part of the requestor, to verify that the data item it has obtained is, in fact, the item it requested.

A common operation in a DP system is adding a new data item to the system. When a new data item is added to the system, a name can be assigned to it only by updating the context in which names are defined. Thus such systems require a centralized mechanism for the management of names. Such a mechanism is required even in a multi-processing system when data items are created and identified at separate processors in distinct locations, and in which there is no other need for communication when data items are added.

In many data processing systems or environments, data items are transferred between different locations in the system. These locations may be processors in the data processing system, storage devices, memory, or the like. For example,

one processor may obtain a data item from another processor or from an external storage device, such as a floppy disk, and may incorporate that data item into its system (using the name provided with that data item).

However, when a processor (or some location) obtains a data item from another location in the DP system, it is possible that this obtained data item is already present in the system (either at the location of the processor or at some other location accessible by the processor) and therefore a duplicate of the data item is created. This situation is common in a network data processing environment where proprietary software products are installed from floppy disks onto several processors sharing a common file server. In these systems, it is often the case that the same product will be installed on several systems, so that several copies of each file will reside on the common file server.

In some data processing systems in which several processors are connected in a network, one system is designated as a cache server to maintain master copies of data items, and other systems are designated as cache clients to copy local copies of the master data items into a local cache on an as-needed basis. Before using a cached item, a cache client must either reload the cached item, be informed of changes to the cached item, or confirm that the master item corresponding to the cached item has not changed. In other words, a cache client must synchronize its data items with those on the cache server. This synchronization may involve reloading data items onto the cache client. The need to keep the cache synchronized or reload it adds significant overhead to existing caching mechanisms.

In view of the above and other problems with prior art systems, it is therefore desirable to have a mechanism which allows each processor in a multiprocessor system to determine a common and substantially unique identifier for a data item, using only the data in the data item and not relying on any sort of context.

It is further desirable to have a mechanism for reducing multiple copies of data items in a data processing system and to have a mechanism which enables the identification of identical data items so as to reduce multiple copies. It is further desirable to determine whether two instances of a data item are in fact the same data item, and to perform various other systems' functions and applications on data items without relying on any context information or properties of the data item.

It is also desirable to provide such a mechanism in such a way as to make it transparent to users of the data processing system, and it is desirable that a single mechanism be used to address each of the problems described above.

SUMMARY OF THE INVENTION

This invention provides, in a data processing system, a method and apparatus for identifying a data item in the system, where the identity of the data item depends on all of the data in the data item and only on the data in the data item. Thus the identity of a data item is independent of its name, origin, location, address, or other information not derivable directly from the data, and depends only on the data itself.

This invention further provides an apparatus and a method for determining whether a particular data item is present in the system or at a location in the system, by examining only the data identities of a plurality of data items.

Using the method or apparatus of the present invention, the efficiency and integrity of a data processing system can be improved. The present invention improves the design and operation of a data storage system, file system, relational database, object-oriented database, or the like that stores a plurality of data items, by making possible or improving the design and operation of at least some or all of the following features:

the system stores at most one copy of any data item at a given location, even when multiple data names in the system refer to the same contents;

the system avoids copying data from source to destination locations when the destination locations already have the data;

the system provides transparent access to any data item by reference only to its identity and independent of its present location, whether it be local, remote, or offline;

the system caches data items from a server, so that only the most recently accessed data items need be retained;

when the system is being used to cache data items, problems of maintaining cache consistency are avoided;

the system maintains a desired level of redundancy of data items in a network of servers, to protect against failure by ensuring that multiple copies of the data items are present at different locations in the system;

the system automatically archives data items as they are created or modified;

the system provides the size, age, and location of groups of data items in order to decide whether they can be safely removed from a local file system;

the system can efficiently record and preserve any collection of data items;

the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items;

the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of the data items;

the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM, the inventory is independent of other properties of the data items such as their name, location, and date of creation;

the system allows closely related sets of data items, such as matching or corresponding directories on disconnected computers, to be periodically resynchronized with one another;

the system can verify that data retrieved from another location is the desired or requested data, using only the data identifier used to retrieve the data;

the system can prove possession of specific data items by content without disclosing the content of the data items, for purposes of later legal verification and to provide anonymity;

the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes.

Other objects, features, and characteristics of the present invention as well as the methods of operation and functions of the related elements of structure, and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1**(*a*) and **1**(*b*) depict a typical data processing system in which a preferred embodiment of the present invention operates;

FIG. **2** depicts a hierarchy of data items stored at any location in such a data processing system;

FIGS. **3-9** depict data structures used to implement an embodiment of the present invention; and

FIGS. **10**(*a*)-**28** are flow charts depicting operation of various aspects of the present invention.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

An embodiment of the present invention is now described with reference to a typical data processing system **100**, which, with reference to FIGS. **1**(*a*) and **1**(*b*), includes one or more processors (or computers) **102** and various storage devices **104** connected in some way, for example by a bus **106**.

Each processor **102** includes a CPU **108**, a memory **110** and one or more local storage devices **112**. The CPU **108**, memory **110**, and local storage device **112** may be internally connected, for example by a bus **114**. Each processor **102** may also include other devices (not shown), such as a keyboard, a display, a printer, and the like.

In a data processing system **100**, wherein more than one processor **102** is used, that is, in a multiprocessor system, the processors may be in one of various relationships. For example, two processors **102** may be in a client/server, client/client, or a server/server relationship. These inter-processor relationships may be dynamic, changing depending on particular situations and functions. Thus, a particular processor **102** may change its relationship to other processors as needed, essentially setting up a peer-to-peer relationship with other processors. In a peer-to-peer relationship, sometimes a particular processor **102** acts as a client processor, whereas at other times the same processor acts as a server processor. In other words, there is no hierarchy imposed on or required of processors **102**.

In a multiprocessor system, the processors **102** may be homogeneous or heterogeneous. Further, in a multiprocessor data processing system **100**, some or all of the processors **102** may be disconnected from the network of processors for periods of time. Such disconnection may be part of the normal operation of the system **100** or it may be because a particular processor **102** is in need of repair.

Within a data processing system **100**, the data may be organized to form a hierarchy of data storage elements, wherein lower level data storage elements are combined to form higher level elements. This hierarchy can consist of, for example, processors, file systems, regions, directories, data files, segments, and the like. For example, with reference to FIG. **2**, the data items on a particular processor **102** may be organized or structured as a file system **116** which comprises regions **117**, each of which comprises directories **118**, each of which can contain other directories **118** or files **120**. Each file **120** being made up of one or more data segments **122**.

In a typical data processing system, some or all of these elements can be named by users given certain implementation specific naming conventions, the name (or pathname) of an element being relative to a context. In the context of a data processing system **100**, a pathname is fully specified by a processor name, a file system name, a sequence of zero or more directory names identifying nested directories, and a final file name. (Usually the lowest level elements, in this case segments **122**, cannot be named by users.)

In other words, a file system **116** is a collection of directories **118**. A directory **118** is a collection of named files **120**—both data files **120** and other directory files **118**. A file **120** is a named data item which is either a data file (which may be simple or compound) or a directory file **118**. A simple file **120** consists of a single data segment **122**. A compound file **120** consists of a sequence of data segments **122**. A data segment **122** is a fixed sequence of bytes. An important property of any data segment is its size, the number of bytes in the sequence.

A single processor **102** may access one or more file systems **116**, and a single storage device **104** may contain one or more file systems **116**, or portions of a file system **116**. For instance, a file system **116** may span several storage devices **104**.

In order to implement controls in a file system, file system **116** may be divided into distinct regions, where each region is a unit of management and control. A region consists of a given directory **118** and is identified by the pathname (user defined) of the directory.

In the following, the term "location", with respect to a data processing system **100**, refers to any of a particular processor **102** in the system, a memory of a particular processor, a storage device, a removable storage medium (such as a floppy disk or compact disk), or any other physical location in the system. The term "local" with respect to a particular processor **102** refers to the memory and storage devices of that particular processor.

In the following, the terms "True Name", "data identity" and "data identifier" refer to the substantially unique data identifier for a particular data item. The term "True File" refers to the actual file, segment, or data item identified by a True Name.

A file system for a data processing system **100** is now described which is intended to work with an existing operating system by augmenting some of the operating system's file management system codes. The embodiment provided relies on the standard file management primitives for actually storing to and retrieving data items from disk, but uses the mechanisms of the present invention to reference and access those data items.

The processes and mechanisms (services) provided in this embodiment are grouped into the following categories: primitive mechanisms, operating system mechanisms, remote mechanisms, background mechanisms, and extended mechanisms.

Primitive mechanisms provide fundamental capabilities used to support other mechanisms. The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

Operating system mechanisms provide typical familiar file system mechanisms, while maintaining the data structures required to offer the mechanisms of the, present invention. Operating system mechanisms are designed to augment existing operating systems, and in this way to make the present invention compatible with, and generally transparent to, existing applications. The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;

4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

Remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation. The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

Background mechanisms are intended to run occasionally and at a low priority. These provide automated management capabilities with respect to the present invention. The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links; and
4. Verify Region; and
5. Groom Source List.

Extended mechanisms run within application programs over the operating system. These mechanisms provide solutions to specific problems and applications. The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at location;
7. Verify True File;
8. Track for accounting purposes; and
9. Track for licensing purposes.

The file system herein described maintains sufficient information to provide a variety of mechanisms not ordinarily offered by an operating system, some of which are listed and described here. Various processing performed by this embodiment of the present invention will now be described in greater detail.

In some embodiments, some files **120** in a data processing system **100** do not have True Names because they have been recently received or created or modified, and thus their True Names have not yet been computed. A file that does not yet have a True Name is called a scratch file. The process of assigning a True Name to a file is referred to as assimilation, and is described later. Note that a scratch file may have a user provided name.

Some of the processing performed by the present invention can take place in a background mode or on a delayed or as-needed basis. This background processing is used to determine information that is not immediately required by the system or which may never be required. As an example, in some cases a scratch file is being changed at a rate greater than the rate at which it is useful to determine its True Name. In these cases, determining the True Name of the file can be postponed or performed in the background.

Data Structures

The following data structures, stored in memory **110** of one of more processors **102** are used to implement the mechanisms described herein. The data structures can be local to each processor **102** of the system **100**, or they can reside on only some of the processors **102**.

The data structures described are assumed to reside on individual peer processors **102** in the data processing system **100**. However, they can also be shared by placing them on a remote, shared file server (for instance, in a local area network of machines). In order to accommodate sharing data structures, it is necessary that the processors accessing the shared database use the appropriate locking techniques to ensure that changes to the shared database do not interfere with one another but are appropriately serialized. These locking techniques are well understood by ordinarily skilled programmers of distributed applications.

It is sometimes desirable to allow some regions to be local to a particular processor **102** and other regions to be shared among processors **102**. (Recall that a region is a unit of file system management and control consisting of a given directory identified by the pathname of the directory.) In the case of local and shared regions, there would be both local and shared versions of each data structure. Simple changes to the processes described below must be made to ensure that appropriate data structures are selected for a given operation.

The local directory extensions (LDE) table **124** is a data structure which provides information about files **120** and directories **118** in the data processing system **100**. The local directory extensions table **124** is indexed by a pathname or contextual name (that is, a user provided name) of a file and includes the True Name for most files. The information in local directory extension table **124** is in addition to that provided by the native file system of the operating system.

The True File registry (TFR) **126** is a data store for listing actual data items which have True Names, both files **120** and segments **122**. When such data items occur in the True File registry **126** they are known as True Files. True Files are identified in True File registry **126** by their True Names or identities. The table True File registry **126** also stores location, dependency, and migration information about True Files.

The region table (RT) **128** defines areas in the network storage which are to be managed separately. Region table **128** defines the rules for access to and migration of files **120** among various regions with the local file system **116** and remote peer file systems.

The source table (ST) **130** is a list of the sources of True Files other than the current True File registry **126**. The source table **130** includes removable volumes and remote processors.

The audit file (AF) **132** is a list of records indicating changes to be made in local or remote files, these changes to be processed in background.

The accounting log (AL) **134** is a log of file transactions used to create accounting information in a manner which preserves the identity of files being tracked independent of their name or location.

The license table (LT) **136** is a table identifying files, which may only be used by licensed users, in a manner independent of their name or location, and the users licensed to use them.

Detailed Descriptions of the Data Structures

The following table summarizes the fields of an local directory extensions table entry, as illustrated by record **138** in FIG. **3**.

Field	Description
Region ID	identifies the region in which this file is contained.
Pathname	the user provided name or contextual name of the file or directory, relative to the region in which it occurs.
True Name	the computed True Name or identity of the file or directory. This True Name is not always up to date, and it is set to a special value when a file is modified and is later recomputed in the background.
Type	indicates whether the file is a data file or a directory.
Scratch File ID	the physical location of the file in the file system, when no True Name has been calculated for the file. As noted above, such a file is called a scratch file.
Time of last access	the last access time to this file. If this file is a directory, this is the last access time to any file in the directory.
Time of last modification	the time of last change of this file. If this file is a directory, this is the last modification time of any file in the directory.
Safe flag	indicates that this file (and, if this file is a directory, all of its subordinate files) have been backed up on some other system, and it is therefore safe to remove them.
Lock flag	indicates whether a file is locked, that is, it is being modified by the local processor or a remote processor. Only one processor may modify a file at a time.
Size	the full size of this directory (including all subordinate files), if all files in it were fully expanded and duplicated. For a file that is not a directory this is the size of the actual True File.
Owner	the identity of the user who owns this file, for accounting and license tracking purposes.

Each record of the True File registry **126** has the fields shown in the True File registry record **140** in FIG. **4**. The True File registry **126** consists of the database described in the table below as well as the actual True Files identified by the True File IDs below.

Field	Description
True Name	computed True Name or identity of the file.
Compressed File ID	compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.
Grooming delete count	tentative count of how many references have been selected for deletion during a grooming operation.
Time of last access	most recent date and time the content of this file was accessed.
Expiration	date and time after which this file may be deleted by this server.
Dependent processors	processor IDs of other processors which contain references to this True File.
Source IDs	source ID(s) of zero or more sources from which this file or data item may be retrieved.
True File ID	identity or disk location of the actual physical representation of the file or file segment. It is sufficient to use a filename in the registration directory of the underlying operating system. The True File ID is absent if cthe actual file is not currently present at the urrent location.
Use count	number of other records on this processor which identify this True File.

A region table **128**, specified by a directory pathname, records storage policies which allow files in the file system to be stored, accessed and migrated in different ways. Storage policies are programmed in a configurable way using a set of rules described below.

Each region table record **142** of region table **128** includes the fields described in the following table (with reference to FIG. **5**):

Field	Description
Region ID	internally used identifier for this region.
Region file system	file system on the local processor of which this region is a part.
Region pathname	a pathname relative to the region file system which defines the location of this region. The region consists of all files and directories subordinate to this pathname, except those in a region subordinate to this region.
Mirror processor(s)	zero or more identifiers of processors which are to keep mirror or archival copies of all files in the current region. Multiple mirror processors can be defined to form a mirror group.
Mirror duplication count	number of copies of each file in this region that should be retained in a mirror group.
Region status	specifies whether this region is local to a single processor 102, shared by several processors 102 (if, for instance, it resides on a shared file server), or managed by a remote processor.
Policy	the migration policy to apply to this region. A single region might participate in several policies. The policies are as follows (parameters in brackets are specified as part of the policy): region is a cached version from [processor ID]; region is a member of a mirror set defined by [processor ID]. region is to be archived on [processor ID]. region is to be backed up locally, by placing new copies in [region ID]. region is read only and may not be changed. region is published and expires on [date]. Files in this region should be compressed.

A source table **130** identifies a source location for True Files. The source table **130** is also used to identify client processors making reservations on the current processor. Each source record **144** of the source table **130** includes the fields summarized in the following table, with reference to FIG. **6**:

Field	Description
source ID	internal identifier used to identify a particular source.
source type	type of source location: Removable Storage Volume Local Region Cache Server Mirror Group Server Cooperative Server Publishing Server Client
source rights	includes information about the rights of this processor, such as whether it can ask the local processor to store data items for it.
source availability	measurement of the bandwidth, cost, and reliability of the connection to this source of True Files. The availability is used to select from among several possible sources.
source location	information on how the local processor is to access the source. This may be, for example, the name of a removable storage volume, or the processor ID and region path of a region on a remote processor.

The audit file **132** is a table of events ordered by timestamp, each record **146** in audit file **132** including the fields summarized in the following table (with reference to FIG. **7**):

Field	Description
Original Name	path of the file in question.
Operation	whether the file was created, read, written, copied or deleted.
Type	specifies whether the source is a file or a directory.
Processor ID	ID of the remote processor generating this event (if not local).
Timestamp	time and date file was closed (required only for accessed/modified files).
Pathname	Name of the file (required only for rename).
True Name	computed True Name of the file. This is used by remote systems to mirror changes to the directory and is filled in during background processing.

Each record **148** of the accounting log **134** records an event which may later be used to provide information for billing mechanisms. Each accounting log entry record **148** includes at least the information summarized in the following table, with reference to FIG. **8**:

Field	Description
date of entry	date and time of this log entry.
type of entry	Entry types include create file, delete file, and transmit file.
True Name	True Name of data item in question.
owner	identity of the user responsible for this action.

Each record **150** of the license table **136** records a relationship between a licensable data item and the user licensed to have access to it. Each license table record **150** includes the information summarized in the following table, with reference to FIG. **9**:

Field	Description
True Name	True Name of a data item subject to license validation.
licensee	identity of a user authorized to have access to this object.

Various other data structures are employed on some or all of the processors **102** in the data processing system **100**. Each processor **102** has a global freeze lock (GFL) **152** (FIG. **1**), which is used to prevent synchronization errors when a directory is frozen or copied. Any processor **102** may include a special archive directory (SAD) **154** into which directories may be copied for the purposes of archival. Any processor **102** may include a special media directory (SMD) **156**, into which the directories of removable volumes are stored to form a media inventory. Each processor has a grooming lock **158**, which is set during a grooming operation. During this period the grooming delete count of True File registry entries **140** is active, and no True Files should be deleted until grooming is complete. While grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

Primitive Mechanisms

The first of the mechanisms provided by the present invention, primitive mechanisms, are now described. The mechanisms described here depend on underlying data management mechanisms to create, copy, read, and delete data items in the True File registry **126**, as identified by a True File ID. This support may be provided by an underlying operating system or disk storage manager.

The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link Path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

1. Calculate True Name

A True Name is computed using a function, MD, which reduces a data block B of arbitrary length to a relatively small, fixed size identifier, the True Name of the data block, such that the True Name of the data block is virtually guaranteed to represent the data block B and only data block B.

The function MD must have the following properties:

1. The domain of the function MD is the set of all data items. The range of the function MD is the set of True Names.
2. The function MD must take a data item of arbitrary length and reduce it to an integer value in the range 0 to N−1, where N is the cardinality of the set of True Names. That is, for an arbitrary length data block B, $0 \leqq MD(B) < N$.
3. The results of MD(B) must be evenly and randomly distributed over the range of N, in such a way that simple or regular changes to B are virtually guaranteed to produce a different value of MD(B).
4. It must be computationally difficult to find a different value B' such that MD(B)=MD(B').
5. The function MD(B) must be efficiently computed.

A family of functions with the above properties are the so-called message digest functions, which are used in digital security systems as techniques for authentification of data. These functions (or algorithms) include MD4, MD5, and SHA.

In the presently preferred embodiments, either MD5 or SHA is employed as the basis for the computation of True Names. Whichever of these two message digest functions is employed, that same function must be employed on a system-wide basis.

It is impossible to define a function having a unique output for each possible input when the number of elements in the range of the function is smaller than the number of elements in its domain. However, a crucial observation is that the actual data items that will be encountered in the operation of any system embodying this invention form a very sparse subset of all the possible inputs.

A colliding set of data items is defined as a set wherein, for one or more pairs x and y in the set, MD(x)=MD(y). Since a function conforming to the requirements for MD must evenly and randomly distribute its outputs, it is possible, by making the range of the function large enough, to make the probability arbitrarily small that actual inputs encountered in the operation of an embodiment of this invention will form a colliding set.

To roughly quantify the probability of a collision, assume that there are no more than 2^{30} storage devices in the world, and that each storage device has an average of at most 2^{20}

different data items. Then there are at most 2^{50} data items in the world. If the outputs of MD range between 0 and 2^{128}, it can be demonstrated that the probability of a collision is approximately 1 in 2^{29}. Details on the derivation of these probability values are found, for example, in P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," *Lecture Notes in Computer Science* 434: *Advances in Cryptology—Eurocrypt '89 Proceedings*, Springer-Verlag, pp. 329-354.

Note that for some less preferred embodiments of the present invention, lower probabilities of uniqueness may be acceptable, depending on the types of applications and mechanisms used. In some embodiments it may also be useful to have more than one level of True Names, with some of the True Names having different degrees of uniqueness. If such a scheme is implemented, it is necessary to ensure that less unique True Names are not propagated in the system.

While the invention is described herein using only the True Name of a data item as the identifier for the data item, other preferred embodiments use tagged, typed, categorized or classified data items and use a combination of both the True Name and the tag, type, category or class of the data item as an identifier. Examples of such categorizations are files, directories, and segments; executable files and data files, and the like. Examples of classes are classes of objects in an object-oriented system. In such a system, a lower degree of True Name uniqueness is acceptable over the entire universe of data items, as long as sufficient uniqueness. is provided per category of data items. This is because the tags provide an additional level of uniqueness.

A mechanism for calculating a True Name given a data item is now described, with reference to FIGS. **10**(*a*) and **10**(*b*).

A simple data item is a data item whose size is less than a particular given size (which must be defined in each particular implementation of the invention). To determine the True Name of a simple data item, with reference to FIG. **10**(*a*), first compute the MD function (described above) on the given simple data item (Step S**212**). Then append to the resulting 128 bits, the byte length modulo 32 of the data item (Step S**214**). The resulting 160-bit value is the True Name of the simple data item.

A compound data item is one whose size is greater than the particular given size of a simple data item. To determine the True Name of an arbitrary (simple or compound) data item, with reference to FIG. **10**(*b*), first determine if the data item is a simple or a compound data item (Step S**216**). If the data item is a simple data item, then compute its True Name in step S**218** (using steps S**212** and S**214** described above), otherwise partition the data item into segments (Step S**220**) and assimilate each segment (Step S**222**) (the primitive mechanism, Assimilate a Data Item, is described below), computing the True Name of the segment. Then create an indirect block consisting of the computed segment True Names (Step S**224**). An indirect block is a data item which consists of the sequence of True Names of the segments. Then, in step S**226**, assimilate the indirect block and compute its True Name. Finally, replace the final thirty-two (32) bits of the resulting True Name (that is, the length of the indirect block) by the length modulo 32 of the compound data item (Step S**228**). The result is the True Name of the compound data item.

Note that the compound data item may be so large that the indirect block of segment True Names is itself a compound data item. In this case the mechanism is invoked recursively until only simple data items are being processed.

Both the use of segments and the attachment of a length to the True Name are not strictly required in a system using the present invention, but are currently considered desirable features in the preferred embodiment.

2. Assimilate Data Item

A mechanism for assimilating a data item (scratch file or segment) into a file system, given the scratch file ID of the data item, is now described with reference to FIG. **11**. The purpose of this mechanism is to add a given data item to the True File registry **126**. If the data item already exists in the True File registry **126**, this will be discovered and used during this process, and the duplicate will be eliminated.

Thereby the system stores at most one copy of any data item or file by content, even when multiple names refer to the same content.

First, determine the True Name of the data item corresponding to the given scratch File ID using the Calculate True Name primitive mechanism (Step S**230**). Next, look for an entry for the True Name in the True File registry **126** (Step S**232**) and determine whether a True Name entry, record **140**, exists in the True File registry **126**. If the entry record includes a corresponding True File ID or compressed File ID (Step S**237**), delete the file with the scratch File ID (Step S**238**). Otherwise store the given True File ID in the entry record (step S**239**).

If it is determined (in step S**232**) that no True Name entry exists in the True File registry **126**, then, in Step S**236**, create a new entry in the True File registry **126** for this True Name. Set the True Name of the entry to the calculated True Name, set the use count for the new entry to one, store the given True File ID in the entry and set the other fields of the entry as appropriate.

Because this procedure may take some time to compute, it is intended to run in background after a file has ceased to change. In the meantime, the file is considered an unassimilated scratch file.

3. New True File

The New True File process is invoked when processing the audit file **132**, some time after a True File has been assimilated (using the Assimilate Data Item primitive mechanism). Given a local directory extensions table entry record **138** in the local directory extensions table **124**, the New True File process can provide the following steps (with reference to FIG. **12**), depending on how the local processor is configured:

First, in step S**238**, examine the local directory extensions table entry record **138** to determine whether the file is locked by a cache server. If the file is locked, then add the ID of the cache server to the dependent processor list of the True File registry table **126**, and then send a message to the cache server to update the cache of the current processor using the Update Cache remote mechanism (Step **242**).

If desired, compress the True File (Step S**246**), and, if desired, mirror the True File using the Mirror True File background mechanism (Step S**248**).

4. Get True Name from Path

The True Name of a file can be used to identify a file by contents, to confirm that a file matches its original contents, or to compare two files. The mechanism to get a True Name given the pathname of a file is now described with reference to FIG. **13**.

First, search the local directory extensions table **124** for the entry record **138** with the given pathname (Step S**250**). If the pathname is not found, this process fails and no True Name corresponding to the given pathname exists. Next, determine whether the local directory extensions table entry record **138** includes a True Name (Step S**252**), and if so, the mechanism's task is complete. Otherwise, determine whether the local directory extensions table entry record **138** identifies a direc-

tory (Step S**254**), and if so, freeze the directory (Step S**256**) (the primitive mechanism Freeze Directory is described below).

Otherwise, in step S**258**, assimilate the file (using the Assimilate Data Item primitive mechanism) defined by the File ID field to generate its True Name and store its True Name in the local directory extensions entry record. Then return the True Name identified by the local directory extensions table **124**.

5. Link Path to True Name

The mechanism to link a path to a True Name provides a way of creating a new directory entry record identifying an existing, assimilated file. This basic process may be used to copy, move, and rename files without a need to copy their contents. The mechanism to link a path to a True Name is now described with reference to FIG. **14**.

First, if desired, confirm that the True Name exists locally by searching for it in the True Name registry or local directory extensions table **135** (Step S**260**). Most uses of this mechanism will require this form of validation. Next, search for the path in the local directory extensions table **135** (Step S**262**). Confirm that the directory containing the file named in the path already exists (Step S**264**). If the named file itself exists, delete the File using the Delete True File operating system mechanism (see below) (Step S**268**).

Then, create an entry record in the local directory extensions with the specified path (Step S**270**) and update the entry record and other data structures as follows: fill in the True Name field of the entry with the specified True Name; increment the use count for the True File registry entry record **140** of the corresponding True Name; note whether the entry is a directory by reading the True File to see if it contains a tag (magic number) indicating that it represents a frozen directory (see also the description of the Freeze Directory primitive mechanism regarding the tag); and compute and set the other fields of the local directory extensions appropriately. For instance, search the region table **128** to identify the region of the path, and set the time of last access and time of last modification to the current time.

6. Realize True File from Location

This mechanism is used to try to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. This mechanism is now described with reference to FIG. **15**.

First, in step S**272**, determine whether the location specified is a processor. If it is determined that the location specified is a processor, then send a Request True File message (using the Request True File remote mechanism) to the remote processor and wait for a response (Step S**274**). If a negative response is received or no response is received after a timeout period, this mechanism fails. If a positive response is received, enter the True File returned in the True File registry **126** (Step S**276**). (If the file received was compressed, enter the True File ID in the compressed File ID field.)

If, on the other hand, it is determined in step S**272** that the location specified is not a processor, then, if necessary, request the user or operator to mount the indicated volume (Step S**278**). Then (Step S**280**) find the indicated file on the given volume and assimilate the file using the Assimilate Data Item primitive mechanism. If the volume does not contain a True File registry **126**, search the media inventory to find the path of the file on the volume. If no such file can be found, this mechanism fails.

At this point, whether or not the location is determined (in step S**272**) to be a processor, if desired, verify the True File (in step S**282**).

7. Locate Remote File

This mechanism allows a processor to locate a file or data item from a remote source of True Files, when a specific source is unknown or unavailable. A client processor system may ask one of several or many sources whether it can supply a data object with a given True Name. The steps to perform this mechanism are as follows (with reference to FIGS. **16**(*a*) and **16**(*b*)).

The client processor **102** uses the source table **145** to select one or more source processors (Step S**284**). If no source processor can be found, the mechanism fails. Next, the client processor **102** broadcasts to the selected sources a request to locate the file with the given True Name using the Locate True File remote mechanism (Step S**286**). The request to locate may be augmented by asking to propagate this request to distant servers. The client processor then waits for one or more servers to respond positively (Step S**288**). After all servers respond negatively, or after a timeout period with no positive response, the mechanism repeats selection (Step S**284**) to attempt to identify alternative sources. If any selected source processor responds, its processor ID is the result of this mechanism. Store the processor ID in the source field of the True File registry entry record **140** of the given True Name (Step S**290**).

If the source location of the True Name is a different processor or medium than the destination (Step S**290***a*), perform the following steps:

(i) Look up the True File registry entry record **140** for the corresponding True Name, and add the source location ID to the list of sources for the True Name (Step S**290***b*); and

(ii) If the source is a publishing system, determine the expiration date on the publishing system for the True Name and add that to the list of sources. If the source is not a publishing system, send a message to reserve the True File on the source processor (Step S**290***c*).

Source selection in step S**284** may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost, and ignoring previously selected processors which did not respond in step S**288**.

8. Make True File Local

This mechanism is used when a True Name is known and a locally accessible copy of the corresponding file or data item is required. This mechanism makes it possible to actually read the data in a True File. The mechanism takes a True Name and returns when there is a local, accessible copy of the True File in the True File registry **126**. This mechanism is described here with reference to the flow chart of FIGS. **17**(*a*) and **17**(*b*).

First, look in the True File registry **126** for a True File entry record **140** for the corresponding True Name (Step S**292**). If no such entry is found this mechanism fails. If there is already a True File ID for the entry (Step S**294**), this mechanism's task is complete. If there is a compressed file ID for the entry (Step S**296**), decompress the file corresponding to the file ID (Step S**298**) and store the decompressed file ID in the entry (Step S**300**). This mechanism is then complete.

If there is no True File ID for the entry (Step S**294**) and there is no compressed file ID for the entry (Step S**296**), then continue searching for the requested file. At this time it may be necessary to notify the user that the system is searching for the requested file.

If there are one or more source IDs, then select an order in which to attempt to realize the source ID (Step S**304**). The order may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost. For each source in the order chosen, realize the True File from the source location (using the Realize True File

from Location primitive mechanism), until the True File is realized (Step S306). If it is realized, continue with step S294. If no known source can realize the True File, use the Locate Remote File primitive mechanism to attempt to find the True File (Step S308). If this succeeds, realize the True File from the identified source location and continue with step S296.

9. Create Scratch File

A scratch copy of a file is required when a file is being created or is about to be modified. The scratch copy is stored in the file system of the underlying operating system. The scratch copy is eventually assimilated when the audit file record entry **146** is processed by the Process Audit File Entry primitive mechanism. This Create Scratch File mechanism requires a local directory extensions table entry record **138**. When it succeeds, the local directory extensions table entry record **138** contains the scratch file ID of a scratch file that is not contained in the True File registry **126** and that may be modified. This mechanism is now described with reference to FIGS. **18**(*a*) and **18**(*b*).

First determine whether the scratch file should be a copy of the existing True File (Step S310). If so, continue with step S312. Otherwise, determine whether the local directory extensions table entry record **138** identifies an existing True File (Step S316), and if so, delete the True File using the Delete True File primitive mechanism (Step S318). Then create a new, empty scratch file and store its scratch file ID in the local directory extensions table entry record **138** (step S320). This mechanism is then complete.

If the local directory extensions table entry record **138** identifies a scratch file ID (Step S312), then the entry already has a scratch file, so this mechanism succeeds.

If the local directory extensions table entry record **138** identifies a True File (S316), and there is no True File ID for the True File (S312), then make the True File local using the Make True File Local primitive mechanism (Step S322). If there is still no True File ID, this mechanism fails.

There is now a local True File for this file. If the use count in the corresponding True File registry entry record **140** is one (Step S326), save the True File ID in the scratch file ID of the local directory extensions table entry record **138**, and remove the True File registry entry record **140** (Step S328). (This step makes the True File into a scratch file.) This mechanism's task is complete.

Otherwise, if the use count in the corresponding True File registry entry record **140** is not one (in step S326), copy the file with the given True File ID to a new scratch file, using the Read File OS mechanism and store its file ID in the local directory extensions table entry record **138** (Step S330), and reduce the use count for the True File by one. If there is insufficient space to make a copy, this mechanism fails.

10. Freeze Directory

This mechanism freezes a directory in order to calculate its True Name. Since the True Name of a directory is a function of the files within the directory, they must not change during the computation of the True Name of the directory. This mechanism requires the pathname of a directory to freeze. This mechanism is described with reference to FIGS. **19**(*a*) and **19**(*b*).

In step S332, add one to the global freeze lock. Then search the local directory extensions table **124** to find each subordinate data file and directory of the given directory, and freeze each subordinate directory found using the Freeze Directory primitive mechanism (Step S334). Assimilate each unassimilated data file in the directory using the Assimilate Data Item primitive mechanism (Step S336). Then create a data item which begins with a tag or marker (a "magic number") being a unique data item indicating that this data item is a frozen directory (Step S337). Then list the file name and True Name for each file in the current directory (Step S338). Record any additional information required, such as the type, time of last access and modification, and size (Step S340). Next, in step S342, using the Assimilate Data Item primitive mechanism, assimilate the data item created in step S338. The resulting True Name is the True Name of the frozen directory. Finally, subtract one from the global freeze lock (Step S344).

11. Expand Frozen Directory

This mechanism expands a frozen directory in a given location. It requires a given pathname into which to expand the directory, and the True Name of the directory and is described with reference to FIG. **20**.

First, in step S346, make the True File with the given True Name local using the Make True File Local primitive mechanism. Then read each directory entry in the local file created in step S346 (Step S348). For each such directory entry, do the following:

Create a full pathname using the given pathname and the file name of the entry (Step S350); and

link the created path to the True Name (Step S352) using the Link Path to True Name primitive mechanism.

12. Delete True File

This mechanism deletes a reference to a True Name. The underlying True File is not removed from the True File registry **126** unless there are no additional references to the file. With reference to FIG. **21**, this mechanism is performed as follows:

If the global freeze lock is on, wait until the global freeze lock is turned off (Step S354). This prevents deleting a True File while a directory which might refer to it is being frozen. Next, find the True File registry entry record **140** given the True Name (Step S356). If the reference count field of the True File registry **126** is greater than zero, subtract one from the reference count field (Step S358). If it is determined (in step S360) that the reference count field of the True File registry entry record **140** is zero, and if there are no dependent systems listed in the True File registry entry record **140**, then perform the following steps:

(i) If the True File is a simple data item, then delete the True File, otherwise,

(ii) (the True File is a compound data item) for each True Name in the data item, recursively delete the True File corresponding to the True Name (Step S362).

(iii) Remove the file indicated by the True File ID and compressed file ID from the True File registry **126**, and remove the True File registry entry record **140** (Step S364).

13. Process Audit File Entry

This mechanism performs tasks which are required to maintain information in the local directory extensions table **124** and True File registry **126**, but which can be delayed while the processor is busy doing more time-critical tasks. Entries **142** in the audit file **132** should be processed at a background priority as long as there are entries to be processed. With reference to FIG. **22**, the steps for processing an entry are as follows:

Determine the operation in the entry **142** currently being processed (Step S365). If the operation indicates that a file was created or written (Step S366), then assimilate the file using the Assimilate Data Item primitive mechanism (Step S368), use the New True File primitive mechanism to do additional desired processing (such as cache update, compression, and mirroring) (Step S369), and record the newly computed True Name for the file in the audit file record entry (Step S370).

Otherwise, if the entry being processed indicates that a compound data item or directory was copied (or deleted)

(Step S376), then for each component True Name in the compound data item or directory, add (or subtract) one to the use count of the True File registry entry record **140** corresponding to the component True Name (Step S378).

In all cases, for each parent directory of the given file, update the size, time of last access, and time of last modification, according to the operation in the audit record (Step S379).

Note that the audit record is not removed after processing, but is retained for some reasonable period so that it may be used by the Synchronize Directory extended mechanism to allow a disconnected remote processor to update its representation of the local system.

14. Begin Grooming

This mechanism makes it possible to select a set of files for removal and determine the overall amount of space to be recovered. With reference to FIG. **23**, first verify that the global grooming lock is currently unlocked (Step S382). Then set the global grooming lock, set the total amount of space freed during grooming to zero and empty the list of files selected for deletion (Step S384). For each True File in the True File registry **126**, set the delete count to zero (Step S386).

15. Select For Removal

This grooming mechanism tentatively selects a pathname to allow its corresponding True File to be removed. With reference to FIG. **24**, first find the local directory extensions table entry record **138** corresponding to the given pathname (Step S388). Then find the True File registry entry record **140** corresponding to the True File name in the local directory extensions table entry record **138** (Step S390). Add one to the grooming delete count in the True File registry entry record **140** and add the pathname to a list of files selected for deletion (Step S392). If the grooming delete count of the True File registry entry record **140** is equal to the use count of the True File registry entry record **140**, and if the there are no entries in the dependency list of the True File registry entry record **140**, then add the size of the file indicated by the True File ID and or compressed file ID to the total amount of space freed during grooming (Step S394).

16. End Grooming

This grooming mechanism ends the grooming phase and removes all files selected for removal. With reference to FIG. **25**, for each file in the list of files selected for deletion, delete the file (Step S396) and then unlock the global grooming lock (Step S398).

Operating System Mechanisms

The next of the mechanisms provided by the present invention, operating system mechanisms, are now described.

The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

1. Open File

A mechanism to open a file is described with reference to FIGS. **26**(*a*) and **26**(*b*). This mechanism is given as input a pathname and the type of access required for the file (for example, read, write, read/write, create, etc.) and produces either the File ID of the file to be opened or an indication that no file should be opened. The local directory extensions table record **138** and region table record **142** associated with the opened file are associated with the open file for later use in other processing functions which refer to the file, such as read, write, and close.

First, determine whether or not the named file exists locally by examining the local directory extensions table **124** to determine whether there is an entry corresponding to the given pathname (Step S400). If it is determined that the file name does not exist locally, then, using the access type, determine whether or not the file is being created by this opening process (Step S402). If the file is not being created, prohibit the open (Step S404). If the file is being created, create a zero-length scratch file using an entry in local directory extensions table **124** and produce the scratch file ID of this scratch file as the result (Step S406).

If, on the other hand, it is determined in step S400 that the file name does exist locally, then determine the region in which the file is located by searching the region table **128** to find the record **142** with the longest region path which is a prefix of the file pathname (Step S408). This record identifies the region of the specified file.

Next, determine using the access type, whether the file is being opened for writing or whether it is being opened only for reading (Step S410). If the file is being opened for reading only, then, if the file is a scratch file (Step S419), return the scratch File ID of the file (Step S424). Otherwise get the True Name from the local directory extensions table **124** and make a local version of the True File associated with the True Name using the Make True File Local primitive mechanism, and then return the True File ID associated with the True Name (Step S420).

If the file is not being opened for reading only (Step S410), then, if it is determined by inspecting the region table entry record **142** that the file is in a read-only directory (Step S416), then prohibit the opening (Step S422).

If it is determined by inspecting the region table **128** that the file is in a cached region (Step S423), then send a Lock Cache message to the corresponding cache server, and wait for a return message (Step S418). If the return message says the file is already locked, prohibit the opening.

If the access type indicates that the file being modified is being rewritten completely (Step S419), so that the original data will not be required, then Delete the File using the Delete File OS mechanism (Step S421) and perform step S406. Otherwise, make a scratch copy of the file (Step S417) and produce the scratch file ID of the scratch file as the result (Step S424).

2. Close File

This mechanism takes as input the local directory extensions table entry record **138** of an open file and the data maintained for the open file. To close a file, add an entry to the audit file indicating the time and operation (create, read or write). The audit file processing (using the Process Audit File Entry primitive mechanism) will take care of assimilating the file and thereby updating the other records.

3. Read File

To read a file, a program must provide the offset and length of the data to be read, and the location of a buffer into which to copy the data read.

The file to be read from is identified by an open file descriptor which includes a File ID as computed by the Open File operating system mechanism defined above. The File ID may identify either a scratch file or a True File (or True File segment). If the File ID identifies a True File, it may be either a simple or a compound True File. Reading a file is accomplished by the following steps:

In the case where the File ID identifies a scratch file or a simple True File, use the read capabilities of the underlying operating system.

In the case where the File ID identifies a compound file, break the read operation into one or more read operations on component segments as follows:

A. Identify the segment(s) to be read by dividing the specified file offset and length each by the fixed size of a segment (a system dependent parameter), to determine the segment number and number of segments that must be read.

B. For each segment number computed above, do the following:

i. Read the compound True File index block to determine the True Name of the segment to be read.

ii. Use the Realize True File from Location primitive mechanism to make the True File segment available locally. (If that mechanism fails, the Read File mechanism fails).

iii. Determine the File ID of the True File specified by the True Name corresponding to this segment.

iv. Use the Read File mechanism (recursively) to read from this segment into the corresponding location in the specified buffer.

4. Write File

File writing uses the file ID and data management capabilities of the underlying operating system. File access (Make File Local described above) can be deferred until the first read or write.

5. Delete File or Directory

The process of deleting a file, for a given pathname, is described here with reference to FIGS. **27**(*a*) and **27**(*b*).

First, determine the local directory extensions table entry record **138** and region table entry record **142** for the file (Step S**422**). If the file has no local directory extensions table entry record **138** or is locked or is in a read-only region, prohibit the deletion.

Identify the corresponding True File given the True Name of the file being deleted using the True File registry **126** (Step S**424**). If the file has no True Name, (Step S**426**) then delete the scratch copy of the file based on its scratch file ID in the local directory extensions table **124** (Step S**427**), and continue with step S**428**.

If the file has a True Name and the True File's use count is one (Step S**429**), then delete the True File (Step S**430**), and continue with step S**428**.

If the file has a True Name and the True File's use count is greater than one, reduce its use count by one (Step S**431**). Then proceed with step S**428**.

In Step S**428**, delete the local directory extensions table entry record, and add an entry to the audit file **132** indicating the time and the operation performed (delete).

6. Copy File or Directory

A mechanism is provided to copy a file or directory given a source and destination processor and pathname. The Copy File mechanism does not actually copy the data in the file, only the True Name of the file. This mechanism is performed as follows:

(A) Given the source path, get the True Name from the path. If this step fails, the mechanism fails.

(B) Given the True Name and the destination path, link the destination path to the True Name.

(C) If the source and destination processors have different True File registries, find (or, if necessary, create) an entry for the True Name in the True File registry table **126** of the destination processor. Enter into the source ID field of this new entry the source processor identity.

(D) Add an entry to the audit file **132** indicating the time and operation performed (copy).

This mechanism addresses capability of the system to avoid copying data from a source location to a destination location when the destination already has the data. In addition, because of the ability to freeze a directory, this mechanism also addresses capability of the system immediately to make a copy of any collection of files, thereby to support an efficient version control mechanisms for groups of files.

7. Move File or Directory

A mechanism is described which moves (or renames) a file from a source path to a destination path. The move operation, like the copy operation, requires no actual transfer of data, and is performed as follows:

(A) Copy the file from the source path to the destination path.

(B) If the source path is different from the destination path, delete the source path.

8. Get File Status

This mechanism takes a file pathname and provides information about the pathname. First the local directory extensions table entry record **138** corresponding to the pathname given is found. If no such entry exists, then this mechanism fails, otherwise, gather information about the file and its corresponding True File from the local directory extensions table **124**. The information can include any information shown in the data structures, including the size, type, owner, True Name, sources, time of last access, time of last modification, state (local or not, assimilated or not, compressed or not), use count, expiration date, and reservations.

9. Get Files in Directory

This mechanism enumerates the files in a directory. It is used (implicitly) whenever it is necessary to determine whether a file exists (is present) in a directory. For instance, it is implicitly used in the Open File, Delete File, Copy File or Directory, and Move File operating system mechanisms, because the files operated on are referred to by pathnames containing directory names. The mechanism works as follows:

The local directory extensions table **124** is searched for an entry **138** with the given directory pathname. If no such entry is found, or if the entry found is not a directory, then this mechanism fails.

If there is a corresponding True File field in the local directory extensions table record, then it is assumed that the True File represents a frozen directory. The Expand Frozen Directory primitive mechanism is used to expand the existing True File into directory entries in the local directory extensions table.

Finally, the local directory extensions table **124** is again searched, this time to find each directory subordinate to the given directory. The names found are provided as the result.

Remote Mechanisms

The remote mechanisms provided by the present invention are now described. Recall that remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation.

In a presently preferred embodiment, processors communicate with each other using a remote procedure call (RPC) style interface, running over one of any number of communication protocols such as IPX/SPX or TCP/IP. Each peer processor which provides access to its True File registry **126** or file regions, or which depends on another peer processor, provides a number of mechanisms which can be used by its peers.

The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

1. Locate True File

This mechanism allows a remote processor to determine whether the local processor contains a copy of a specific True File. The mechanism begins with a True Name and a flag indicating whether to forward requests for this file to other servers. This mechanism is now described with reference to FIG. **28**.

First determine if the True File is available locally or if there is some indication of where the True File is located (for example, in the Source IDs field). Look up the requested True Name in the True File registry **126** (Step S**432**).

If a True File registry entry record **140** is not found for this True Name (Step S**434**), and the flag indicates that the request is not to be forwarded (Step S**436**), respond negatively (Step S**438**). That is, respond to the effect that the True File is not available.

One the other hand, if a True File registry entry record **140** is not found (Step S**434**), and the flag indicates that the request for this True File is to be forwarded (Step S**436**), then forward a request for this True File to some other processors in the system (Step S**442**). If the source table for the current processor identifies one or more publishing servers which should have a copy of this True File, then forward the request to each of those publishing servers (Step S**436**).

If a True File registry entry record **140** is found for the required True File (Step S**434**), and if the entry includes a True File ID or Compressed File ID (Step S**440**), respond positively (Step S**444**). If the entry includes a True File ID then this provides the identity or disk location of the actual physical representation of the file or file segment required. If the entry include a Compressed File ID, then a compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.

If the True File registry entry record **140** is found (Step S**434**) but does not include a True File ID (the File ID is absent if the actual file is not currently present at the current location) (Step S**440**), and if the True File registry entry record **140** includes one or more source processors, and if the request can be forwarded, then forward the request for this True File to one or more of the source processors (Step S**444**).

2. Reserve True File

This mechanism allows a remote processor to indicate that it depends on the local processor for access to a specific True File. It takes a True Name as input. This mechanism is described here.

(A) Find the True File registry entry record **140** associated with the given True File. If no entry exists, reply negatively.

(B) If the True File registry entry record **140** does not include a True File ID or compressed File ID, and if the True File registry entry record **140** includes no source IDs for removable storage volumes, then this processor does not have access to a copy of the given file. Reply negatively.

(C) Add the ID of the sending processor to the list of dependent processors for the True File registry entry record **140**. Reply positively, with an indication of whether the reserved True File is on line or off line.

3. Request True File

This mechanism allows a remote processor to request a copy of a True File from the local processor. It requires a True Name and responds positively by sending a True File back to the requesting processor. The mechanism operates as follows:

(A) Find the True File registry entry record **140** associated with the given True Name. If there is no such True File registry entry record **140**, reply negatively.

(B) Make the True File local using the Make True File Local primitive mechanism. If this mechanism fails, the Request True File mechanism also fails.

(C) Send the local True File in either it is uncompressed or compressed form to the requesting remote processor. Note that if the True File is a compound file, the components are not sent.

(D) If the remote file is listed in the dependent process list of the True File registry entry record **140**, remove it.

4. Retire True File

This mechanism allows a remote processor to indicate that it no longer plans to maintain a copy of a given True File. An alternate source of the True File can be specified, if, for instance, the True File is being moved from one server to another. It begins with a True Name, a requesting processor ID, and an optional alternate source. This mechanism operates as follows:

(A) Find a True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Find the requesting processor on the source list and, if it is there, remove it.

(C) If an alternate source is provided, add it to the source list for the True File registry entry record **140**.

(D) If the source list of the True File registry entry record **140** has no items in it, use the Locate Remote File primitive mechanism to search for another copy of the file. If it fails, raise a serious error.

5. Cancel Reservation

This mechanism allows a remote processor to indicate that it no longer requires access to a True File stored on the local processor. It begins with a True Name and a requesting processor ID and proceeds as follows:

(A) Find the True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Remove the identity of the requesting processor from the list of dependent processors, if it appears.

(C) If the list of dependent processors becomes zero and the use count is also zero, delete the True File.

6. Acquire True File

This mechanism allows a remote processor to insist that a local processor make a copy of a specified True File. It is used, for example, when a cache client wants to write through a new version of a file. The Acquire True File mechanism begins with a data item and an optional True Name for the data item and proceeds as follows:

(A) Confirm that the requesting processor has the right to require the local processor to acquire data items. If not, send a negative reply.

(B) Make a local copy of the data item transmitted by the remote processor.

(C) Assimilate the data item into the True File registry of the local processor.

(D) If a True Name was provided with the file, the True Name calculation can be avoided, or the mechanism can verify that the file received matches the True Name sent.

(E) Add an entry in the dependent processor list of the true file registry record indicating that the requesting processor depends on this copy of the given True File.

(F) Send a positive reply.

7. Lock Cache

This mechanism allows a remote cache client to lock a local file so that local users or other cache clients cannot change it while the remote processor is using it. The mechanism begins with a pathname and proceeds as follows:

(A) Find the local directory extensions table entry record **138** of the specified pathname. If no such entry exists, reply negatively.

(B) If an local directory extensions table entry record **138** exists and is already locked, reply negatively that the file is already locked.

(C) If an local directory extensions table entry record **138** exists and is not locked, lock the entry. Reply positively.

8. Update Cache

This mechanism allows a remote cache client to unlock a local file and update it with new contents. It begins with a pathname and a True Name. The file corresponding to the True Name must be accessible from the remote processor. This mechanism operates as follows:

Find the local directory extensions table entry record **138** corresponding to the given pathname. Reply negatively if no such entry exists or if the entry is not locked.

Link the given pathname to the given True Name using the Link Path to True Name primitive mechanism.

Unlock the local directory extensions table entry record **138** and return positively.

9. Check Expiration Date

Return current or new expiration date and possible alternative source to caller.

Background Processes and Mechanisms

The background processes and mechanisms provided by the present invention are now described. Recall that background mechanisms are intended to run occasionally and at a low priority to provide automated management capabilities with respect to the present invention.

The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links;
4. Verify Region; and
5. Groom Source List.

1. Mirror True File

This mechanism is used to ensure that files are available in alternate locations in mirror groups or archived on archival servers. The mechanism depends on application-specific migration/archival criteria (size, time since last access, number of copies required, number of existing alternative sources) which determine under what conditions a file should be moved. The Mirror True File mechanism operates as follows, using the True File specified, perform the following steps:

(A) Count the number of available locations of the True File by inspecting the source list of the True File registry entry record **140** for the True File. This step determines how many copies of the True. File are available in the system.

(B) If the True File meets the specified migration criteria, select a mirror group server to which a copy of the file should be sent. Use the Acquire True File remote mechanism to copy the True File to the selected mirror group server. Add the identity of the selected system to the source list for the True File.

2. Groom Region

This mechanism is used to automatically free up space in a processor by deleting data items that may be available elsewhere. The mechanism depends on application-specific grooming criteria (for instance, a file may be removed if there is an alternate online source for it, it has not been accessed in a given number of days, and it is larger than a given size). This mechanism operates as follows:

Repeat the following steps (i) to (iii) with more aggressive grooming criteria until sufficient space is freed or until all grooming criteria have been exercised. Use grooming information to determine how much space has been freed. Recall that, while grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

(i) Begin Grooming (using the primitive mechanism).

(ii) For each pathname in the specified region, for the True File corresponding to the pathname, if the True File is present, has at least one alternative source, and meets application specific grooming criteria for the region, select the file for removal (using the primitive mechanism).

(iii) End Grooming (using the primitive mechanism).

If the region is used as a cache, no other processors are dependent on True Files to which it refers, and all such True Files are mirrored elsewhere. In this case, True Files can be removed with impunity. For a cache region, the grooming criteria would ordinarily eliminate the least recently accessed True Files first. This is best done by sorting the True Files in the region by the most recent access time before performing step (ii) above. The application specific criteria would thus be to select for removal every True File encountered (beginning with the least recently used) until the required amount of free space is reached.

3. Check for Expired Links

This mechanism is used to determine whether dependencies on published files should be refreshed. The following steps describe the operation of this mechanism:

For each pathname in the specified region, for each True File corresponding to the pathname, perform the following step:

If the True File registry entry record **140** corresponding to the True File contains at least one source which is a publishing server, and if the expiration date on the dependency is past or close, then perform the following steps:

(A) Determine whether the True File registry entry record contains other sources which have not expired.

(B) Check the True Name expiration of the server. If the expiration date has been extended, or an alternate source is suggested, add the source to the True File registry entry record **140**.

(C) If no acceptable alternate source was found in steps (A) or (B) above, make a local copy of the True File.

(D) Remove the expired source.

4. Verify Region

This mechanism can be used to ensure that the data items in the True File registry **126** have not been damaged accidentally or maliciously. The operation of this mechanism is described by the following steps:

(A) Search the local directory extensions table **124** for each pathname in the specified region and then perform the following steps:

(i) Get the True File name corresponding to the pathname;

(ii) If the True File registry entry **140** for the True File does not have a True File ID or compressed file ID, ignore it.

(iii) Use the Verify True File mechanism (see extended mechanisms below) to confirm that the True File specified is correct.

5. Groom Source List

The source list in a True File entry should be groomed sometimes to ensure there are not too many mirror or archive copies. When a file is deleted or when a region definition or its mirror criteria are changed, it may be necessary to inspect the affected True Files to determine whether there are too many mirror copies. This can be done with the following steps:

For each affected True File,

(A) Search the local directory extensions table to find each region that refers to the True File.

(B) Create a set of "required sources", initially empty.

(C) For each region found,

(a) determine the mirroring criteria for that region,

(b) determine which sources for the True File satisfy the mirroring criteria, and

(c) add these sources to the set of required sources.

(D) For each source in the True File registry entry, if the source identifies a remote processor (as opposed to removable media), and if the source is not a publisher, and if the source is not in the set of required sources, then eliminate the source, and use the Cancel Reservation remote mechanism to eliminate the given processor from the list of dependent processors recorded at the remote processor identified by the source.

Extended Mechanisms

The extended mechanisms provided by the present invention are now described. Recall that extended mechanisms run within application programs over the operating system to provide solutions to specific problems and applications.

The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize Directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at Location;
7. Verify True File;
8. Track for Accounting Purposes; and
9. Track for Licensing Purposes.

1. Inventory Existing Directory

This mechanism determines the True Names of files in an existing on-line directory in the underlying operating system. One purpose of this mechanism is to install True Name mechanisms in an existing file system.

An effect of such an installation is to eliminate immediately all duplicate files from the file system being traversed. If several file systems are inventoried in a single True File registry, duplicates across the volumes are also eliminated.

(A) Traverse the underlying file system in the operating system. For each file encountered, excluding directories, perform the following:

(i) Assimilate the file encountered (using the Assimilate File primitive mechanism). This process computes its True Name and moves its data into the True File registry **126**.

(ii) Create a pathname consisting of the path to the volume directory and the relative path of the file on the media. Link this path to the computed True Name using the Link Path to True Name primitive mechanism.

2. Inventory Removable, Read-Only Files

A system with access to removable, read-only media volumes (such as WORM disks and CD-ROMs) can create a usable inventory of the files on these disks without having to make online copies. These objects can then be used for archival purposes, directory overlays, or other needs. An operator must request that an inventory be created for such a volume.

This mechanism allows for maintaining inventories of the contents of files and data items on removable media, such as diskettes and CD-ROMs, independent of other properties of the files such as name, location, and date of creation.

The mechanism creates an online inventory of the files on one or more removable volumes, such as a floppy disk or CD-ROM, when the data on the volume is represented as a directory. The inventory service uses a True Name to identify each file, providing a way to locate the data independent of its name, date of creation, or location.

The inventory can be used for archival of data (making it possible to avoid archiving data when those data are already on a separate volume), for grooming (making it possible to delete infrequently accessed files if they can be retrieved from removable volumes), for version control (making it possible to generate a new version of a CD-ROM without having to copy the old version), and for other purposes.

The inventory is made by creating a volume directory in the media inventory in which each file named identifies the data item on the volume being inventoried. Data items are not copied from the removable volume during the inventory process.

An operator must request that an inventory be created for a specific volume. Once created, the volume directory can be frozen or copied like any other directory. Data items from either the physical volume or the volume directory can be accessed using the Open File operating system mechanism which will cause them to be read from the physical volume using the Realize True File from Location primitive mechanism.

To create an inventory the following steps are taken:

(A) A volume directory in the media inventory is created to correspond to the volume being inventoried. Its contextual name identifies the specific volume.

(B) A source table entry **144** for the volume is created in the source table **130**. This entry **144** identifies the physical source volume and the volume directory created in step (A).

(C) The file system on the volume is traversed. For each file encountered, excluding directories, the following steps are taken:

(i) The True Name of the file is computed. An entry is created in the True Name registry **124**, including the True Name of the file using the primitive mechanism. The source field of the True Name registry entry **140** identifies the source table entry **144**.

(ii) A pathname is created consisting of the path to the volume directory and the relative path of the file on the media. This path is linked to the computed True Name using Link Path to True Name primitive mechanism.

(D) After all files have been inventoried, the volume directory is frozen. The volume directory serves as a table of contents for the volume. It can be copied using the Copy File or Directory primitive mechanism to create an "overlay" directory which can then be modified, making it possible to edit a virtual copy of a read-only medium.

3. Synchronize Directories

Given two versions of a directory derived from the same starting point, this mechanism creates a new, synchronized version which includes the changes from each. Where a file is changed in both versions, this mechanism provides a user exit for handling the discrepancy. By using True Names, comparisons are instantaneous, and no copies of files are necessary.

This mechanism lets a local processor synchronize a directory to account for changes made at a remote processor. Its purpose is to bring a local copy of a directory up to date after a period of no communication between the local and remote processor. Such a period might occur if the local processor

were a mobile processor detached from its server, or if two distant processors were run independently and updated nightly.

An advantage of the described synchronization process is that it does not depend on synchronizing the clocks of the local and remote processors. However, it does require that the local processor track its position in the remote processor's audit file.

This mechanism does not resolve changes made simultaneously to the same file at several sites. If that occurs, an external resolution mechanism such as, for example, operator intervention, is required.

The mechanism takes as input a start time, a local directory pathname, a remote processor name, and a remote directory pathname name, and it operates by the following steps:

(A) Request a copy of the audit file **132** from the remote processor using the Request True File remote mechanism.

(B) For each entry **146** in the audit file **132** after the start time, if the entry indicates a change to a file in the remote directory, perform the following steps:

(i) Compute the pathname of the corresponding file in the local directory. Determine the True Name of the corresponding file.

(ii) If the True Name of the local file is the same as the old True Name in the audit file, or if there is no local file and the audit entry indicates a new file is being created, link the new True Name in the audit file to the local pathname using the Link Path to True Name primitive mechanism.

(iii) Otherwise, note that there is a problem with the synchronization by sending a message to the operator or to a problem resolution program, indicating the local pathname, remote pathname, remote processor, and time of change.

(C) After synchronization is complete, record the time of the final change. This time is to be used as the new start time the next time this directory is synchronized with the same remote processor.

4. Publish Region

The publish region mechanism allows a processor to offer the files in a region to any client processors for a limited period of time.

The purpose of the service is to eliminate any need for client processors to make reservations with the publishing processor. This in turn makes it possible for the publishing processor to service a much larger number of clients.

When a region is published, an expiration date is defined for all files in the region, and is propagated into the publishing system's True File registry entry record **140** for each file.

When a remote file is copied, for instance using the Copy File operating system mechanism, the expiration date is copied into the source field of the client's True File registry entry record **140**. When the source is a publishing system, no dependency need be created.

The client processor must occasionally and in background, check for expired links, to make sure it still has access to these files. This is described in the background mechanism Check for Expired Links.

5. Retire Directory

This mechanism makes it possible to eliminate safely the True Files in a directory, or at least dependencies on them, after ensuring that any client processors depending on those files remove their dependencies. The files in the directory are not actually deleted by this process. The directory can be deleted with the Delete File operating system mechanism.

The mechanism takes the pathname of a given directory, and optionally, the identification of a preferred alternate source processor for clients to use. The mechanism performs the following steps:

(A) Traverse the directory. For each file in the directory, perform the following steps:

(i) Get the True Name of the file from its path and find the True File registry entry **140** associated with the True Name.

(ii) Determine an alternate source for the True File. If the source IDs field of the TFR entry includes the preferred alternate source, that is the alternate source. If it does not, but includes some other source, that is the alternate source. If it contains no alternate sources, there is no alternate source.

(iii) For each dependent processor in the True File registry entry **140**, ask that processor to retire the True File, specifying an alternate source if one was determined, using the remote mechanism.

6. Realize Directory at Location

This mechanism allows the user or operating system to force copies of files from some source location to the True File registry **126** at a given location. The purpose of the mechanism is to ensure that files are accessible in the event the source location becomes inaccessible. This can happen for instance if the source or given location are on mobile computers, or are on removable media, or if the network connection to the source is expected to become unavailable, or if the source is being retired.

This mechanism is provided in the following steps for each file in the given directory, with the exception of subdirectories:

(A) Get the local directory extensions table entry record **138** given the pathname of the file. Get the True Name of the local directory extensions table entry record **138**. This service assimilates the file if it has not already been assimilated.

(B) Realize the corresponding True File at the given location. This service causes it to be copied to the given location from a remote system or removable media.

7. Verify True File

This mechanism is used to verify that the data item in a True File registry **126** is indeed the correct data item given its True Name. Its purpose is to guard against device errors, malicious changes, or other problems.

If an error is found, the system has the ability to "heal" itself by finding another source for the True File with the given name. It may also be desirable to verify that the error has not propagated to other systems, and to log the problem or indicate it to the computer operator. These details are not described here.

To verify a data item that is not in a True File registry **126**, use the Calculate True Name primitive mechanism described above.

The basic mechanism begins with a True Name, and operates in the following steps:

(A) Find the True File registry entry record **140** corresponding to the given True Name.

(B) If there is a True File ID for the True File registry entry record **140** then use it. Otherwise, indicate that no file exists to verify.

(C) Calculate the True Name of the data item given the file ID of the data item.

(D) Confirm that the calculated True Name is equal to the given True Name.

(E) If the True Names are not equal, there is an error in the True File registry **126**. Remove the True File ID from the True

File registry entry record **140** and place it somewhere else. Indicate that the True File registry entry record **140** contained an error.

8. Track for Accounting Purposes

This mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another. The mechanism can be used as a basis for a value-based accounting system in which charges are based on the identity of the data stored or transmitted, rather than simply on the number of bits.

This mechanism allows the system to track possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. True names make it possible to identify each file briefly yet uniquely for this purpose.

Tracking the identities of files requires maintaining an accounting log **134** and processing it for accounting or billing purposes. The mechanism operates in the following steps:

(A) Note every time a file is created or deleted, for instance by monitoring audit entries in the Process Audit File Entry primitive mechanism. When such an event is encountered, create an entry **148** in the accounting log **134** that shows the responsible party and the identity of the file created or deleted.

(B) Every time a file is transmitted, for instance when a file is copied with a Request True File remote mechanism or an Acquire True File remote mechanism, create an entry in the accounting log **134** that shows the responsible party, the identity of the file, and the source and destination processors.

(C) Occasionally run an accounting program to process the accounting log **134**, distributing the events to the account records of each responsible party. The account records can eventually be summarized for billing purposes.

9. Track for Licensing Purposes

This mechanism ensures that licensed files are not used by unauthorized parties. The True Name provides a safe way to identify licensed material. This service allows proof of possession of specific files according to their contents without disclosing their contents.

Enforcing use of valid licenses can be active (for example, by refusing to provide access to a file without authorization) or passive (for example, by creating a report of users who do not have proper authorization).

One possible way to perform license validation is to perform occasional audits of employee systems. The service described herein relies on True Names to support such an audit, as in the following steps:

(A) For each licensed product, record in the license table **136** the True Name of key files in the product (that is, files which are required in order to use the product, and which do not occur in other products) Typically, for a software product, this would include the main executable image and perhaps other major files such as clip-art, scripts, or online help. Also record the identity of each system which is authorized to have a copy of the file.

(B) Occasionally, compare the contents of each user processor against the license table **136**. For each True Name in the license table do the following:

(i) Unless the user processor is authorized to have a copy of the file, confirm that the user processor does not have a copy of the file using the Locate True File mechanism.

(ii) If the user processor is found to have a file that it is not authorized to have, record the user processor and True Name in a license violation table.

The System in Operation

Given the mechanisms described above, the operation of a typical DP system employing these mechanisms is now described in order to demonstrate how the present invention meets its requirements and capabilities.

In operation, data items (for example, files, database records, messages, data segments, data blocks, directories, instances of object classes, and the like) in a DP system employing the present invention are identified by substantially unique identifiers (True Names), the identifiers depending on all of the data in the data items and only on the data in the data items. The primitive mechanisms Calculate True Name and Assimilate Data Item support this property. For any given data item, using the Calculate True Name primitive mechanism, a substantially unique identifier or True Name for that data item can be determined.

Further, in operation of a DP system incorporating the present invention, multiple copies of data items are avoided (unless they are required for some reason such as backups or mirror copies in a fault-tolerant system). Multiple copies of data items are avoided even when multiple names refer to the same data item. The primitive mechanisms Assimilate Data Items and New True File support this property. Using the Assimilate Data Item primitive mechanism, if a data item already exists in the system, as indicated by an entry in the True File registry **126**, this existence will be discovered by this mechanism, and the duplicate data item (the new data item) will be eliminated (or not added). Thus, for example, if a data file is being copied onto a system from a floppy disk, if, based on the True Name of the data file, it is determined that the data file already exists in the system (by the same or some other name), then the duplicate copy will not be installed. If the data item was being installed on the system by some name other than its current name, then, using the Link Path to True Name primitive mechanism, the other (or new) name can be linked to the already existing data item.

In general, the mechanisms of the present invention operate in such a way as to avoid recreating an actual data item at a location when a copy of that data item is already present at that location. In the case of a copy from a floppy disk, the data item (file) may have to be copied (into a scratch file) before it can be determined that it is a duplicate. This is because only one processor is involved. On the other hand, in a multiprocessor environment or DP system, each processor has a record of the True Names of the data items on that processor. When a data item is to be copied to another location (another processor) in the DP system, all that is necessary is to examine the True Name of the data item prior to the copying. If a data item with the same True Name already exists at the destination location (processor), then there is no need to copy the data item. Note that if a data item which already exists locally at a destination location is still copied to the destination location (for example, because the remote system did not have a True Name for the data item or because it arrives as a stream of un-named data), the Assimilate Data Item primitive mechanism will prevent multiple copies of the data item from being created.

Since the True Name of a large data item (a compound data item) is derived from and based on the True Names of components of the data item, copying of an entire data item can be avoided. Since some (or all) of the components of a large data item may already be present at a destination location, only those components which are not present there need be copied. This property derives from the manner in which True Names are determined.

When a file is copied by the Copy File or Directory operating system mechanism, only the True Name of the file is actually replicated.

When a file is opened (using the Open File operating system mechanism), it uses the Make True File Local primitive mechanism (either directly or indirectly through the Create Scratch File primitive mechanism) to create a local copy of the file. The Open File operating system mechanism uses the Make True File Local primitive mechanism, which uses the Realize True File from Location primitive mechanism, which, in turn uses the Request True File remote mechanism.

The Request True File remote mechanism copies only a single data item from one processor to another. If the data item is a compound file, its component segments are not copied, only the indirect block is copied. The segments are copied only when they are read (or otherwise needed).

The Read File operating system mechanism actually reads data. The Read File mechanism is aware of compound files and indirect blocks, and it uses the Realize True File from Location primitive mechanism to make sure that component segments are locally available, and then uses the operating system file mechanisms to read data from the local file.

Thus, when a compound file is copied from a remote system, only its True Name is copied. When it is opened, only its indirect block is copied. When the corresponding file is read, the required component segments are realized and therefore copied.

In operation data items can be accessed by reference to their identities (True Names) independent of their present location. The actual data item or True File corresponding to a given data identifier or True Name may reside anywhere in the system (that is, locally, remotely, offline, etc). If a required True File is present locally, then the data in the file can be accessed. If the data item is not present locally, there are a number of ways in which it can be obtained from wherever it is present. Using the source IDs field of the True File registry table, the location(s) of copies of the True File corresponding to a given True Name can be determined. The Realize True File from Location primitive mechanism tries to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. If, on the other hand, for some reason it is not known where there is a copy of the True File, or if the processors identified in the source IDs field do not respond with the required True File, the processor requiring the data item can make a general request for the data item using the Request True File remote mechanism from all processors in the system that it can contact.

As a result, the system provides transparent access to any data item by reference to its data identity, and independent of its present location.

In operation, data items in the system can be verified and have their integrity checked. This is from the manner in which True Names are determined. This can be used for security purposes, for instance, to check for viruses and to verify that data retrieved from another location is the desired, and requested data. For example, the system might store the True Names of all executable applications on the system and then periodically redetermine the True Names of each of these applications to ensure that they match the stored True Names. Any change in a True Name potentially signals corruption in the system and can be further investigated. The Verify Region background mechanism and the Verify True File extended mechanisms provide direct support for this mode of operation. The Verify Region mechanism is used to ensure that the data items in the True File registry have not been damaged accidentally or maliciously. The Verify True File mechanism verifies that a data item in a True File registry is indeed the correct data item given its True Name.

Once a processor has determined where (that is, at which other processor or location) a copy of a data item is in the DP system, that processor might need that other processor or location to keep a copy of that data item. For example, a processor might want to delete local copies of data items to make space available locally while knowing that it can rely on retrieving the data from somewhere else when needed. To this end the system allows a processor to Reserve (and cancel the reservation of) True Files at remote locations (using the remote mechanism). In this way the remote locations are put on notice that another location is relying on the presence of the True File at their location.

A DP system employing the present invention can be made into a fault-tolerant system by providing a certain amount of redundancy of data items at multiple locations in the system. Using the Acquire True File and Reserve True File remote mechanisms, a particular processor can implement its own form of fault-tolerance by copying data items to other processors and then reserving them there. However, the system also provides the Mirror True File background mechanism to mirror (make copies) of the True File available elsewhere in the system. Any degree of redundancy (limited by the number of processors or locations in the system) can be implemented. As a result, this invention maintains a desired degree or level of redundancy in a network of processors, to protect against failure of any particular processor by ensuring that multiple copies of data items exist at different locations.

The data structures used to implement various features and mechanisms of this invention store a variety of useful information which can be used, in conjunction with the various mechanisms, to implement storage schemes and policies in a DP system employing the invention. For example, the size, age and location of a data item (or of groups of data items) is provided. This information can be used to decide how the data items should be treated. For example, a processor may implement a policy of deleting local copies of all data items over a certain age if other copies of those data items are present elsewhere in the system. The age (or variations on the age) can be determined using the time of last access or modification in the local directory extensions table, and the presence of other copies of the data item can be determined either from the Safe Flag or the source IDs, or by checking which other processors in the system have copies of the data item and then reserving at least one of those copies.

In operation, the system can keep track of data items regardless of how those items are named by users (or regardless of whether the data items even have names). The system can also track data items that have different names (in different or the same location) as well as different data items that have the same name. Since a data item is identified by the data in the item, without regard for the context of the data, the problems of inconsistent naming in a DP system are overcome.

In operation, the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of these data items. True Names are globally unique identifiers which can be published simply by copying them. For example, a user might create a textual representation of a file on system A with True Name N (for instance as a hexadecimal string), and post it on a computer bulletin board. Another user on system B could create a directory entry F for this True Name N by using the Link Path to True Name primitive mechanism.

(Alternatively, an application could be developed which hides the True Name from the users, but provides the same public transfer service.)

When a program on system B attempts to open pathname F linked to True Name N, the Locate Remote File primitive mechanism would be used, and would use the Locate True File remote mechanism to search for True Name N on one or more remote processors, such as system A. If system B has access to system A, it would be able to realize the True File (using the Realize True File from Location primitive mechanism) and use it locally. Alternatively, system B could find True Name N by accessing any publicly available True Name server, if the server could eventually forward the request to system A.

Clients of a local server can indicate that they depend on a given True File (using the Reserve True File remote mechanism) so that the True File is not deleted from the server registry as long as some client requires access to it. (The Retire True File remote mechanism is used to indicate that a client no longer needs a given True File.)

A publishing server, on the other hand, may want to provide access to many clients, and possibly anonymous ones, without incurring the overhead of tracking dependencies for each client. Therefore, a public server can provide expiration dates for True Files in its registry. This allows client systems to safely maintain references to a True File on the public server. The Check For Expired Links background mechanism allows the client of a publishing server to occasionally confirm that its dependencies on the publishing server are safe.

In a variation of this aspect of the invention, a processor that is newly connected (or reconnected after some absence) to the system can obtain a current version of all (or of needed) data in the system by requesting it from a server processor. Any such processor can send a request to update or resynchronize all of its directories (starting at a root directory), simply by using the Synchronize Directories extended mechanism on the needed directories.

Using the accounting log or some other user provided mechanism, a user can prove the existence of certain data items at certain times. By publishing (in a public place) a list of all True Names in the system on a given day (or at some given time), a user can later refer back to that list to show that a particular data item was present in the system at the time that list was published. Such a mechanism is useful in tracking, for example, laboratory notebooks or the like to prove dates of conception of inventions. Such a mechanism also permits proof of possession of a data item at a particular date and time.

The accounting log file can also track the use of specific data items and files by content for accounting purposes. For instance, an information utility company can determine the data identities of data items that are stored and transmitted through its computer systems, and use these identities to provide bills to its customers based on the identities of the data items being transmitted (as defined by the substantially unique identifier). The assignment of prices for storing and transmitting specific True Files would be made by the information utility and/or its data suppliers; this information would be joined periodically with the information in the accounting log file to produce customer statements.

Backing up data items in a DP system employing the present invention can be done based on the True Names of the data items. By tracking backups using True Names, duplication in the backups is prevented. In operation, the system maintains a backup record of data identifiers of data items already backed up, and invokes the Copy File or Directory operating system mechanism to copy only those data items whose data identifiers are not recorded in the backup record. Once a data item has been backed up, it can be restored by retrieving it from its backup location, based on the identifier of the data item. Using the backup record produced by the backup to identify the data item, the data item can be obtained using, for example, the Make True File Local primitive mechanism.

In operation, the system can be used to cache data items from a server, so that only the most recently accessed data items need be retained. To operate in this way, a cache client is configured to have a local registry (its cache) with a remote Local Directory Extensions table (from the cache server). Whenever a file is opened (or read), the Local Directory Extensions table is used to identify the True Name, and the Make True File Local primitive mechanism inspects the local registry. When the local registry already has a copy, the file is already cached. Otherwise, the Locate True File remote mechanism is used to get a copy of the file. This mechanism consults the cache server and uses the Request True File remote mechanism to make a local copy, effectively loading the cache.

The Groom Cache background mechanism flushes the cache, removing the least-recently-used files from the cache client's True File registry. While a file is being modified on a cache client, the Lock Cache and Update Cache remote mechanisms prevent other clients from trying to modify the same file.

In operation, when the system is being used to cache data items, the problems of maintaining cache consistency are avoided.

To access a cache and to fill it from its server, a key is required to identify the data item desired. Ordinarily, the key is a name or address (in this case, it would be the pathname of a file). If the data associated with such a key is changed, the client's cache becomes inconsistent; when the cache client refers to that name, it will retrieve the wrong data. In order to maintain cache consistency it is necessary to notify every client immediately whenever a change occurs on the server.

By using an embodiment of the present invention, the cache key uniquely identifies the data it represents. When the data associated with a name changes, the key itself changes. Thus, when a cache client wishes to access the modified data associated with a given file name, it will use a new key (the True Name of the new file) rather than the key to the old file contents in its cache. The client will always request the correct data, and the old data in its cache will be eventually aged and flushed by the Groom Cache background mechanism.

Because it is not necessary to immediately notify clients when changes on the cache server occur, the present invention makes it possible for a single server to support a much larger number of clients than is otherwise possible.

In operation, the system automatically archives data items as they are created or modified. After a file is created or modified, the Close File operating system mechanism creates an audit file record, which is eventually processed by the Process Audit File Entry primitive mechanism. This mechanism uses the New True File primitive mechanism for any file which is newly created, which in turn uses the Mirror True File background mechanism if the True File is in a mirrored or archived region. This mechanism causes one or more copies of the new file to be made on remote processors.

In operation, the system can efficiently record and preserve any collection of data items. The Freeze Directory primitive mechanism creates a True File which identifies all of the files in the directory and its subordinates. Because this True File includes the True Names of its constituents, it represents the

exact contents of the directory tree at the time it was frozen. The frozen directory can be copied with its components preserved.

The Acquire True File remote mechanism (used in mirroring and archiving) preserves the directory tree structure by ensuring that all of the component segments and True Files in a compound data item are actually copied to a remote system. Of course, no transfer is necessary for data items already in the registry of the remote system.

In operation, the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items.

The Freeze Directory primitive mechanism is used to create a collection of data items. The constituent files and segments referred to by the frozen directory are maintained in the registry, without any need to make copies of the constituents each time the directory is frozen.

Whenever a pathname is traversed, the Get Files in Directory operating system mechanism is used, and when it encounters a frozen directory, it uses the Expand Frozen Directory primitive mechanism.

A frozen directory can be copied from one pathname to another efficiently, merely by copying its True Name. The Copy File operating system mechanism is used to copy a frozen directory.

Thus it is possible to efficiently create copies of different versions of a directory, thereby creating a record of its history (hence a version control system).

In operation, the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM. The inventory is independent of other properties of the data items such as their name, location, and date of creation.

The Inventory Existing Directory extended mechanism provides a way to create True File Registry entries for all of the files in a directory. One use of this inventory is as a way to pre-load a True File registry with backup record information. Those files in the registry (such as previously installed software) which are on the volumes inventoried need not be backed up onto other volumes.

The Inventory Removable, Read-only Files extended mechanism not only determines the True Names for the files on the medium, but also records directory entries for each file in a frozen directory structure. By copying and modifying this directory, it is possible to create an on line patch, or small modification of an existing read-only file. For example, it is possible to create an online representation of a modified CD-ROM, such that the unmodified files are actually on the CD-ROM, and only the modified files are online.

In operation, the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. Using the Track for Accounting Purposes extended mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another.

True Names in Relational and Object-Oriented Databases

Although the preferred embodiment of this invention has been presented in the context of a file system, the invention of True Names would be equally valuable in a relational or object-oriented database. A relational or object-oriented database system using True Names would have similar benefits to those of the file system employing the invention. For instance, such a database would permit efficient elimination of duplicate records, support a cache for records, simplify the process of maintaining cache consistency, provide location-independent access to records, maintain archives and histories of records, and synchronize with distant or disconnected systems or databases.

The mechanisms described above can be easily modified to serve in such a database environment. The True Name registry would be used as a repository of database records. All references to records would be via the True Name of the record. (The Local Directory Extensions table is an example of a primary index that uses the True Name as the unique identifier of the desired records.)

In such a database, the operations of inserting, updating, and deleting records would be implemented by first assimilating records into the registry, and then updating a primary key index to map the key of the record to its contents by using the True Name as a pointer to the contents.

The mechanisms described in the preferred embodiment, or similar mechanisms, would be employed in such a system. These mechanisms could include, for example, the mechanisms for calculating true names, assimilating, locating, realizing, deleting, copying, and moving True Files, for mirroring True Files, for maintaining a cache of True Files, for grooming True Files, and other mechanisms based on the use of substantially unique identifiers.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiments, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

We claim:

1. A computer-implemented method, the method comprising:

(A) for a first data item comprising a first plurality of parts,

(a1) applying a first function to each part of said first plurality of parts to obtain a corresponding part value for each part of said first plurality of parts, wherein each part of said first plurality of parts comprises a corresponding sequence of bits, and wherein the part value for each particular part of said first plurality of parts is based, at least in part, on the corresponding bits in the particular part, and wherein two identical parts will have the same part value as determined using said first function, wherein said first function comprises a first hash function; and

(a2) obtaining a first value for the first data item, said first value obtained by applying a second function to the part values of said first plurality of parts of said first data item, said second function comprising a second hash function;

(B) for a second data item comprising a second plurality of parts,

(b1) applying said first function to each part of said second plurality of parts to obtain a corresponding part value for each part of said second plurality of parts, wherein each part of said second plurality of parts consists of a corresponding sequence of bits, and wherein the part value for each particular part of said second plurality of parts is based, at least in part, on the corresponding bits in the particular part of the second plurality of parts; and

(b2) obtaining a second value for the second data item by applying said second function to the part values of said second plurality of parts of said second data item; and

(C) ascertaining whether or not said first data item corresponds to said second data item based, at least in part, on said first value and said second value.

2. The method of claim **1** wherein said first data item corresponds to said second data item when said first value is identical to said second value.

3. The method of claim **1** wherein said ascertaining in (C) is used to determine whether the first data item matches the second data item.

4. The method of claim **1** wherein said ascertaining in (C) is used to determine whether the first data item is a copy of the second data item.

5. The method of claim **1** wherein the first function is the same as the second function.

6. The method of claim **1** wherein the first plurality of parts of the first data item are non-overlapping, and wherein the second plurality of parts of the second data item are non-overlapping.

7. The method of claim **1** wherein each of the parts in the first plurality of parts and each of the parts in the second plurality of parts is the same size.

8. A computer-implemented method comprising:

(A) maintaining a database of values, at least one value for each data item of a plurality of data items,

wherein each data item of the plurality of data items comprises a corresponding one or more parts, and

wherein each of the one or more parts of each data item comprises a corresponding sequence of bits, and

wherein each of the one or more parts of each data item has a corresponding part value, the part value for each particular part being based on a first given function of the corresponding sequence bits for that particular part,

wherein two identical parts will have the same part value as determined using the first given function, and

the value for each particular data item being based, at least in part, on a second given function of the part values of the one or more parts of that particular data item, and

wherein the first given function comprises a first hash function, and the second given function comprises a second hash function;

(B) obtaining a second value, the second value corresponding to a second data item, the second data item comprising a corresponding one or more parts,

each of the one or more parts of the second data item comprising a corresponding sequence of bits,

each of the one or more parts of the second data item having a corresponding part value,

wherein the part value for each particular part of the second data item is based on the first given function of the corresponding sequence of bits in that particular part of the second data item; and

wherein the second value is based on the second function of the one or more part values of the second data item; and

(C) ascertaining whether or not the second data item corresponds to any of the plurality of data items, based, at least in part, on whether or not the second value corresponds to any value in the database of values.

9. The method of claim **8** wherein the second value corresponds to a particular value in the database of values when the second value is equal to the particular value in the database of values.

10. The method of claim **8** wherein the first hash function is selected from the functions MD5 and SHA.

11. The method of claim **10** wherein the second hash function is selected from the functions MD5 and SHA.

12. The method of claim **8** wherein the first given function is the same as the second given function.

13. The method of claim **8** wherein the database comprises a mapping from data item values to corresponding data items.

14. The method of claim **8** wherein the second value is obtained as part of a search for the second data item.

15. The method of claim **8** wherein the second value is obtained as part of a search for data items matching the second data item.

16. The method of claim **8** further comprising:

(D) when the second value corresponds to a particular value in the database, providing information about a particular data item corresponding to the particular entry.

17. The method of claim **8** wherein the step (B) of obtaining the second value comprises calculating the second value.

18. The method of claim **8** wherein at least some of the data items are files.

19. The method of claim **8** wherein the database comprises a mapping from values to corresponding data items, and wherein, when the second value corresponds to the first value, information about the corresponding data item is provided.

20. The method of claim **8** wherein the parts are segments.

21. The method of claim **8** further comprising:

(D) adding a new value to the database, wherein the new value corresponds to a new data item distinct from the plurality of data items.

22. A computer-implemented method comprising:

(A) obtaining a particular data item value corresponding to a particular data item, the particular data item comprising a corresponding one or more parts,

each of the one or more parts of the particular data item comprising a corresponding sequence of bits,

each of the one or more parts of the particular data item having a corresponding part value,

wherein the part value for each specific part of the one or more parts of the particular data item is based, at least in part, on a first given function of the corresponding sequence of bits in that specific part of the particular data item;

wherein two identical parts will have the same part value as determined using the first given function, and

wherein the particular data item value is based, at least in part, on a second given function of the one or more part values of the particular data item,

wherein the first given function comprises a first hash function, and the second given function comprises a second hash function; and

(B) ascertaining whether or not the particular data item corresponds to any of a plurality of data items, based, at least in part, on whether or not the particular data item value corresponds to any value in a database of data item values,

wherein the database of data item values comprises at least one data item value for each data item of the plurality of data items,

wherein each data item of the plurality of data items comprises a corresponding one or more parts, and

wherein each of the one or more parts of each data item of the plurality of data items comprises a corresponding sequence of bits, and

wherein each of the one or more parts of each data item of the plurality of data items has a corresponding part value, the part value for each particular part of the one or

more parts of each data item being based on the first given function of the corresponding sequence bits for that particular part,

the data item value for each particular data item being based, at least in part, on the second given function of the part values of the one or more parts of that particular data item.

23. The method of claim **22** wherein step (D) of adding is a background process.

24. The method of claim **22** wherein the first given function is selected from the functions MD5 and SHA.

25. The method of claim **22** wherein the second given function is selected from the functions MD5 and SHA.

26. The method of claim **22** wherein the database comprises a mapping from data item values to corresponding data items.

27. The method of claim **22** wherein the particular data item value is obtained as part of a search for the particular data item.

28. The method of claim **22** wherein the particular data item value is obtained as part of a search for data items matching the particular data item.

29. The method of claim **22** wherein the particular data item value corresponds to a value in the database when the particular data item value is equal to the value in the database.

30. The method of claim **22** wherein the data items are files.

31. The method of claim **22** wherein the first given function is the same as the second given function.

32. A computer-implemented method comprising:

(A) maintaining a database comprising a mapping of data item keys to corresponding data item information for each of a plurality of data items, wherein each data item of the plurality of data items has at least one data item key,

wherein each data item of the plurality of data items comprises a corresponding one or more portions, and

wherein each of the one or more portions of each data item comprises a corresponding sequence of bits, and

wherein each of the one or more portions of each data item has a corresponding portion value, the portion value for each particular portion being based on a first given function of the corresponding sequence bits for that particular portion, wherein the first given function comprises a first hash function, and wherein two identical portions will have the same portion value as determined using the first given function,

the particular data item key for each particular data item being based on a second given function of the portion values of the one or more portions of that particular data item, wherein the second given function comprises a second hash function;

(B) obtaining a particular value, the particular value having been determined from a corresponding one or more particular portions,

each of the one or more particular portions comprising a corresponding sequence of bits,

each of the one or more particular portions having a corresponding portion value, wherein the portion value for each specific portion of the one or more particular portions is based on the first given function of the corresponding bits in that specific portion; and

wherein the particular value is based on the second function of the portion values of the one or more particular portions; and

(C) using the particular value and the database to ascertain whether or not the one or more particular portions correspond to any of the plurality of data items.

33. The method of claim **32** wherein the one or more particular portions are obtained as part of a search.

34. The method of claim **32** further comprising:

(D) when the one or more particular portions correspond to a specific data item, providing information about that specific data item.

35. The method of claim **34** wherein the information provided in (D) includes information about a location of a copy of the specific data item.

36. A computer-implemented method comprising:

(A) for each particular data item of a plurality of data items:

(a1) determining a corresponding particular data item key; and

(a2) adding an entry to a database to map said particular data item key to information about the particular data item,

wherein each data item of the plurality of data items comprises a corresponding one or more parts, and

wherein each of the one or more parts of each data item comprises a corresponding sequence of bits, and

wherein each of the one or more parts of each data item has a corresponding part value, the part value for each particular part being based on a first given function of the corresponding sequence bits for that particular part, wherein the first given function comprises a first hash function, wherein two identical parts will have the same part value as determined using the first given function,

the data item key for each particular data item being based on a second given function of the part values of the one or more parts of that data item, wherein the second given function comprises a second hash function;

(B) determining a second key value, the second key value being based on one or more particular parts,

each of the one or more particular parts comprising a corresponding sequence of bits,

each of the one or more particular parts having a corresponding part value, wherein the part value for each specific part of the one or more particular parts is based on the first given function of the corresponding bits in that specific part; and

wherein the second key value is based on the second function of the part values of the one or more particular parts; and

(C) using the second key value and the database to ascertain whether or not the one or more particular parts correspond to any of the plurality of data items.

37. The method of claim **36** wherein the one or more particular parts are obtained as part of a search.

38. The method of claim **36** wherein the one or more particular parts are obtained as part of a search for data items comprising the one or more particular parts.

39. The method of claim **36** wherein the one or more particular parts correspond to a specific data item of the plurality of data items when the second key value is equal to a data item key of the specific data item.

40. The method of claim **36** wherein the parts are segments.

41. The method of claim **36** wherein the step (C) of using comprises:

looking up the second key value in the database.

42. The method of claim **36** wherein step (A) is a background process.

43. The method of claim **36** wherein the information about a particular data item in the database includes location information about the particular data item.

44. The method of claim **43** wherein the location information includes a list of one or more locations of the particular data item.

45. The method of claim **36** wherein the information about a particular data item in the database includes a copy of the data item.

46. A computer-implemented method comprising:

(A) for each particular file of a plurality of files:

(a2) determining a particular digital key for the particular file, wherein the particular file comprises a first one or more parts,

each part of said first one or more parts having a corresponding part value,

the part value of each specific part of said first one or more parts being based on a first function of the contents of the specific part,

wherein two identical parts will have the same part value as determined by the first function, and

wherein the particular digital key for the particular file is determined using a second function of the one or more of part values of said first one or more parts; and

(a2) adding the particular digital key of the particular file to a database, the database including a mapping from digital keys of files to information about the corresponding files;

(B) determining a search key based on search criteria,

wherein the search criteria comprise a second one or more parts, each of said second one or more parts of said search criteria having a corresponding part value, the part value of each specific part of said second one or more parts being based on the first function of the contents of the specific part, and wherein the search key is determined using the second function of the one or more of part values of said second one or more parts;

(C) attempting to match the search key with a digital key in the database; and

(D) if the search key matches a particular digital key in the database, providing information about the file corresponding to the particular digital key.

47. The method of claim **46** wherein step (A) is a background process.

48. The method of claim **46** wherein the plurality of files are files located in a network of computers.

49. The method of claim **46** wherein the information provided in (D) includes location information about the file corresponding to the particular digital key.

50. The method of claim **49** wherein the location information includes a list of one or more locations of the file corresponding to the particular digital key.

51. The method of claim **46** wherein the information provided in (D) includes a copy of the file corresponding to the particular digital key.

52. A computer-implemented method comprising:

(A) for each particular file of a plurality of files:

(a1) determining a corresponding particular file key; and

(a2) adding an entry to a database to map said particular file key to information about the particular file, the information about the particular file including one or more locations of the particular file,

wherein each file of the plurality of files comprises a corresponding one or more parts, and

wherein each of the one or more parts of each file has a corresponding part value, the part value for each particular part being based on a first hash function of that particular part, wherein two identical parts will have the same part value as determined using the first hash function,

the file key for each particular file being based on a second hash function of the part values of the one or more parts of that file;

(B) determining a second key value, the second key value being based on one or more particular parts,

each of the one or more particular parts having a corresponding part value, wherein the part value for each specific part of the one or more particular parts is based on the first hash function of that specific part; and

wherein the second key value is based on the second hash function of the part values of the one or more particular parts; and

(C) comparing the second key value to key values in the database to ascertain whether or not the one or more particular parts correspond to any of the plurality of files.

53. The method of claim **52** wherein step (A) is a background process.

54. The method of claim **52** wherein the information about a particular file in the database includes a copy of the file.

55. The method of claim **52** further comprising:

(D) when the second key value corresponds to a certain key value in the database, providing information from the database corresponding to the certain key value.

56. The method of claim **55** wherein the certain key value was determined from a certain file, and wherein the information provided in (D) includes one or more locations of the certain file.

* * * * *

US008099420B2

(12) **United States Patent**
Farber et al.

(10) **Patent No.: US 8,099,420 B2**
(45) **Date of Patent: *Jan. 17, 2012**

(54) **ACCESSING DATA IN A DATA PROCESSING SYSTEM**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignees: **PersonalWeb Technologies, LLC**, Tyler, TX (US); **Level 3 Communications, LLC**, Broomfield, CO (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 1312 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **11/017,650**

(22) Filed: **Dec. 22, 2004**

(65) **Prior Publication Data**

US 2005/0114296 A1 May 26, 2005

Related U.S. Application Data

(60) Continuation of application No. 09/987,723, filed on Nov. 15, 2001, now Pat. No. 6,928,442, which is a continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.** **707/758**; 707/781; 707/821

(58) **Field of Classification Search** 707/758, 707/781, 821

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

3,668,647	A	6/1972	Evangelisti
3,835,260	A	9/1974	Prescher et al.
4,096,568	A	6/1978	Bennett et al.
4,215,402	A	7/1980	Mitchell
4,221,003	A	9/1980	Chang et al.
4,290,105	A	9/1981	Cichelli
4,376,299	A	3/1983	Rivest
4,405,829	A	9/1983	Rivest
4,412,285	A	10/1983	Neches

(Continued)

FOREIGN PATENT DOCUMENTS

EP	0 268 069	A2	5/1988

(Continued)

OTHER PUBLICATIONS

Affidavit of Timothy P. Walker in Support of CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.

(Continued)

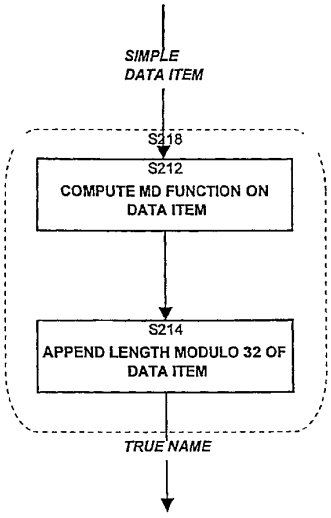
Primary Examiner — Khanh B Pham

(74) *Attorney, Agent, or Firm* — Davidson Berquist Jackson & Gowdey, LLP; Brian Siritzky

(57) **ABSTRACT**

Access to data items uses names based on the data in the data items; the name of a data item may be based, at least in part, on a function of some or all of the bits that comprise the data item. A data item may comprise an arbitrary sequence of bits. The function may include a hash function or a message digest function. The name of a data item may be compared to a list of names of other data items.

178 Claims, 31 Drawing Sheets



U.S. PATENT DOCUMENTS

4,414,624	A	11/1983	Summer, Jr.	
4,441,155	A	4/1984	Fletcher	
4,464,713	A	8/1984	Benhase	
4,490,782	A	12/1984	Dixon	
4,558,413	A	12/1985	Schmidt et al.	
4,571,700	A	2/1986	Emry, Jr.	
4,577,293	A	3/1986	Matick	
4,642,764	A	2/1987	Auslander	
4,642,793	A	2/1987	Meaden	
4,658,093	A	4/1987	Hellman	
4,675,810	A	6/1987	Gruner	
4,691,299	A	9/1987	Rivest	
4,725,945	A	2/1988	Kronstadt	
4,773,039	A	9/1988	Zamora	
4,821,184	A	4/1989	Clancy et al.	
4,887,235	A	12/1989	Holloway	
4,888,681	A	12/1989	Barnes	
4,914,571	A	4/1990	Baratz et al.	
4,914,586	A	4/1990	Swinehart et al.	
4,922,414	A	5/1990	Holloway	
4,922,417	A	5/1990	Churm et al.	707/1
4,937,863	A	6/1990	Robert et al.	
4,949,302	A	8/1990	Arnold et al.	
4,953,209	A	8/1990	Ryder, Sr. et al.	
4,972,367	A	11/1990	Burke	
5,014,192	A	5/1991	Mansfield et al.	
5,025,421	A	6/1991	Cho	
5,032,979	A	7/1991	Hecht et al.	
5,047,918	A	9/1991	Schwartz et al.	
5,050,074	A	9/1991	Marca	
5,050,212	A	9/1991	Dyson	
5,057,837	A	10/1991	Colwell	
5,077,658	A	12/1991	Bendert	
5,084,815	A	1/1992	Mazzario	
5,117,351	A	5/1992	Miller	
5,129,081	A	7/1992	Kobayashi	
5,129,082	A	7/1992	Tirfing	
5,144,667	A	9/1992	Pogue, Jr.	
5,163,147	A	11/1992	Orita	
5,179,680	A	1/1993	Colwell	
5,182,799	A	1/1993	Tamura et al.	
5,199,073	A	3/1993	Scott	
5,202,982	A	4/1993	Gramlich et al.	707/2
5,204,897	A	4/1993	Wyman	
5,204,958	A	4/1993	Cheng et al.	
5,204,966	A	4/1993	Wittenberg et al.	
5,208,858	A	5/1993	Vollert	
5,222,134	A	6/1993	Waite et al.	
5,230,051	A	7/1993	Quan	
5,239,648	A	8/1993	Nukui	
5,241,671	A	8/1993	Reed et al.	
5,247,620	A	9/1993	Fukuzawa et al.	
5,260,999	A	11/1993	Wyman	
5,276,869	A	1/1994	Forrest et al.	
5,276,901	A	1/1994	Howell	
5,287,499	A	2/1994	Nemes	707/2
5,287,514	A	2/1994	Gram	
5,297,279	A	3/1994	Bannon et al.	
5,301,286	A	4/1994	Rajani	
5,301,316	A	4/1994	Hamilton	
5,317,693	A	5/1994	Cuenod et al.	
5,321,841	A	6/1994	East et al.	
5,339,403	A	8/1994	Parker	
5,341,477	A	8/1994	Pitkin et al.	709/226
5,343,527	A	8/1994	Moore	
5,347,653	A	9/1994	Flynn et al.	
5,351,302	A	9/1994	Leighton et al.	
5,357,440	A	10/1994	Talbott et al.	
5,357,623	A	10/1994	Megory-Cohen	
5,357,630	A	10/1994	Oprescu et al.	
5,359,523	A	10/1994	Talbott et al.	
5,361,356	A	11/1994	Clark et al.	
5,371,897	A	12/1994	Brown et al.	
5,375,206	A	12/1994	Hunter et al.	
5,384,565	A	1/1995	Cannon	
5,394,555	A	2/1995	Hunter et al.	
5,403,639	A	4/1995	Belsan et al.	
5,404,508	A	4/1995	Konrad	
5,438,508	A	8/1995	Wyman	
5,442,343	A	8/1995	Cato et al.	
5,448,668	A	9/1995	Perelson et al.	
5,448,718	A	9/1995	Cohn et al.	
5,452,447	A	9/1995	Nelson et al.	707/205
5,454,000	A	9/1995	Dorfman	
5,454,039	A	9/1995	Coppersmith et al.	
5,459,860	A	10/1995	Burnett	
5,465,365	A	11/1995	Winterbottom	
5,467,471	A	11/1995	Bader	
5,475,826	A	12/1995	Fischer	
5,479,654	A	12/1995	Squibb	
5,491,817	A	2/1996	Gopal et al.	
5,499,294	A	3/1996	Friedman	
5,504,879	A	4/1996	Eisenberg et al.	
5,530,757	A	6/1996	Krawczyk	
5,537,585	A	7/1996	Blickenstaff et al.	
5,542,087	A	7/1996	Neimat et al.	707/10
5,548,724	A	8/1996	Akizawa et al.	
5,553,143	A	9/1996	Ross et al.	
5,568,181	A *	10/1996	Greenwood et al.	725/92
5,581,615	A	12/1996	Stern	
5,581,758	A	12/1996	Burnett	
5,581,764	A	12/1996	Fitzgerald et al.	
5,583,995	A *	12/1996	Gardner et al.	709/219
5,588,147	A	12/1996	Neeman et al.	
5,596,744	A	1/1997	Dao et al.	
5,600,834	A	2/1997	Howard	
5,604,803	A	2/1997	Aziz	
5,604,892	A	2/1997	Nuttall et al.	
5,630,067	A	5/1997	Kindell et al.	
5,632,031	A	5/1997	Velissaropoulos et al.	
5,638,443	A	6/1997	Stefik et al.	705/54
5,640,564	A	6/1997	Hamilton et al.	709/303
5,649,196	A	7/1997	Woodhill et al.	
5,677,952	A	10/1997	Blakley, III et al.	
5,678,038	A	10/1997	Dockter et al.	
5,678,046	A	10/1997	Cahill et al.	
5,694,472	A	12/1997	Johnson et al.	
5,694,596	A	12/1997	Campbell	
5,701,316	A	12/1997	Alferness et al.	
5,710,922	A	1/1998	Alley et al.	
5,724,425	A	3/1998	Chang et al.	
5,724,552	A	3/1998	Taoda	
5,742,807	A	4/1998	Masinter	
5,745,879	A	4/1998	Wyman	
5,757,913	A	5/1998	Bellare et al.	
5,757,915	A	5/1998	Aucsmith et al.	
5,781,629	A	7/1998	Haber	713/177
5,802,291	A	9/1998	Balick et al.	709/202
5,809,494	A	9/1998	Nguyen	707/1
5,826,049	A	10/1998	Ogata et al.	
5,835,087	A	11/1998	Herz	346/810
5,864,683	A	1/1999	Boebert et al.	
5,907,619	A	5/1999	Davis	
5,907,704	A	5/1999	Gudmundson et al.	
5,940,504	A	8/1999	Griswold	
5,978,791	A	11/1999	Farber et al.	
5,991,414	A	11/1999	Garay et al.	
6,006,018	A	12/1999	Burnett et al.	395/200.49
6,134,603	A	10/2000	Jones et al.	709/330
6,135,646	A	10/2000	Kahn et al.	
6,415,280	B1	7/2002	Farber et al.	
6,732,180	B1	5/2004	Hale et al.	
6,816,872	B1	11/2004	Squibb	
6,928,442	B2	8/2005	Farber et al.	
2002/0052884	A1	5/2002	Farber et al.	
2002/0082999	A1	6/2002	Lee et al.	
2003/0078888	A1	4/2003	Lee et al.	
2003/0078889	A1	4/2003	Lee et al.	
2003/0095660	A1	5/2003	Lee et al.	
2004/0139097	A1	7/2004	Farber et al.	
2005/0010792	A1	1/2005	Carpentier et al.	
2005/0114296	A1	5/2005	Farber et al.	
2007/0185848	A1	3/2007	Farber et al.	
2008/0065635	A1	3/2008	Farber et al.	
2008/0066191	A1	3/2008	Farber et al.	
2008/0071855	A1	3/2008	Farber et al.	
2008/0082551	A1	3/2008	Farber et al.	

FOREIGN PATENT DOCUMENTS

EP	0 315 425		5/1989
EP	0 558 945	A2	9/1993
EP	0 566 967	A2	10/1993
EP	0592045		4/1994
EP	0631 226	A1	12/1994
EP	0 654 920	A2	5/1995
EP	0 658 022	A2	6/1995
GB	2294132	A	4/1996
JP	59058564		4/1984
JP	63-106048		5/1988
JP	63-273961		11/1988
JP	2-127755		5/1990
JP	05162529		6/1993
JP	06187384	A2	7/1994
JP	06348558	A	12/1994
WO	WO 92/20021		11/1992
WO	WO 94/06087		3/1994
WO	WO 94/20913		9/1994
WO	WO 95/01599		1/1995
WO	WO 97/43717		11/1997

OTHER PUBLICATIONS

Akamai and MIT's Memorandum in Support of Their Claim Construction of USPAT 5,978,791, dated Aug. 31, 2001, from Civil Action No. 00-cv-11851RWZ.

Akamai's Answer, Affirmative Defenses and Counterclaims to Amended Complaint, filed Dec. 6, 2002, in Civil Action No. 02-CV-11430RWZ.

Akamai's Brief on Claim Construction, dated Aug. 8, 2003, from Civil Action No. 02-11430 RWZ.

Answer of Defendant RIAA to First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).

Berners-Lee, T. et al., "Hypertext Transfer Protocol—HTTP/1.0," May 1996, pp. 1-54.

Berners-Lee, T., "Universal Resource Identifiers in WWW," Jun. 1994, pp. 1-25.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Aug. 4, 1994, pp. 1-27.

Bowman, C. Mic, et al., "Harvest: A Scalable, Customizable Discovery and Access System," Mar. 12, 1995, pp. 1-29.

Brisco, T., "DNS Support for Load Balancing," Apr. 1995, pp. 1-7.

Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, 7 pages.

Browne, Shirley et al., "Location-Independent Naming for Virtual Distributed Software Repositories," 1995, printed from http:/www.netlib.org/utk/papers/lifn/main.html on Mar. 22, 2006, 18 pages.

Civil Minutes General dated Jan. 25, 2005, from Civil Action No. CV 04-7456-JFW (CTx).

Complaint for Patent Infringement, Permanent Injunction, and Damages, dated Sep. 8, 2004, from Civil Action No. CV 04-7456 JFW (AJWx).

CWIS' Opening Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-11430 RWZ.

CWIS' Reply Markman Brief Construing the Terms at Issue in U.S. Patent No. 6,415,280, dated Aug. 15, 2003, from Civil Action No. 02-11430 RWZ.

Davis, James R., "A Server for a Distributed Digital Technical Report Library," Jan. 15, 1994, pp. 1-8.

Declaration of Robert B.K. Dewar in Support of CWIS' Construction of the Terms at Issue in U.S. Patent No. 6,415,280, dated Jul. 25, 2003, from Civil Action No. 02-cv-11430RWZ.

Defendant Digital Island's Opening Brief on Claim Construction Issues dated Aug. 17, 2001, from Civil Action No. 00-cv-11851-RWZ.

Defendant Media Sentry, Inc.'s Reply Memorandum of Points and Authorities in Further Support of Its Motion to Dismiss, dated Nov. 15, 2004, from Civil Action No. CV04-7456 JFW (CTx).

Defendant MediaSentry Inc.'s Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW .

Defendant MediaSentry, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaims, dated Feb. 8, 2005, from Civil Action No. CV04-7456 JFW (CTx).

Defendant RIAA's Notice of Motion and Motion to Dismiss First Amended Complaint; Memorandum of Points and Authorities in Support Thereof, dated Dec. 13, 2004, from Civil Action No. CV04-7456 JFW (CTx).

Defendants Loudeye Corp.'s and Overpeer, Inc.'s Answer to Plaintiffs' First Amended Complaint and Counterclaim, dated Feb. 8, 2005, from Civil Action No. 04-7456 JFW (AJWx).

Defendants' Preliminary Invalidity Contentions dated Dec. 14, 2006, from Civil Action No. CV 06-5086 SJO (Ex).

European Search Report issued Dec. 23, 2004 in correpsonding European Application No. 96910762.2-2201.

Expert Report of Professor Ellis Horowitz, dated Mar. 6, 2006, from Civil Action No. 04-7456 JFW (CTx).

Expert Report of the Honorable Gerald J. Mossinghoff, dated Mar. 13, 2006, from Civil Action No. 04-7456 JFW (CTx).

Faltstrom, P. et al., "How to Interact with a Whois++ Mesh," Feb. 1996, pp. 1-9.

Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jan. 1997, pp. 1-163.

Fielding, R. et al., "Hypertext Transfer Protocol—HTTP/1.1," Jun. 1999, pp. 1-157.

First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, dated Nov. 24, 2004, from Civil Action No. CV 04-7456 JFW (CTx).

International Search Report dated Jun. 24, 1996 in corresponding international application PCT/US1996/004733.

Khare, R. and Lawrence, S., "Upgrading to TLS Within HTTP/1.1," May 2000, pp. 1-12.

Kim et al., "The Design and Implementation of Tripwire: A file System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 23, 1995, pp. 1-18.

Knuth, Donald E., "The Art of Computer Programming," 1973, vol. 3, Ch. 6.4, pp. 506-549.

Memorandum of Points and Authorities in Support of Loudeye's and Overpeer's Motion to Dismiss the First Amended Complaint for Failure to State a Claim or, in the Alternative, for a More Definitive Statement, dated Dec. 13, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).

Moats, R., "URN Syntax," May 1997, pp. 1-8.

Myers, J. and Rose, M., "The Content-MD5 Header Field," Oct. 1995, pp. 1-4.

Office Action in corresponding Japanese Application No. 531,073/1996 mailed on Apr. 25, 2006.

Office Communication in corresponding European Application No. 96910762.2-1225 dated Jan. 17, 2007.

Order Re Claim Construction dated Nov. 8, 2001, from Civil Action No. 00-11851-RWZ.

Patent Abstracts of Japan, "Device for Generating Database and Method for the Same," Application No. 03-080504, Sun Microsyst. Inc., published Jun. 1993, 38 pages.

Patent Abstracts of Japan, "Method for Registering and Retrieving Data Base," Application No. 03-187303, Nippon Telegr. & Teleph. Corp., published Feb. 1993, 11 pages.

Plaintiffs' Memorandum of Points and Authorities in Opposition to Loudeye Defendants' Motion to Dismiss, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (AJWX).

Plaintiffs' Opposition to Media Sentry's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV 04-7456 JFW (CTx).

Plaintiff's Opposition to Recording Industry Association of America's Motion to Dismiss; Memorandum of Points and Authorities in Support Thereof, dated Nov. 8, 2004, from Civil Action No. CV-04-7456 JFW (CTx).

Plaintiff's Reply to Defendant Loudeye Corp.'s and Overpeer, Inc.'s Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).

Plaintiff's Reply to Defendant MediaSentry's Counterclaims, dated Mar. 3, 2005, from Civil Action No. CV 04-7456 JFW (CTx).

Plaintiff's Reply to Defendant RIAA's Counterclaims, dated Mar. 3, 2005, from Civil Action No. 04-7456 JFW (CTx).

Reed Wade (wade@cs.utk.edu), "re: Dienst and BFD/LIFN document," Aug. 8, 1994, printed from http://www.webhistory.org/www.lists/www-talk1994q3/0416.html on Mar. 22, 2006, (7 pages).
Rivest, R., "The MD5 Message-Digest Algorithm," Apr. 1992, pp. 1-19 and errata sheet (1 page).
Rose, M., "The Content-MD5 Header Field," Nov. 1993, pp. 1-3.
Schneier, Bruce, "One-Way Hash Functions, Using Crypographic Algorithms for Hashing," 1991, printed from http://202.179135.4/data/DDJ/articles/1991/9109/91909g/9109g.htm on Mar. 22, 2006.
Sun Microsystems, Inc., "NFS: Network File System Protocol Specification," Mar. 1989, pp. 1-25.
Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, 1993, 11 pages.
Vincenzetti, David and Cotrrozzi, Massimo, "Anti Tampering Program," Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA, undated, printed from http://www.ja.net/CERI/VincenZetti_and_Cotrozzi/ATP_Anti_Tamp on Mar. 22, 2006, 8 pages.
Berners-Lee, T. et al., "Uniform Resource Locators (URL)," pp. 1-25, Dec. 1994.
Danzig, P.B., et al., ""Distributed Indexing: A Scalable Mechanism for Distributed Information Retrieval,"" Proceedings of the 14th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 220-229, Oct. 13-16, 1991.
Hauzeur, B. M., "A Model for Naming, Addressing, and Routing," ACM Trans. Inf. Syst. 4, Oct. 4, 1986), 293-311.
Khoshafian, S. N. et al. 1986. Object identity. In Conf. Proc. on Object-Oriented Programming Systems, Languages and Applications (Portland, Oregon, United States, Sep. 29-Oct. 2, 1986). N. Meyrowitz, Ed. OOPLSA '86. ACM Press, New York, NY, 406-416.
Lantz, K. A., et al., "Towards a universal directory service." In Proc. 4th Annual ACM Symp. on Principles of Distributed Computing (Minaki, Ontario, Canada). PODC '85. ACM Press, New York, NY, 250-260.
Leach, P. J., et al.. The file system of an integrated local network. In Proc. 1985 ACM 13th Annual Conf. on Comp. Sci. CSC '85. ACM Press, NY, NY, 309-324.
Leach, P.J., et al., "UIDs as Internal Names in a Distributed File System," In Proc. 1st ACM SIGACT-SIGOPS Symp. on Principles of Distributed Computing (Ottawa, Canada, Aug. 18-20, 1982). PODC '82. ACM Press, New York, NY, 34-41.
Ma, C. 1992. On building very large naming systems. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring (France, Sep. 21-23, 1992). EW 5. ACM Press, New York, NY, 1-5.
Peterson, L. L. 1988. A yellow-pages service for a local-area network. In Proc. ACM Workshop on Frontiers in Computer Communications Technology (Vermont, 1987). J. J. Garcia-Luna-Aceves, Ed. SIGCOMM '87. ACM Press, New York, NY, 235-242.
Ravindran, K. and Ramakrishnan, K. K. 1991. A naming system for feature-based service specification in distributed operating systems. SIGSMALL/PC Notes 17, 3-4 (Sep. 1991), 12-21.
Ross, K., "Hash-Routing for Collections of Shared Web Caches," IEEE Network Magazine, pp. 37-44, Nov.-Dec. 1997.
Schwartz, M., et al. 1987. A name service for evolving heterogeneous systems. In Proc. 11th ACM Symp. on OS Principles (Texas, Nov. 8-11, 1987). SOSP '87. ACM Press, NY, NY, 52-62.
Shaheen-Gouda, A. and Loucks, L. 1992. Name borders. In Proc. 5th Workshop on ACM SIGOPS European Workshop: Models and Paradigms for Distributed Systems Structuring (Mont Saint-Michel, France, Sep. 21-23, 1992). EW 5. ACM Press, NY, NY, 1-6.
Terry, D. B. 1984. An analysis of naming conventions for distributed computer systems. In Proc. ACM SIGCOMM Symp. on Communications Architectures and Protocols: Tutorials & Symp. SIGCOMM '84. ACM Press, NY, NY, 218-224.
Cheriton, David R. and Mann, Timothy P., "Decentralizing a global naming service for improved performance and fault tolerance", ACM Transactions on Computer Systems, vol. 7, No. 2, May 1989, pp. 147-183.
Request for Reexamination of U.S. Patent No. 6,928,442: Reexam Control U.S. Appl. No. 90/010,260, filed Aug. 29, 2008.
Kim et al., "Experiences with Tripwire: Using Intergrity Checkers for Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1-12.
Kim et al., "The Design and Implementation of Tripwire: A file System Intergrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1-21.
Zhiyu Tian et al., A New Hashing Function: Statistical Behaviour and Algorithm, pp. 3-13.
G. L. Friedman, Digital Camera with Apparatu for Authentication of Images Produced from an Image File, NASA Case No. NPO-19108-1-CU, U.S. Appl. No. 08/159,980, filed Nov. 24, 1993.
H. Goodman, Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Structures and File Management Report Documentation p. AD-A275 385-94-04277.
Advances in Cryptology-EUROCRYPT'93; Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23-27, 1993 Proceedings.
Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.
Advances in Cryptology-AUSCRYPT '92—Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, Dec. 13-16, 1992 Proceedings.
Witold Litwin et al., Linear Hashing for Distributed Files, ACM SIGMOD, May 1993, pp. 327-336.
Ming-Ling Lo et al., On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69-78, May 1993.
Thomas A. Berson, Differential Cryptanalysis Mod 2^{32} with Applications to MD5, pp. 69-81.
William Perrizo et al., Distributed Join Processing Performance Evaluation, Twenty-Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236-244.
Vijay Kumar, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109-113.
Birgit Pfitzman, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93, p. 74-85.
Bert dem Boer et al., Collisions for the compression function of MD_5, pp. 292-304.
Sakti Pramanik et al., Multi-Directory Hasing, 1993, Info. Sys., vol. 18, No. 1, pp. 63-74.
Murlidhar Koushik, Dynamic Hashing with Distributed Overflow Space: A File Organization with Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299-317.
Witold Litwin et al., LH*-Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21, Jun. 1993; pp. 1-22.
Yuliang Zheng et al., HAVAL—A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), pp. 83-105.
Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, pp. 156-164.
Gwertzman, James, et al. "The Case for Geographical Push-Caching." Technical Report HU TR 34-94 (excerpt), Harvard University, DAS, Cambridge, MA 02138, 1994, 2 pgs.
Grigni, Michelangelo, et al. "Tight Bounds on Minimum Broadcasts Networks." SIAM Journal of Discrete Mathematics, vol. 4, No. 2, May 1991, pp. 207-222.
Devine, Robert. "Design and Implementation of DDH: A Distributed Dynamic Hashing Algorithm." In Proceedings of 4th International Conference on Foundations of Data Organizations and Algorithms, 1993, pp. 101-114.
Deering, Stephen, et al. "Multicast Routing in Datagram Internetworks and Extended LANs." ACM Transactions on Computer Systems, vol. 8, No. 2, May 1990, pp. 85-110.
Cormen, Thomas H., et al. *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 1994, pp. 219-243, 991-993.
Naor, Moni, et al. "The Load, Capacity and Availability of Quorum Systems." In Proceedings of the 35th IEEE Symposium on Foundations of Computer Science, Nov. 1994, pp. 214-225.
Nisan, Noam. "Psuedorandom Generators for Space-Bounded Computation." In Proceedings of the Twenty-Second Annual ACM Symposium on Theory of Computing, May 1990, pp. 204-212.
Palmer, Mark, et al. "Fido: A Cache that Learns to Fetch." In Proceedings of the 17th International Conference on Very Large Data Bases, Sep. 1991, pp. 255-264.

Peleg, David, et al. "The Availabiltiy of Quorum Systems." Information and Computation 123, 1995, 210-223.
Rabin, Michael. "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance." Journal of the ACM, vol. 36, No. 2, Apr. 1989, pp. 335-348.
Ravi, R., "Rapid Rumor Ramification: Approximating the Minimum Broadcast Time." In Proceedings of the 35th IEEE Symposium on Foundation of Computer Science, Nov. 1994, pp. 202-213.
Schmidt, Jeanette, et al. "Chernoff-Hoeffding Bounds for Applications with Limited Independence." In Proceedings of the 4th ACS-SIAM Symposium on Discrete Algorithms, 1993, pp. 331-340.
Tarjan, Robert Endre, et al. "Storing a Sparse Table." Communications of the ACM, vol. 22, No. 11, Nov. 1979, pp. 606-611.
Wegman, Mark, et al. "New Hash Functions and Their Use in Authentication and Set Equality." Journal of Computer and System Sciences vol. 22, Jun. 1981, pp. 265-279.
Vitter, Jeffrey Scott, et al. "Optimal Prefetching via Data Compression." In Proceedings of 32nd IEEE Symposium on Foundations of Computer Science, Nov. 1991, pp. 121-130.
Fredman, Michael, et al. "Storing a Sparse Table with 0(1) Worst Case Access Time." Journal of the Association for Computing Machinery, vol. 31, No. 3, Jul. 1984, pp. 538-544.
Yao, Andrew Chi-Chih. "Should Tables be Sorted?" Journal of the Association for Computing Machinery, vol. 28, No. 3, Jul. 1981, pp. 615-628.
Floyd, Sally, et al. "A reliable Multicast Framework for Light-Weight Sessions and Application Level Framing." In Proceedings of ACM SIGCOMM '95, pp. 342-356.
Feeley, Michael, et al. "Implementing Global Memory Management in a Workstation Cluster." In Proceedings of the 15th ACM Symposium on Operating Systems Principles, 1995, pp. 201-212.
Carter, J. Lawrence, et al. "Universal Classes of Hash Functions." Journal of Computer and System Sciences, vol. 18, No. 2, Apr. 1979, pp. 143-154.
Patent Abstracts of Japan, "Electronic Mail Multiplexing System and Communication Control Method in The System." 06/30/19993, JP 05162529.
Kim et al., "Experiencess with Tripwire: Using Intergrity Checkers for Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1-12.
Kim et al., "The Design and Implementation of Tripwire: A file System Intergrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1-21.
Bert Dem Boer et al., Collisions for the compression function of MD_5, pp. 292-304.
Sakti Pramanik et al., Multi-Directory Hashing, 1993, Info. Sys., vol. 18, No. 1, pp. 63-74.
Murlidhar Koushik, Dynamic Hashing with Distrubuted Overflow Space: A File Organization with Good Insertion Performance, 1993, Info. Sys., vol. 18, No. 5, pp. 299-317.
Witold Litwin et al., LH*-Linear Hashing for Distributed Files, HP Labs Tech. Report No. HPL-93-21, Jun. 1993, pp. 1-22.
Yuliang Zheng et al., HAVAL-A One-Way Hashing Algorithm with Variable Length of Output (Extended Abstract), 1993, pp. 83-105.
Chris Charnes and Josef Pieprzky, Linear Nonequivalence versus Nonlinearity, Pieprzky, 1993, pp. 156-164.
Withold Litwin et al., Linear Hashing for Distributed Files, ACM SIGMOD, May 1993, pp. 327-336.
Ming-Ling Lo et al., On Optimal Processor Allocation to Support Pipelined Hash Joins, ACM SIGMOD, pp. 69-78, May 1993.
Thomas A. Berson, Differential Cryptanalysis Mod 2^{32} with Applications to MD5, pp. 69-81.
William Perrizo et al., Distributed Join Processing Performance Evaluation, Twenty-Seventh Hawaii International Conference on System Sciences, vol. II, pp. 236-244.
Vijay Kumar, A Concurrency Control Mechanism Based on Extendible Hashing for Main Memory Database Systems, ACM, vol. 3, 1989, pp. 109-113.
Birgit Pfitzman, Sorting Out Signature Schemes, Nov. 1993, 1st Conf. Computer & Comm. Security '93, p. 74-85.
Zhiyu Tian et al., A New Hashing Function: Statistical Behaviour and Algorithm, Spring 1993, pp. 3-13.
G.L. Friedman, Digital Camera with Apparatus for Authentication of Images Produced from an Image File, NASA Case No. NPO-19108-1-CU, U.S. Appl. No. 08/159,980, Nov. 24, 1993.
H. Goodman, Ada, Object-Oriented Techniques, and Concurrency in Teaching Data Structures and File Management Report Documentation p. AD-A275 385-94-04277.
Advances in Cryptology-EUROCRYPT '93, Workshop on the Theory and Application of Cryptographic Techniques Lofthus, Norway, May 23-27, 1993, Proceedings.
Proceedings on the 1993 ACM SIGMOD International Conference on Management of Data, vol. 22, Issue 2, Jun. 1993.
Advances of Cryptology-AUSCRYPT '92—Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, Queensland, Australia, Dec. 13-16, 1992 Proceedings.
Tarjan, Robert Endre, et al., "Storing a Sparse Table", Communications of the ACM, vol. 22, No. 11, Nov. 1979, pp. 606-611.
Wegman, Mark et al., "New Hash Functions and Their Use in Authentication and Set Equality", Journal of Computer and System Sciences, vol. 22, Jun. 1981, pp. 265-279.
Vitter, Jeffrey Scott et al., "Optimal Prefetching via Data Compression", In Proceedings of 32nd IEEE Symposium on Fundations of Computer Science, Nov. 1991, p. 121-130.
Friedman, Michael et al., "Storing a Sparse Table with 0(1) Worst Case Access Time", Journal of the Association for Computing Machinery, vol. 31, No. 3, Jul. 1984, pp. 538-544.
Yao, Andrew Chi-Chih, "Should Tables be Sorted?", Journal of the Association for Computing Machinery, vol. 28, No. 3, Jul. 1981, pp. 615-628.
Floyd, Sally et al., "A Reliable Multicast Framework for Light-Weight Sessions and Application Level Framing", In Proceedings of ACM SIGCOMM '95, pp. 342-356.
Feeley, Michael, et al., "Implementing Global Memory Management in a Workstation Cluster", In Proceedings of the 15th ACM Symposium on Operating Systems Principles, 1995, pp. 201-212.
Carter, J. Lawrence, et al., "Universal Classes of Hash Functions", Journal of Computer and System Sciences, vol. 18, No. 2, Apr. 1979, pp. 143-154.
Patent Abstracts of Japan, "Electronic Mail Multiplexing System and Communication Control Method in the System", Jun. 30, 1993, JP 05162529.
Kim et al., "Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection", COAST Labs. Dept. of Computer Sciences Purdue University, Feb. 22, 1995, pp. 1-12.
Kim et al., "The Design and Implementation of Tripwire: A File System Integrity Checker", COAST Labs. Dept. of Computer Sciences Purdue University, Nov. 19, 1993, pp. 1-21.
Peter Deutsch (peterd@bunyip.com), "Re: MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0106.html, Apr. 26, 1994.
Alexander Dupuy (dupuy@smarts.com), "RE: MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0113.html, Apr. 26, 1994.
Alexander Dupuy (dupuy@smarts.com), "MD5 and LIFNs (was: Misc Comments)", www.acl.lanl.gov/URI/archive/uri-94q2.messages/0081.html, Apr. 17, 1994.
Albert Langer (cmf851@anu.oz.au), http://groups.google.com/groups?selm=1991Aug7.225159.786%40newshost.anu.edu.au&oe=UTF-8&output=gplain, Aug. 7, 1991.
Clifford Lynch (Calur@uccmvsa.bitnet), "ietf url/uri overview draft paper (long)", www.acl.lanl.gov/URI/archive/uri-93q1.messages/0015.html, Mar. 25, 1993.
K. Sollins and L. Masinter, "Functional Requirements for Uniform Resource Names", www.w3.org/Addressing/rfc1737.txt, Dec. 1994, pp. 1-7.
W3C:ID, HTTP: A protocol for networked information, "Basic HTTP as defined in 1992", www.w3.org/Protocols/HTTP2.html, 1992.
Defendant Lime Wire, LLC's Answer, Affirmative Defenses and Counterclaims dated Nov. 15, 2007, from Civil Action No. 07-06161 VBF (PLAx).

Kim, Gene H., and Spafford, Eugene H., "Writing, Supporting, and Evaluating Tripwire: A Publicly Available Security Tool." COAST Labs. Dept. of Computer Sciences Purdue University, Mar. 12, 1994, pp. 1-23.
U.S. Appl. No. 11/980,679—May 6, 2009 PTO Office Action.
USPTO, Non-Final Office action mailed Jun. 15, 2009 in U.S. Appl. No. 11/980,687.
USPTO, Non-Final Office action mailed Jun. 18, 2009 in Reexam U.S. Appl. No. 90/010,260.
Communication from EPO in European Application No. 96 910 762.2-1225 dated May 8, 2009 [4 pgs.].
McGregor D. R. and Mariani, J. A. "Fingerprinting—A technique for file identification and maintenance," Software: Practice and Experience, vol. 12, No. 12, Dec. 1982, pp. 1165-1166.
[Proposed] Order Regarding Construction of Terms, filed Mar. 29, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [9 pgs.].
Analysis of Plaintiffs' Claim Chart for the '280 Patent As Against Defendant Media Sentry, Inc. 11 pages.
Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Media Sentry, Inc. (11916.001.0150.a) pp. 1-48.
Analysis of Plaintiffs' Claim Chart for the '791 Patent As Against Defendant Overpeer pp. 1-40.
Barbara, D., et al., "Exploiting symmetries for low-cost comparison of file copies," 8th Int'l Conf. on Distributed Computing Systems, Jun. 1988, pp. 471-479, San Jose, CA.
Campbell, M., "The Design of Text Signatures for Text Retrieval Systems," Tech. Report, Sep. 5, 1994, Deakin University, School of Computing & Math., Geelong, Australia.
Chang, W. W. et al., "A signature access method for the Starburst database system," in Proc. 15th Int'l Conf. on Very Large Data Bases (Amsterdam, The Netherlands), pp. 145-153.
Changes to Mar. 23, 2007 Deposition of Robert B. K. Dewar, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [3 pgs + cover letter.].
Communication pursuant to Article 96(2) EPC from EPO (Examination Report), Jan. 17, 2007, in Application No. EP 96 910 762.2-1225 [1 pg. with 5 pg. annex].
Complaint for Patent Infringement, Permanent Injunction and Damages, Aug. 8, 2006, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].
Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [10 pgs.].
Declaration of Charles S. Baker in Support of Defendant Lime Wire's Motion to Stay Pending Reexamination of Patent and Request for Extension of Deadlines, Aug. 29, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Defendant Lime Wire, LLC's First Amended Answer, Affirmative Defenses and Counterclaims, Oct. 2, 2008, C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].
Defendant Lime Wire, LLC's Second Amended Answer, Affirmative Defenses and Counterclaims, Oct. 27, 2008, from C.D. Cal. case No. 07-06161 VBF (PLAx) [13 pgs.].
Defendant Michael Weiss's Answer to Plaintiff's Complaint for Patent Infringement, Permanent Injunction and Damages; Demand for Jury Trial, Sep. 15, 2006, case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendant Recording Industry Association of America's Amended Notice of Motion and Motion for Partial Summary Judgment on Plaintiffs' Claims for Patent Infringement and Inducing Patent Infringement; Memorandum of Points and Authorities, May 22, 2006, redacted, original confidential, filed under seal, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [19 pgs.].
Defendant Recording Industry Association of America's and Mediasentry, Inc.'s Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, in the Alternative, Based on Patent Misuse and Unclean Hands, May 22, 2006, Redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [21 pgs.].
Defendant Recording Industry Association of America's and Mediasentry, Inc's Notice of Motion and Motion for Partial Summary Judgment Based on Implied License or, in the Alternative, Based on Patent Misuse and Unclean Hands, May 8, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [20 pgs.].
Defendant StreamCast Networks Inc.'s Answer to Plaintiff's Complaint for Patent Infringement, Permanent Injunction and Damages; Demand for Jury Trial, Sep. 5, 2006, C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendants' Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 7, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Defendant's Second Amended Preliminary Claim Constructions [Patent Rule 4-2], filed Feb. 9, 2007 in C.D. Cal. case No. CV 06-5086 SJO (Ex) [10 pgs.].
Dewar, Rebuttal Expert Report of Robert B.K. Dewar, in C.D. Cal. case No. CV 04-7456 JFW (CTx), Apr. 10, 2006 [87 pgs].
Faloutsos, C. "Access methods for text," ACM Comput. Surv. 17, 1 (Mar. 1985), 49-74.
Faloutsos, C. et al., "Description and performance analysis of signature file methods for office filing," ACM Trans. Inf. Syst. 5, 3 (Jul. 1987), 237-257.
Faloutsos, C. et al., "Signature files: an access method for documents and its analytical performance evaluation," ACM Trans. Inf. Syst. 2, 4 (Oct. 1984), 267-288.
Federal Information Processing Standards (FIPS) Publication 180-1; Secure Hash Standard, Apr. 17, 1995 [17 pgs.].
Feigenbaum, J. et al., "Cryptographic protection of databases and software," in Distributed Computing and Cryptography: Proc. DIMACS Workshop, Apr. 1991, pp. 161-172, American Mathematical Society, Boston, Mass.
First Amended Answer of Defendant Mediasentry to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [29 pgs.].
First Amended Answer of Defendant RIAA to Second Amended Complaint and Counterclaim, Apr. 24, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [27 pgs.].
First Amended Complaint for Patent Infringement, Permanent Injunction and Damages, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [10 pgs.].
Harrison, M. C., "Implementation of the substring test by hashing," Commun. ACM 14, 12 (Dec. 1971), 777-779.
IEEE, The Authoritative Dictionary of IEEE Standards Terms, 7th ed., Copyright 2000, pp. 107, 176, 209, 240, 241, 432, 468, 505, 506, 682, 1016, 1113, 1266, and 1267.
Ishikawa, Y., et al., "Evaluation of signature files as set access facilities in OODBs," in Proc. of the 1993 ACM SIGMOD Inter. Conf. on Management of Data (Washington, D.C., U.S., May 1993). P. Buneman & S. Jajodia, Eds. SIGMOD '93. ACM, NY, NY, 247-256.
Joint Claim Construction and Prehearing Statement, N.D. Cal. Rule 4-3, Feb. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [20 pgs.].
Karp, R. M. and Rabin, M. O., "Efficient randomized pattern-matching algorithms," IBM J. Res. Dev. 31, 2 (Mar. 1987), 249-260.
List of Asserted Claims and Infringement Chart for Each Asserted Claim, Jul. 28, 2008, in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [31 pgs.].
Notice of Interested Parties, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Notice of Motion and Motion of Defendant Lime Wire to Stay Litigation Pending Reexamination of Patent and Request for Extension of Deadlines, Sep. 22, 2008, C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [11 pgs.].
Notice of Related Cases, filed Sep. 21, 2007 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [2 pgs.].
Panagopoulos, G., et al., "Bit-sliced signature files for very large text databases on a parallel machine architecture," in Proc. of the 4th Inter. Conf. on Extending Database Technology (EDBT), Cambridge, U.K., Mar. 1994, pp. 379-392 (Proc. LNCS 779 Springer 1994, ISBN 3-540-57818-8) [14 pgs.].
Patent Abstract, " Management System for Plural Versions," Pub. No. 63273961 A, published Nov. 11, 1988, NEC Corp.
Patent Abstracts of Japan, "Data Processor," Appln. No. 05135620, filed Jun. 7, 1993, Toshiba Corp.
Plaintiff Kinetech, Inc.'s Responses to Defendant Mediasentry's First set of Interrogatories, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [14 pgs.].

Plaintiff-Counterclaim Defendant Altnet, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Brilliant Digital Entertainment, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories, Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiff-Counterclaim Defendant Kinetech, Inc.'s Supplemental Responses to Defendant-Counterclaim Plaintiff Overpeer Inc.'s First Set of Interrogatories Mar. 8, 2006, redacted, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [24 pgs.].
Plaintiffs Altnet, Inc., Brilliant Digital, Inc., and Kinetech, Inc.'s Responses to Defendant Recording Industry Association of America's First Set of Requests for Admissions, Jan. 6, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [26 pgs.].
Plaintiffs' Claim Construction Opening Brief and Exhibits A-D, F, G; May 7, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [112 pgs.].
Plaintiffs' Preliminary Claim Constructions and Extrinsic Evidence, Feb. 6, 2006, in case CV 06-5086 SJO (Ex) [20 pgs.].
Plaintiff's Reply to Defendant Mediasentry's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. Case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiff's Reply to Defendant RIAA's Counterclaims in its Answer to the Second Amended Complaint, May 1, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [11 pgs.].
Plaintiffs' Reply to Defendants' Claim Construction Brief, filed Apr. 23, 2007 in C.D. Cal. case No. CV 06-5086 ODW (Ex) [15 pgs.].
Reply to Examination Report, Jul. 19, 2007, in Application No. EP 96 910 762.2-1225 [7 pgs.].
Rivest, R., RFC 1320, "The MD4 Message-Digest Algorithm," Apr. 1992.
Sacks-Davis, R., et al., "Multikey access methods based on superimposed coding techniques," ACM Trans. Database Syst. 12, 4 (Nov. 1987), 655-696.
Siegel, A., et al., "Deceit: A Flexible Distributed File System," Proc. Workshop on the Management of Replicated Data, Houston, TX, pp. 15-17, Nov. 8-9, 1990.
Siegel, A., et al., "Deceit: a Flexible Distributed File System," Technical Report, TR89-1042, Cornell University, Nov. 1989.
Stipulation and Proposed order to (1) Amend the Complaint, (2) Amend pretrial Schedule, and (3) Withdraw Motion to Stay, filed Sep. 8, 2008 in C.D. Cal. Case No. CV 07-06161 VBF (PLAx) [6 pgs.].
Streamcast Networks Inc.'s Supplemental Responses to Certain of Plaintiffs' First Set of Interrogatories, Apr. 16, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
StreamCast's Brief Re Claim Construction, Apr. 12, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [11 pgs.].
Transcript of Deposition of David Farber, Feb. 16, 2006, in C.D. Cal. case No. CV 04-7456 JFW (CTx) [94 pgs.].
Transcript of Deposition of Robert B. K. Dewar, Mar. 23, 2007, in C.D. Cal. case No. CV 06-5086 SJO (Ex) [61 pgs.].
Transcript of Deposition of Ronald Lachman, Feb. 1, 2006, C.D. Cal. case No. CV 04-7456 JFW (CTx) [96 pgs.].
U.S. Appl. No. 10/742,972—Dec. 22, 2009 PTO Office Action.
U.S. Appl. No. 11/980,679—Jan. 12, 2010 PTO Office Action.
U.S. Reexam Control U.S. Appl. No. 90/010,260—Jan. 29, 2010 PTO Office Action.
USPTO, Final Office Action mailed Sep. 30, 2009 in U.S. Appl. No. 11/724,232.
Bowman, C.M., et al., "Scalable Internet Resource Discovery: Research Problems and Approaches," University of Colorado, Dept. of Comp. Sci. Technical Report# CU-CS-679-93 (Oct. 1993).
Bowman, C.M., et al., "Research Problems for Scalable Internet Resource Discovery," CU-CS-643-93, Mar. 1993, University of Colorado at Boulder, Dept. of Comp. Sci.
Cate, V., "Alex—a Global Filesystem," Proceedings of the Usenix File Systems Workshop, pp. 1-11, May 1992.
Hardy D. R., et al., "Essence: A resource discovery system based on semantic file indexing," USENIX Winter 1993 Technical Conf., San Diego, CA (Jan. 1993), pp. 361-374.
Kahn, R.E., "Deposit, Registration and Recordation in an Electronic Copyright Mangement System," Tech. Report, Corporation for National Research Initiatives, Reston, Virginia, Aug. 1992 (downloaded from http://archive.ifla.org/documents/infopol/copyright/kahn.txt on Nov. 10, 2010).
Manber, U., Finding similar files in a large file system, Dept. of Computer Science TR 93-33, Oct. 1993., U. of Arizona, Tucson, AZ.
Manber, U., Finding similar files in a large file system, USENIX, pp. 1-10, San Francisco, CA, Jan. 1994. (WTEC'94, Proc. USENIX Winter 1994 Technical Conf.).
Dewitt, et al. (Jun. 1984). "Implementation techniques for main memory database systems". Proc. ACM SIGMOD Conf 14 (4): 1-8.
Fagin, R., et al. Extendible hashing—a fast access method for dynamic files. ACM Trans. Database Syst. 4,3 (Sep. 1979), 315-344.
Griswold, William G.; Townsend, Gregg M. (Apr. 1993), "The Design and Implementation of Dynamic Hashing for Sets and Tables in Icon", Software—Practice and Experience 23 (4): 351-367 (Apr. 1993).
Kitsuregawa, M.; Tanaka, H.; Moto-Oka, T. (Mar. 1983). "Application of Hash to Data Base Machine and Its Architecture". New Generation Computing 1 (1): 63-74, Ohmsha, Ltd. and Springer-Verlag.
Litwin, W., "Linear Hashing: a New Tool for File and Table Addressing," Proc. 6th Int. Conf. on Very Large Databases, IEEE 1980, 212-223.
USPTO, Final Office Action, Mar. 5, 2010 in U.S. Appl. No. 11/980,687.
Fowler, et al. "A User-Level Replicated File System," AT&T Bell Laboratories Technical Memorandum 0112670-930414-05, Apr. 1993, and USENIX 1993 Summer Conference Proceedings, Cincinnati, OH, Jun. 1993.
Greene, D., et al., "Multi-Index Hashing for Information Retrieval", Nov. 20-22, 1994, Proceedings, 35th Annual Symp on Foundations of Computer Science, IEEE, pp. 722-731.
Hirano, et al, "Extendible hashing for concurrent insertions and retrievals," in Proc 4th Euromicro Workshop on Parallel and Distributed Processing, 1996 (PDP '96), Jan. 24, 1996 to Jan. 26, 1996, pp. 235-242, Braga , Portugal.
Preneel et al., "The Cryptographic Hash Function RIPEMD-160", appeared in CryptoBytes RSA Laboratories, vol. 3, No. 2, pp. 9-14, Fall, 1997 (also Bosselaers et al., "The RIPEMD-160 Cryptographic Hash Function", Jan. 1997, Dr. Dobb's Journal, pp. 24-28).
Prusker et al., "The Siphon: Managing Distant Replicated Repositories" Nov. 8-9, 1990, Proc. Management of Replicated Data IEEE.
Reply to Examination Report, Munich, Nov. 18, 2009, in Application No. EP 96 910 762.2 [19 pgs.].
Rich, K. et al, "Hobgoblin: A File and Directory Auditor", Sep. 30-Oct. 3, 1991, Lisa V., San Diego, CA.
USPTO, Advisory Action, Mar. 23, 2010, in U.S. Appl. No. 11/980,679.
USPTO, U.S. Reexam Control U.S. Appl. No. 90/010,260, Notice of Intent to Issue Ex Parte Reexamination Certificate, Apr. 8, 2010.
WIPO, International Preliminary Examination Report (IPER), Jul. 1997, PCT/US96/04733 [5 pgs.].
USPTO, Non-Final Office Action in U.S. Appl. No. 11/980,677, Jun. 4, 2010.
USPTO, Non-Final Office Action mailed Jul. 2, 2010 in U.S. Appl. No. 11/980,688.
USPTO, Notice of Allowance mailed Apr. 30, 2010 in U.S. Appl. No. 11/980,687.
USPTO, Notice of Allowance mailed Jun. 24, 2010 in U.S. Appl. No. 11/980,687.
USPTO, Supplemental Notice of Allowability mailed May 27, 2010 in U.S. Appl. No. 11/980,687.
Birrell et al., A Universal File Server, IEEE Trans. on Software Engineering, vol. SE-6, No. 5, Sep. 1980.
Filing in EPO in related application (EP1996910762), Amended claims with annotations [10 pgs.], Annex [12 pgs.], Letter filed in EPO [2 pgs.], claims [9 pgs.], Nov. 29, 2010.
Guy, R. G., Ficus: A Very Large Scale Reliable Distributed File System, Jun. 3, 1991, Technical Report CSD-910018, Computer Science Department, University of California Los Angeles, CA.
Guy, R.G.,et al, Implementation of the Ficus Replicated File System, Proc. of the Summer USENIX Conference, Anaheim, CA, Jun. 1990, pp. 63-71.

Heidemann, J.S., "Stackable Layers: An Architecture for File System Development," UCLA, Aug., 1991 (available as Technical Report CSD-910056 of the UCLA Computer Science Department).
Needham, R.M., et al., The Cap Filing System, Proc. Sixth ACM Symp. on Operating System Principles, Nov. 1977, 11-16.
Page, Thomas W. Jr., et al., Management of Replicated Volume Location Data in the Ficus Replicated File System, Proc. of the Summer USENIX Conference, Jun. 1991, pp. 17-29.
Popek, Gerald J. et al., Replication in Ficus Distributed File Systems, Proc. of the Workshop on Management of Replicated Data, Nov. 1990, pp. 20-25.
Reiher, P. et al., Resolving File Conflicts in the Ficus File System, in Proc. of the Summer USENIX Conference, Jun. 1994, pp. 183-195.
Sturgis, H. et al, Issues in the design and use of a distributed file system, Sigops Oper. Syst. Rev., vol. 14, No. 3. (1980), pp. 55-69.
Swinehart, D., et al, WFS: A Simple Shared File System for a Distributed Environment, XEROX, Palo Alto Research Center, Palo Alto, CA, CSL-79-13, Oct. 1979.

* cited by examiner

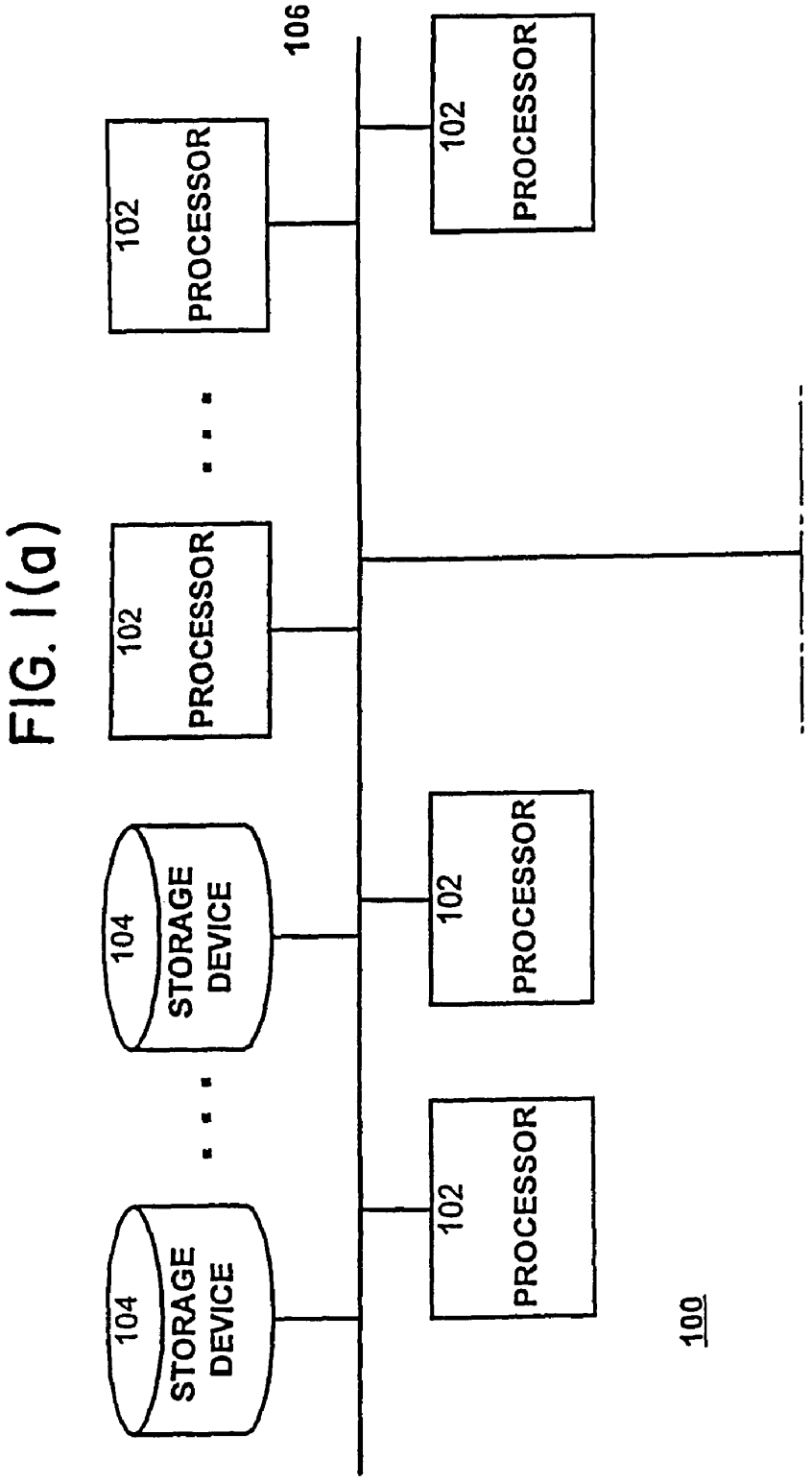

106
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
102
PROCESSOR
104
STORAGE DEVICE
104
STORAGE DEVICE
100

FIG. 1(a)

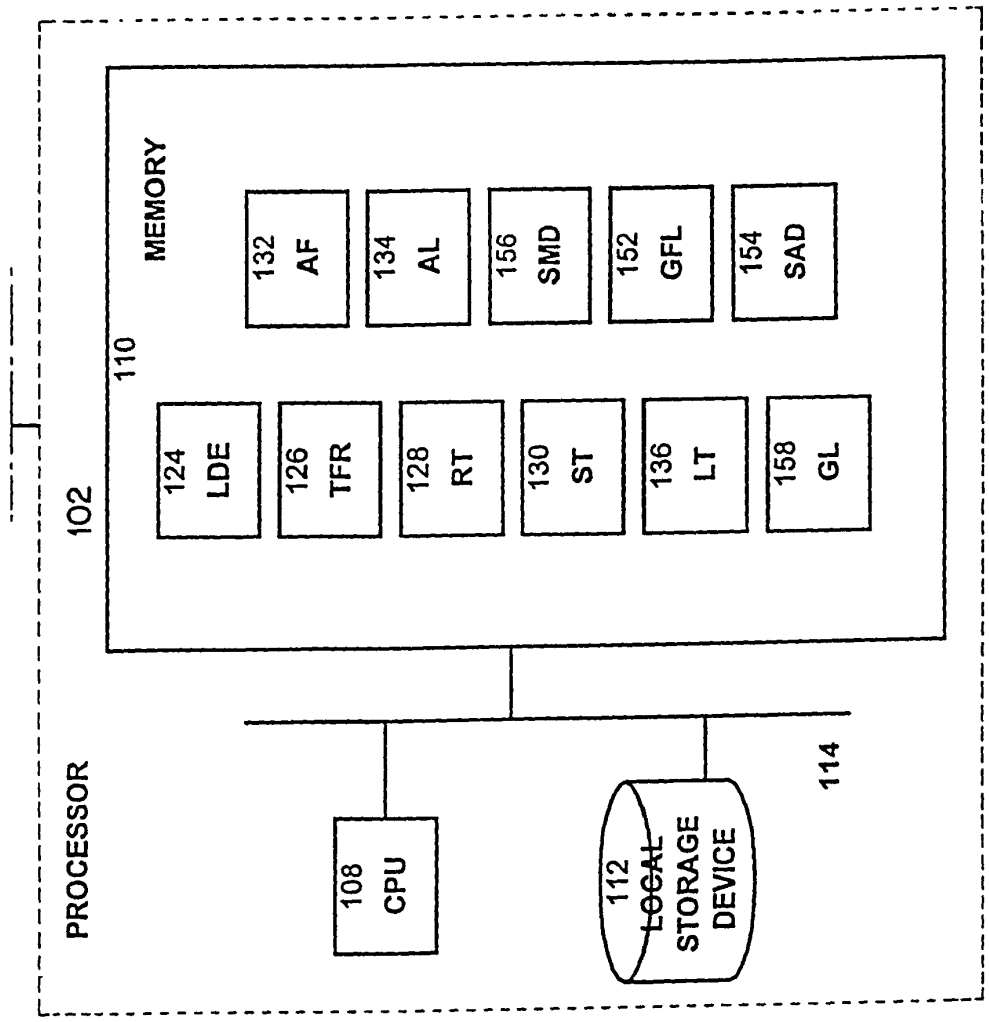

PROCESSOR
102
MEMORY
110
124 LDE
126 TFR
128 RT
130 ST
136 LT
158 GL
132 AF
134 AL
156 SMD
152 GFL
154 SAD
108 CPU
112 LOCAL STORAGE DEVICE
114

FIG. 1(b)

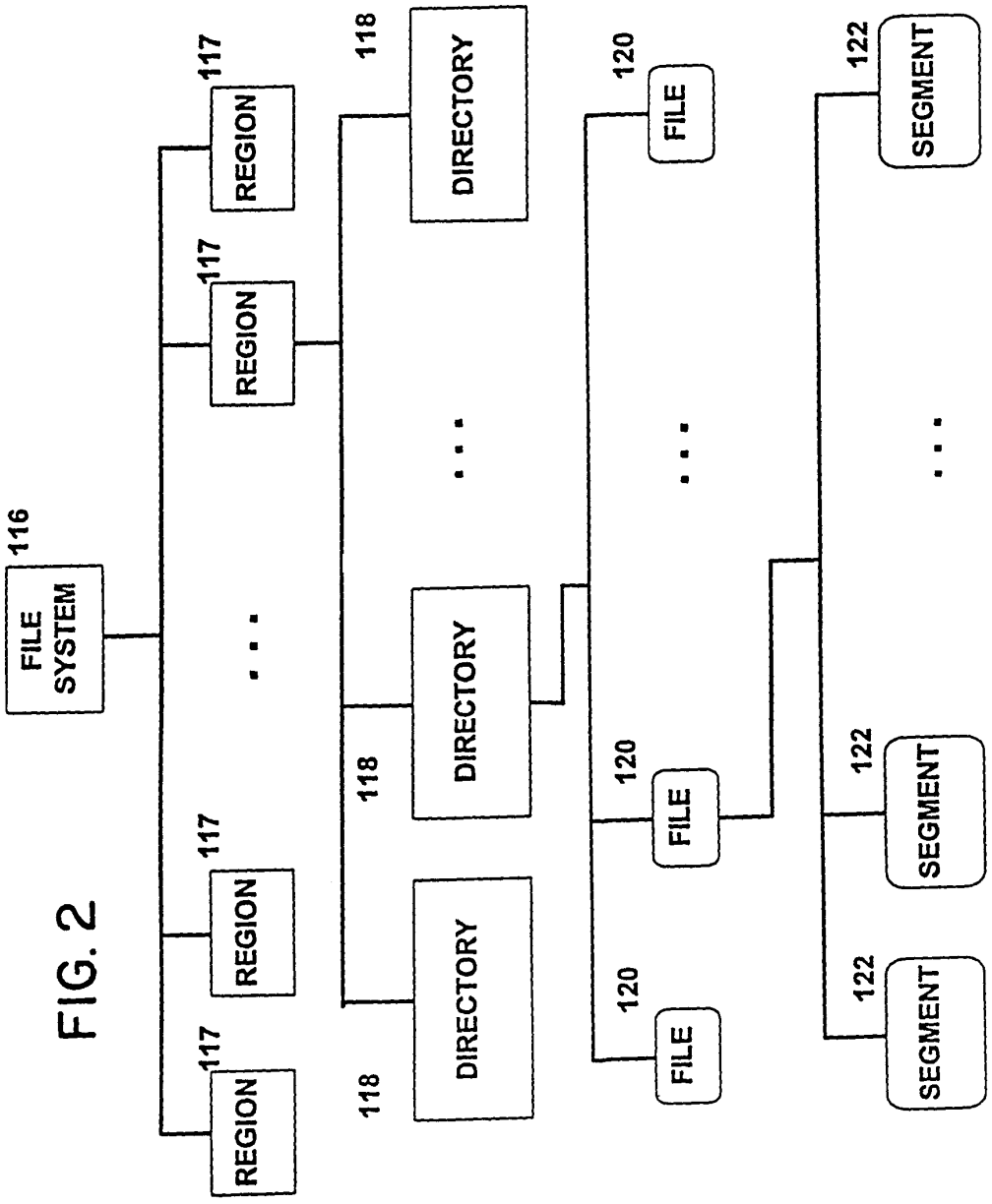

116
FILE SYSTEM
117
REGION
117
REGION
117
REGION
118
DIRECTORY
118
DIRECTORY
118
DIRECTORY
120
FILE
120
FILE
120
FILE
122
SEGMENT
122
SEGMENT
122
SEGMENT

FIG. 2

FIG. 3

138

Region ID
Pathname
True Name
Type
File ID
Time of last access
Time of last modification
Safe flag
Lock flag
Size
Owner

FIG. 4

140

True Name
File ID
Compressed File ID
Source IDs
Dependent processors
Use count
Time of last access
Expiration
Grooming delete count

142

Region ID
Region file system
Region pathname
Region status
Mirror processor(s)
Mirror duplication count
Policy

FIG. 5

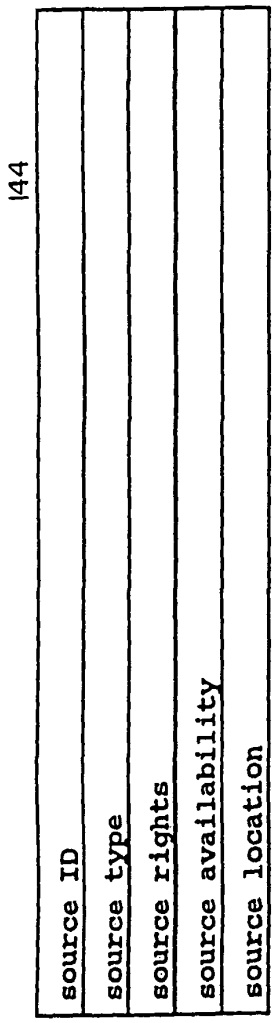


FIG. 6

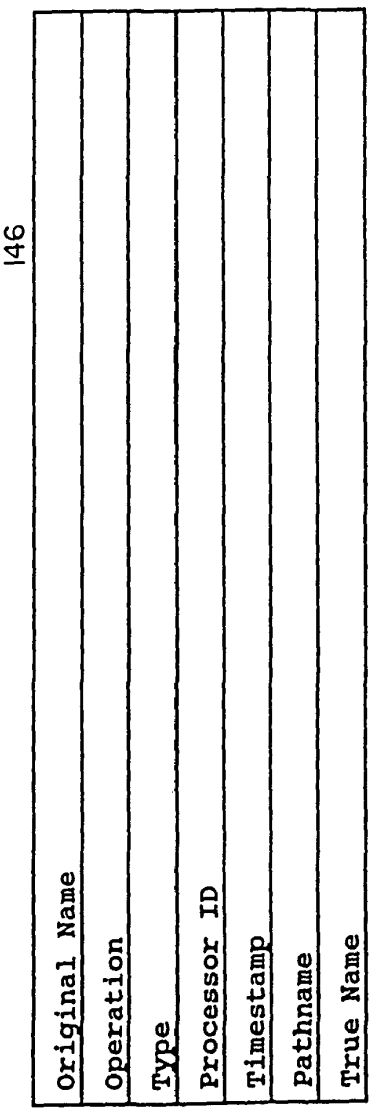


FIG. 7

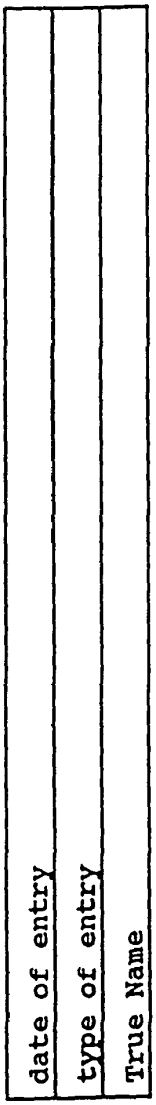


FIG. 8

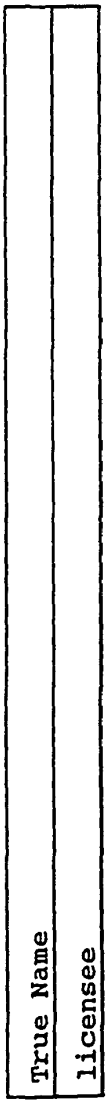
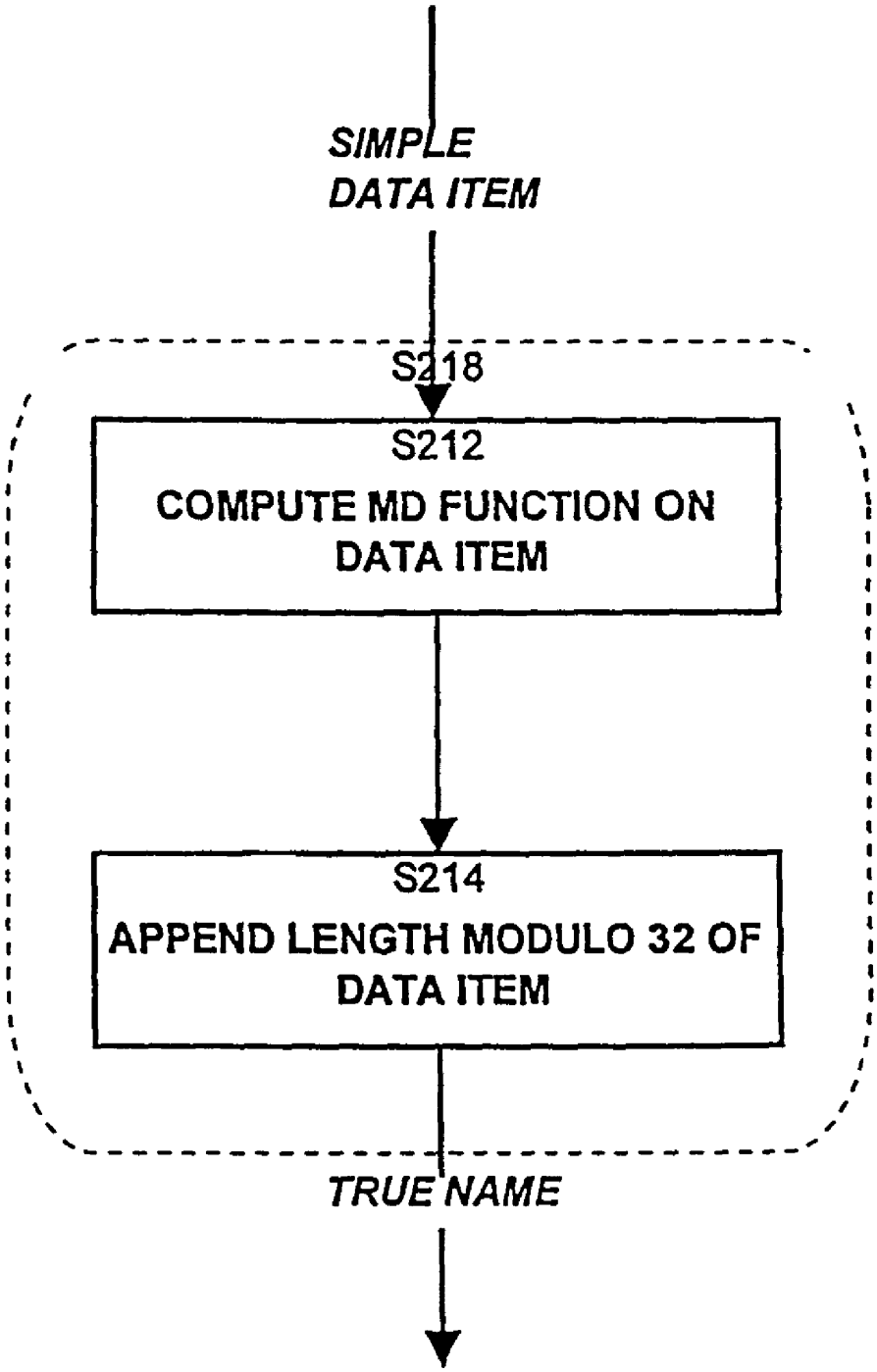


FIG. 9

FIG. 10(a)


SIMPLE
DATA ITEM
S218
S212
COMPUTE MD FUNCTION ON
DATA ITEM
S214
APPEND LENGTH MODULO 32 OF
DATA ITEM
TRUE NAME

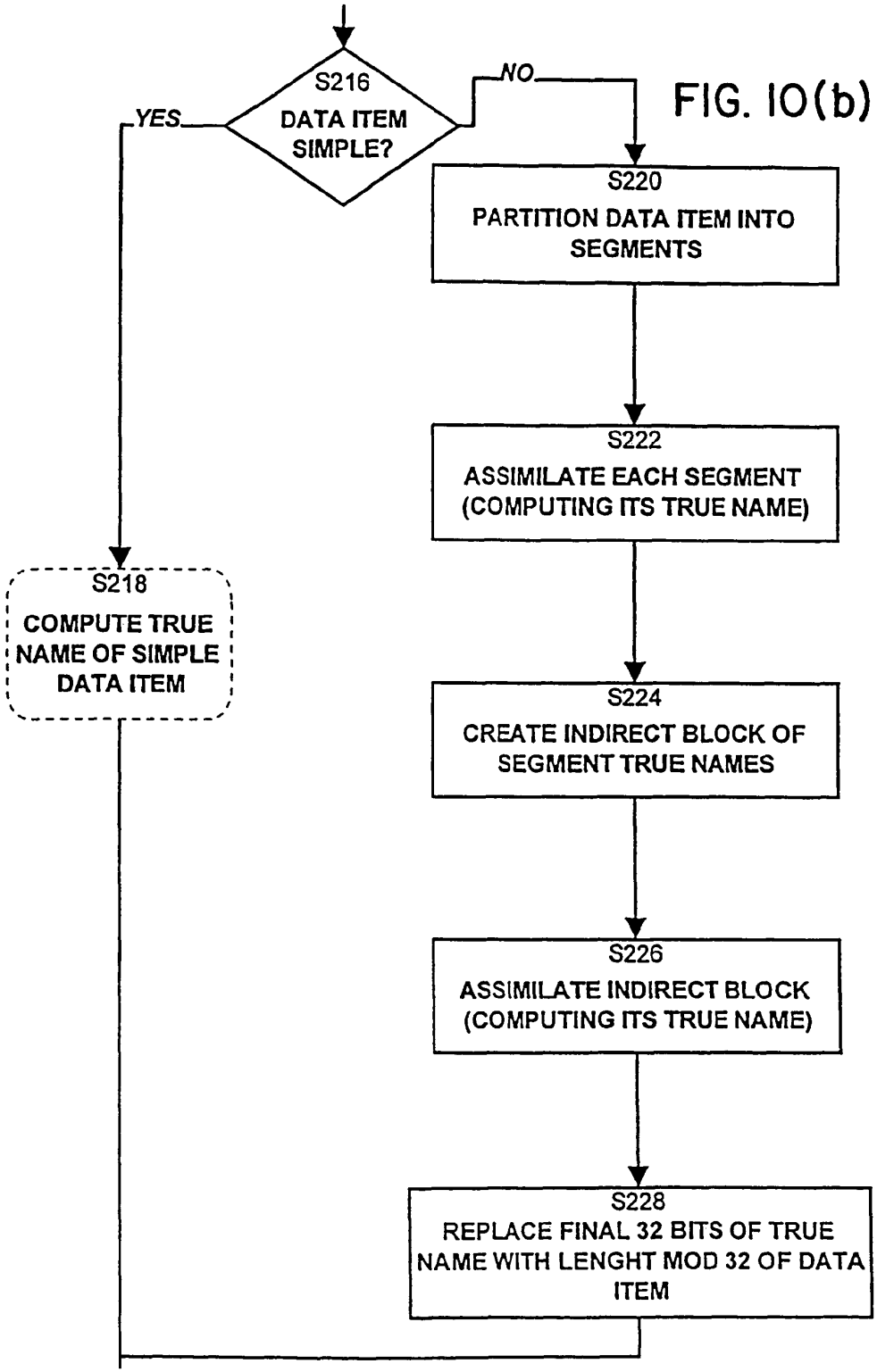

S216
DATA ITEM SIMPLE?
YES
NO
S218
COMPUTE TRUE NAME OF SIMPLE DATA ITEM
S220
PARTITION DATA ITEM INTO SEGMENTS
S222
ASSIMILATE EACH SEGMENT (COMPUTING ITS TRUE NAME)
S224
CREATE INDIRECT BLOCK OF SEGMENT TRUE NAMES
S226
ASSIMILATE INDIRECT BLOCK (COMPUTING ITS TRUE NAME)
S228
REPLACE FINAL 32 BITS OF TRUE NAME WITH LENGHT MOD 32 OF DATA ITEM

FIG. 10(b)

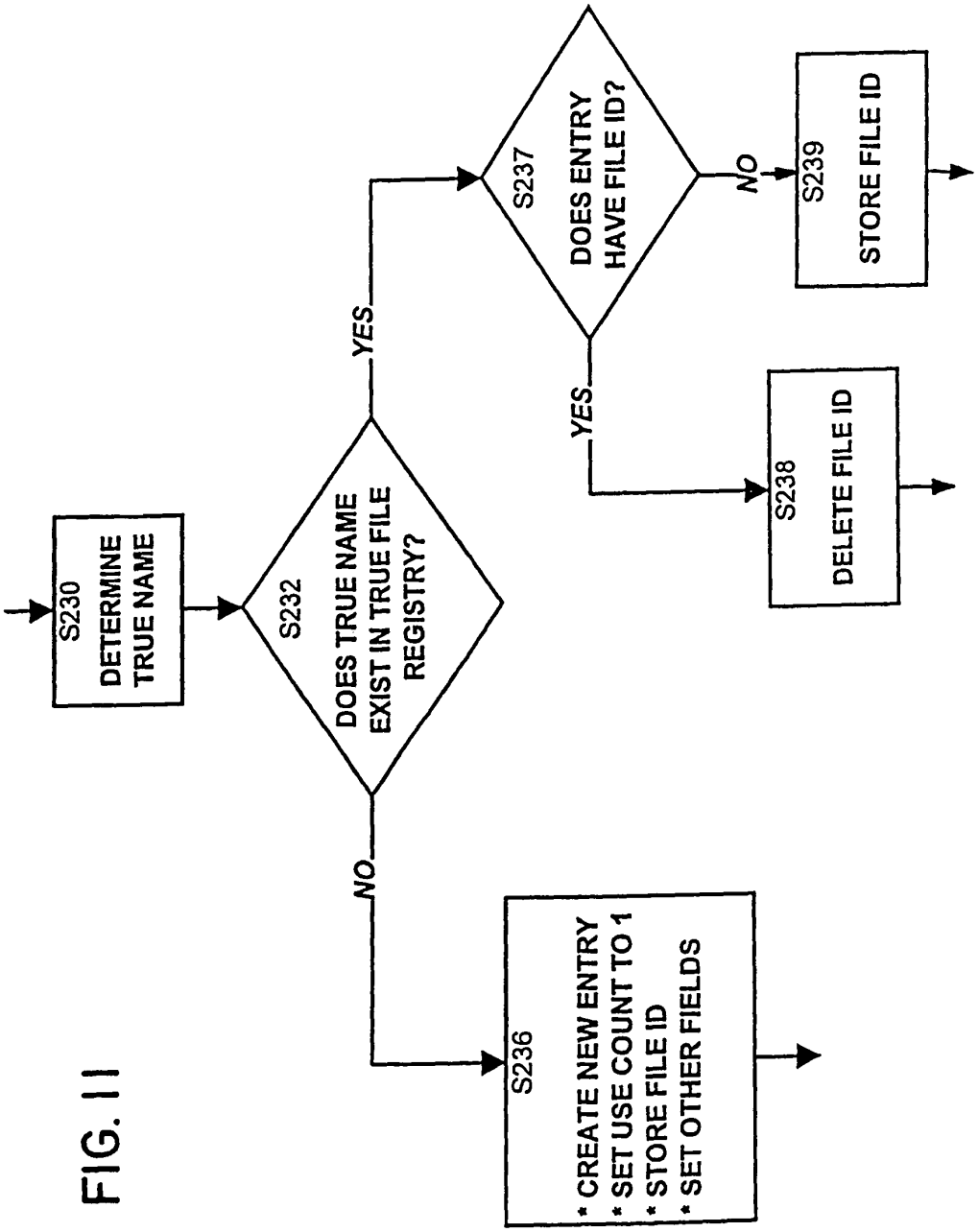

S230
DETERMINE TRUE NAME
S232
DOES TRUE NAME EXIST IN TRUE FILE REGISTRY?
YES
NO
S237
DOES ENTRY HAVE FILE ID?
YES
NO
S238
DELETE FILE ID
S239
STORE FILE ID
S236
* CREATE NEW ENTRY
* SET USE COUNT TO 1
* STORE FILE ID
* SET OTHER FIELDS

FIG. 11

FIG. 12

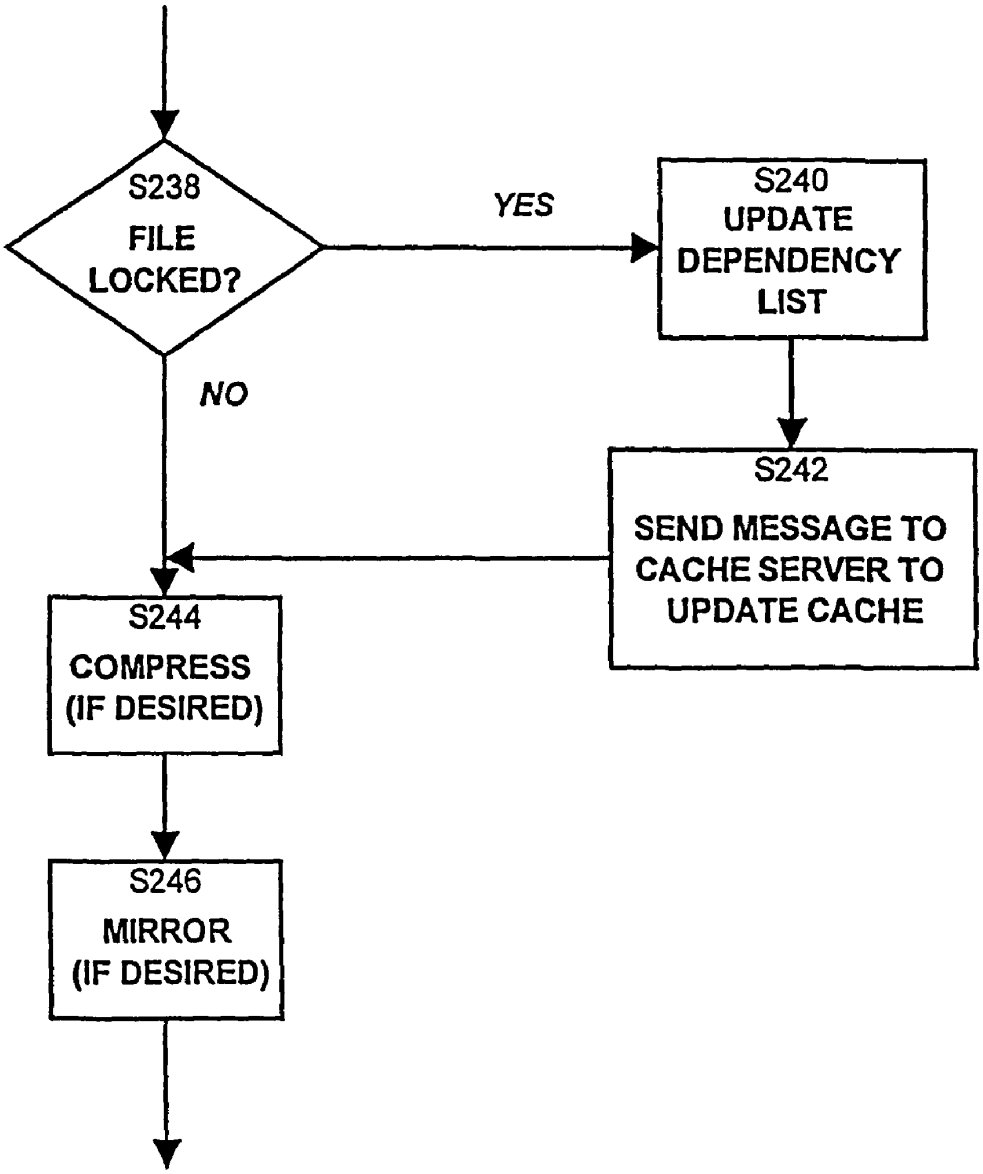
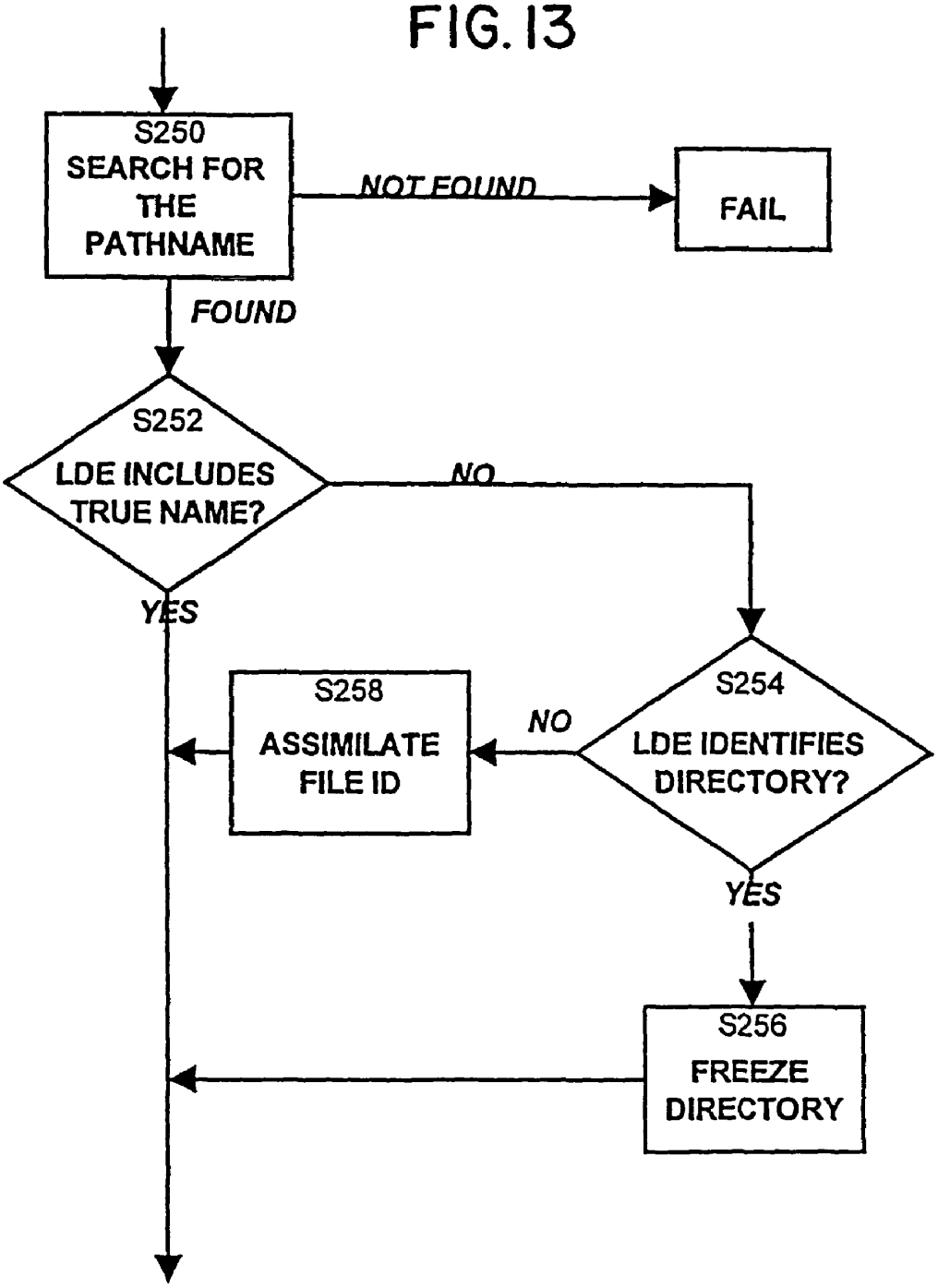

S238
FILE LOCKED?
YES
S240
UPDATE DEPENDENCY LIST
NO
S242
SEND MESSAGE TO CACHE SERVER TO UPDATE CACHE
S244
COMPRESS (IF DESIRED)
S246
MIRROR (IF DESIRED)

FIG. 13


S250
SEARCH FOR THE PATHNAME
NOT FOUND
FAIL
FOUND
S252
LDE INCLUDES TRUE NAME?
NO
YES
S254
LDE IDENTIFIES DIRECTORY?
NO
S258
ASSIMILATE FILE ID
YES
S256
FREEZE DIRECTORY

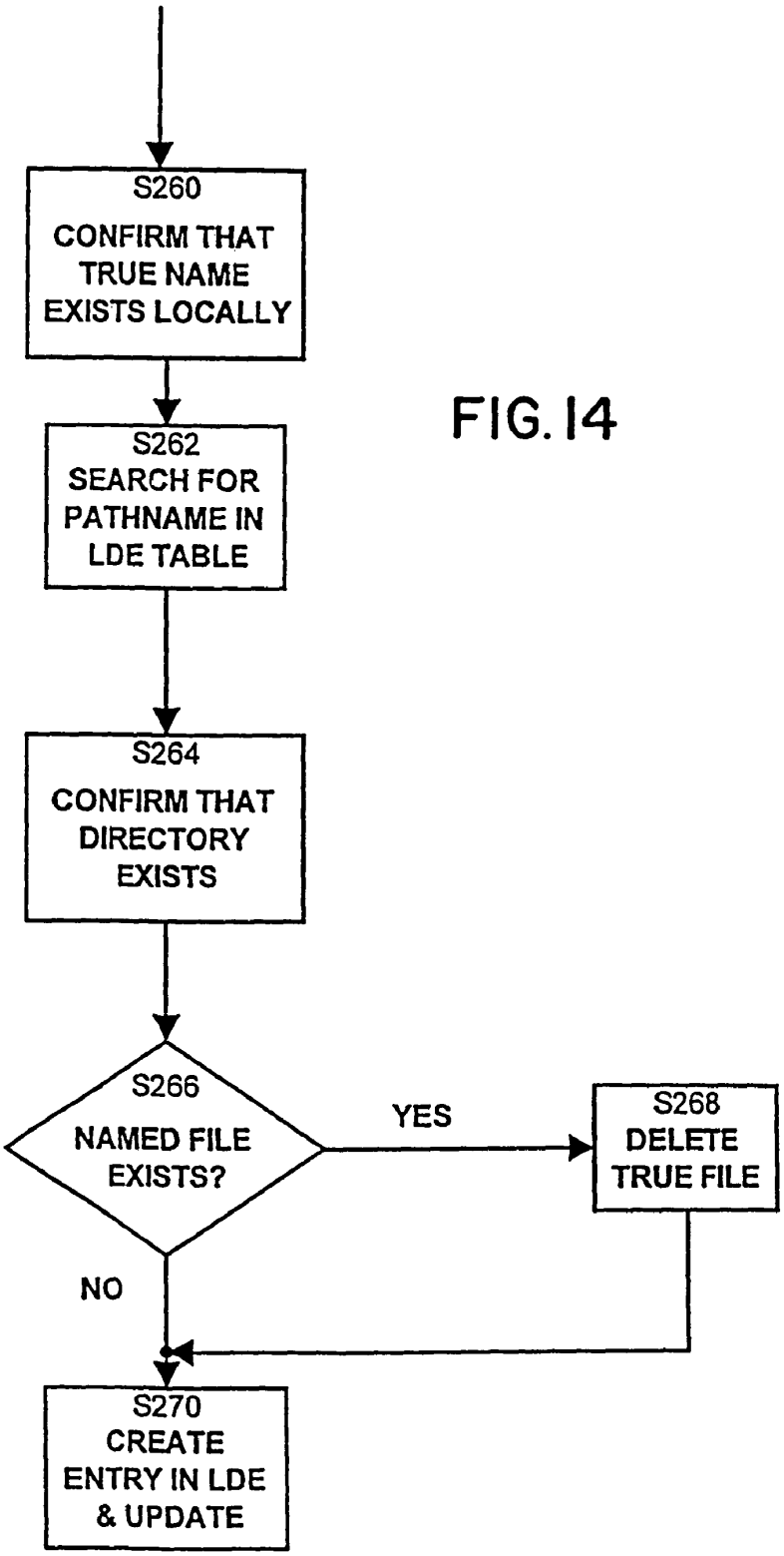

S260
CONFIRM THAT TRUE NAME EXISTS LOCALLY
S262
SEARCH FOR PATHNAME IN LDE TABLE
S264
CONFIRM THAT DIRECTORY EXISTS
S266
NAMED FILE EXISTS?
YES
S268
DELETE TRUE FILE
NO
S270
CREATE ENTRY IN LDE & UPDATE

FIG. 14

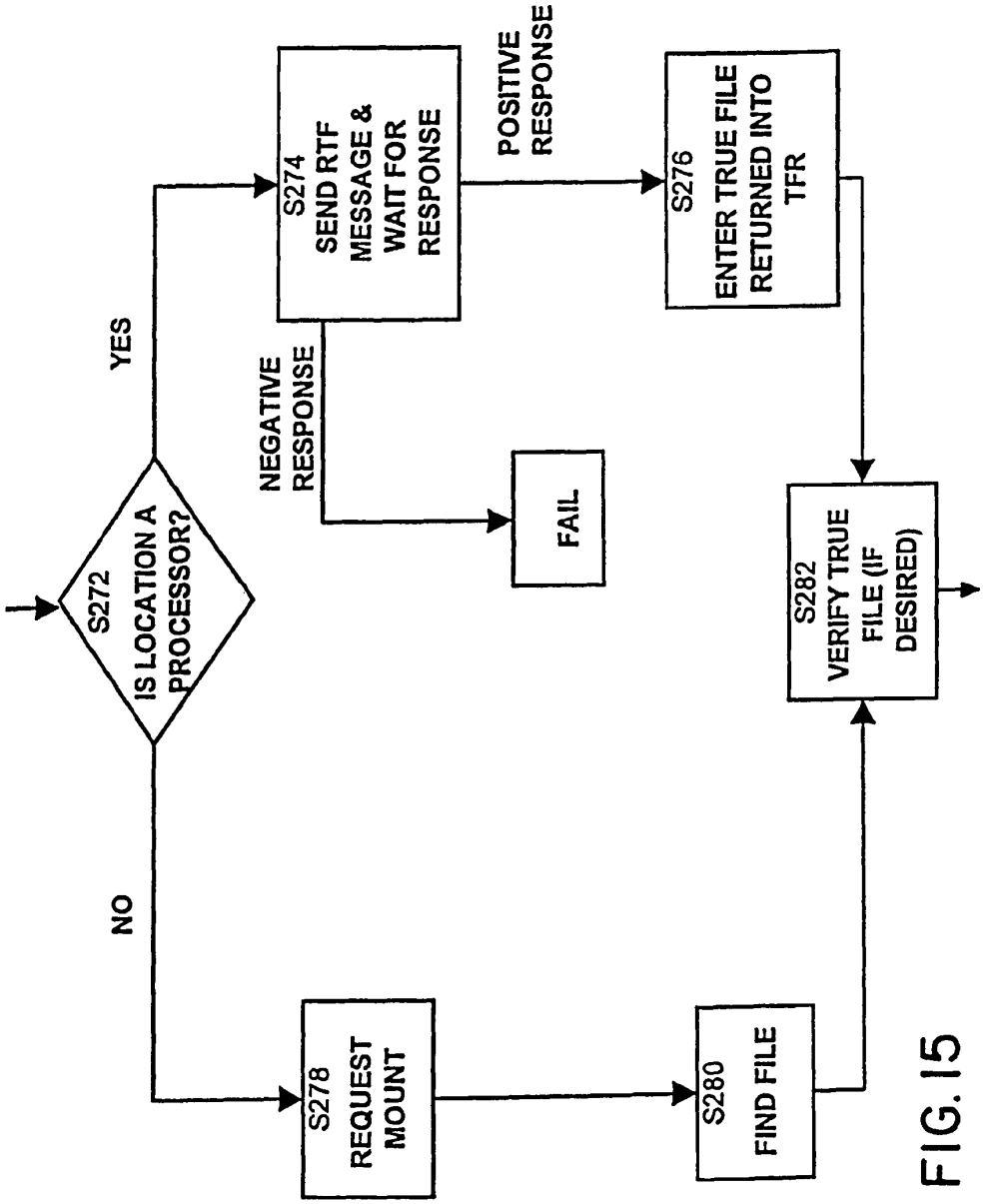

S272
IS LOCATION A PROCESSOR?
YES
NO
S274
SEND RTF MESSAGE & WAIT FOR RESPONSE
POSITIVE RESPONSE
NEGATIVE RESPONSE
FAIL
S276
ENTER TRUE FILE RETURNED INTO TFR
S278
REQUEST MOUNT
S280
FIND FILE
S282
VERIFY TRUE FILE (IF DESIRED)

FIG. 15

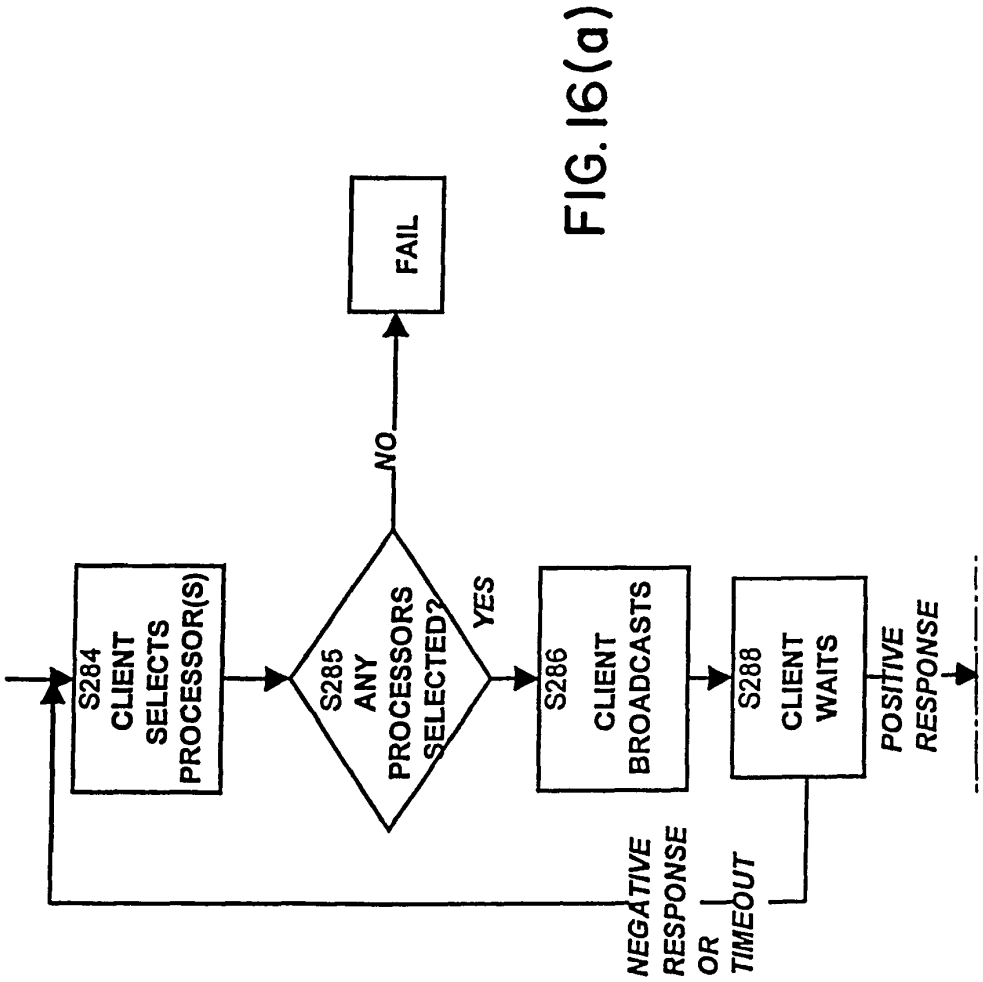

S284
CLIENT
SELECTS
PROCESSOR(S)
S285
ANY
PROCESSORS
SELECTED?
NO
FAIL
YES
S286
CLIENT
BROADCASTS
S288
CLIENT
WAITS
POSITIVE
RESPONSE
NEGATIVE
RESPONSE
OR
TIMEOUT

FIG. 16(a)

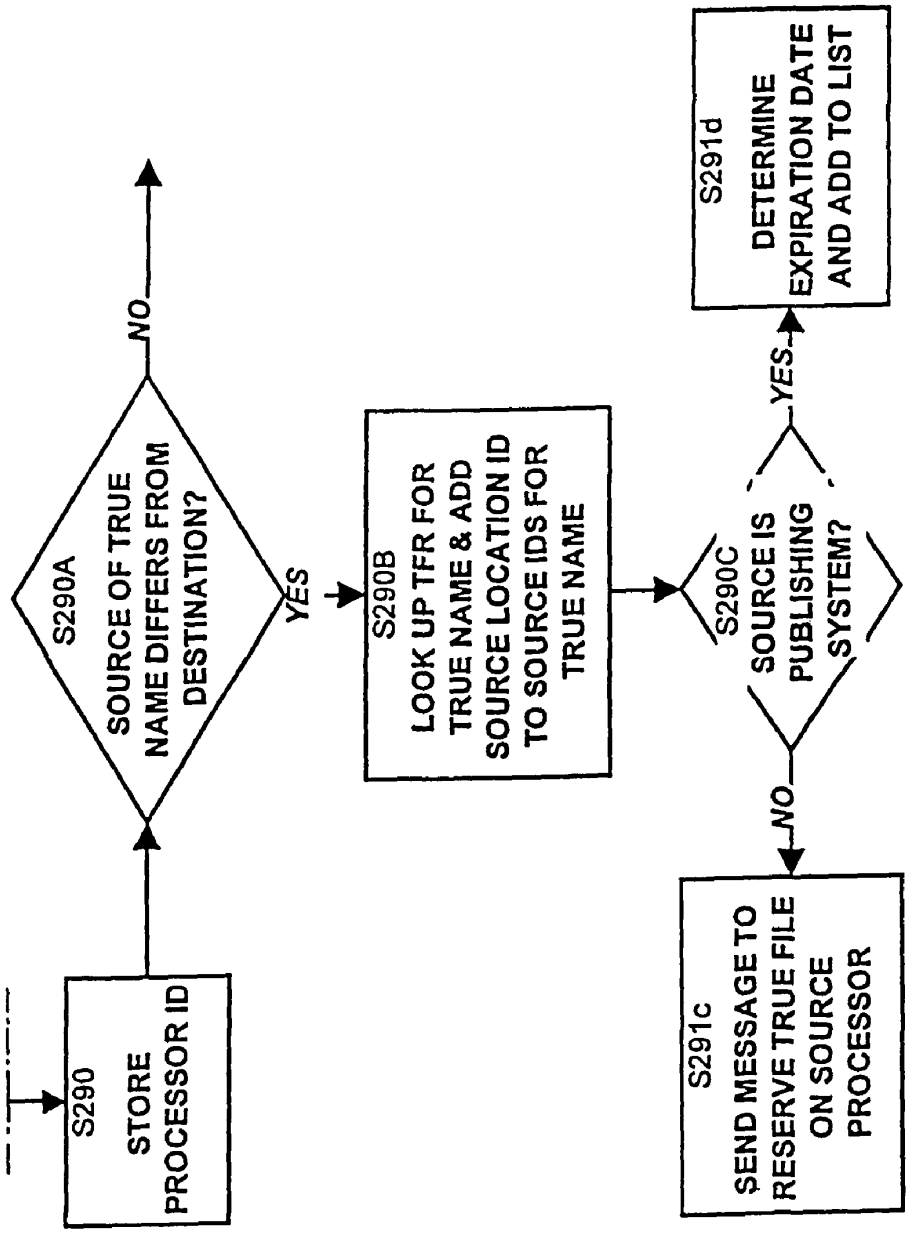

S290
STORE PROCESSOR ID
S290A
SOURCE OF TRUE NAME DIFFERS FROM DESTINATION?
NO
YES
S290B
LOOK UP TFR FOR TRUE NAME & ADD SOURCE LOCATION ID TO SOURCE IDS FOR TRUE NAME
S290C
SOURCE IS PUBLISHING SYSTEM?
YES
NO
S291d
DETERMINE EXPIRATION DATE AND ADD TO LIST
S291c
SEND MESSAGE TO RESERVE TRUE FILE ON SOURCE PROCESSOR

FIG. 16(b)

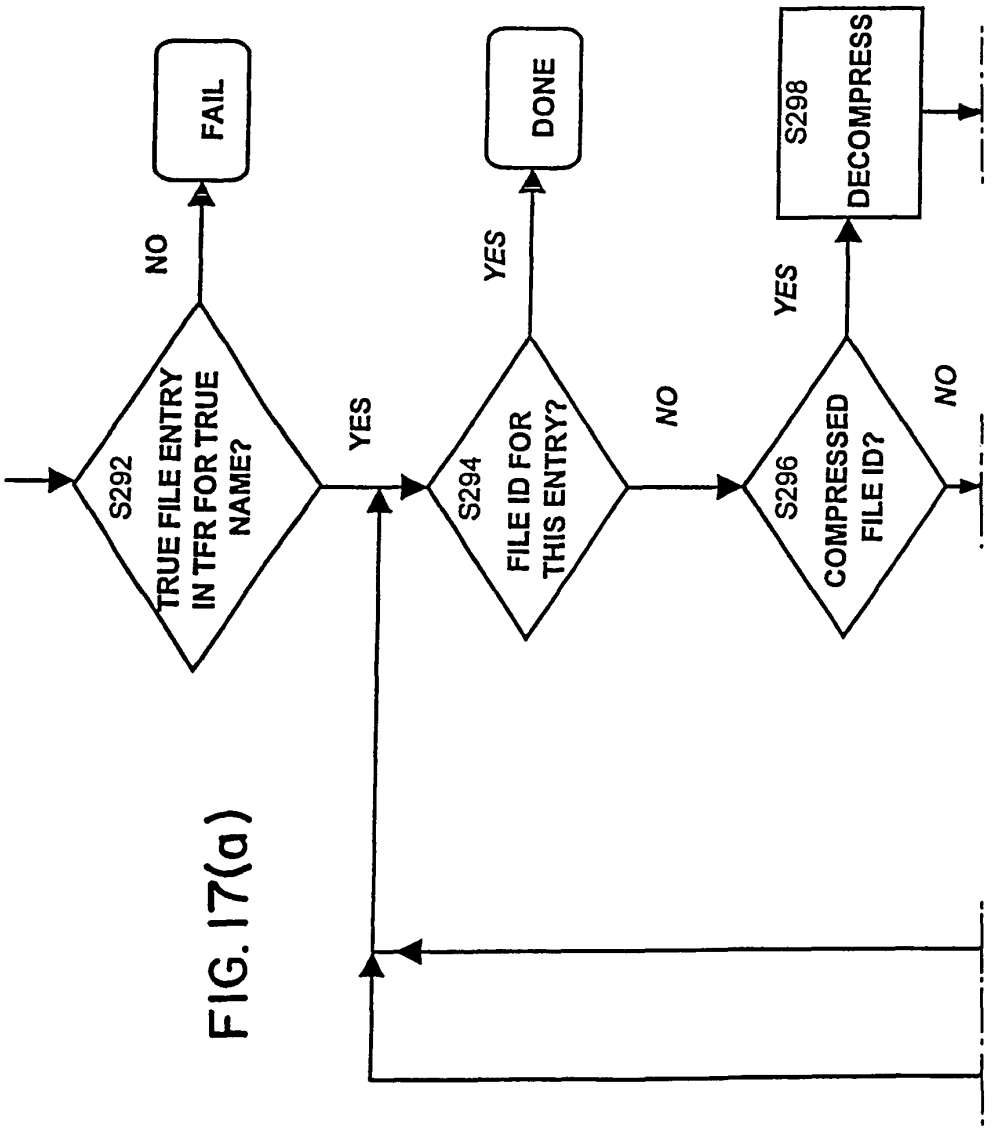

S292
TRUE FILE ENTRY IN TFR FOR TRUE NAME?
NO
FAIL
YES
S294
FILE ID FOR THIS ENTRY?
YES
DONE
NO
S296
COMPRESSED FILE ID?
YES
S298
DECOMPRESS
NO

FIG. 17(a)

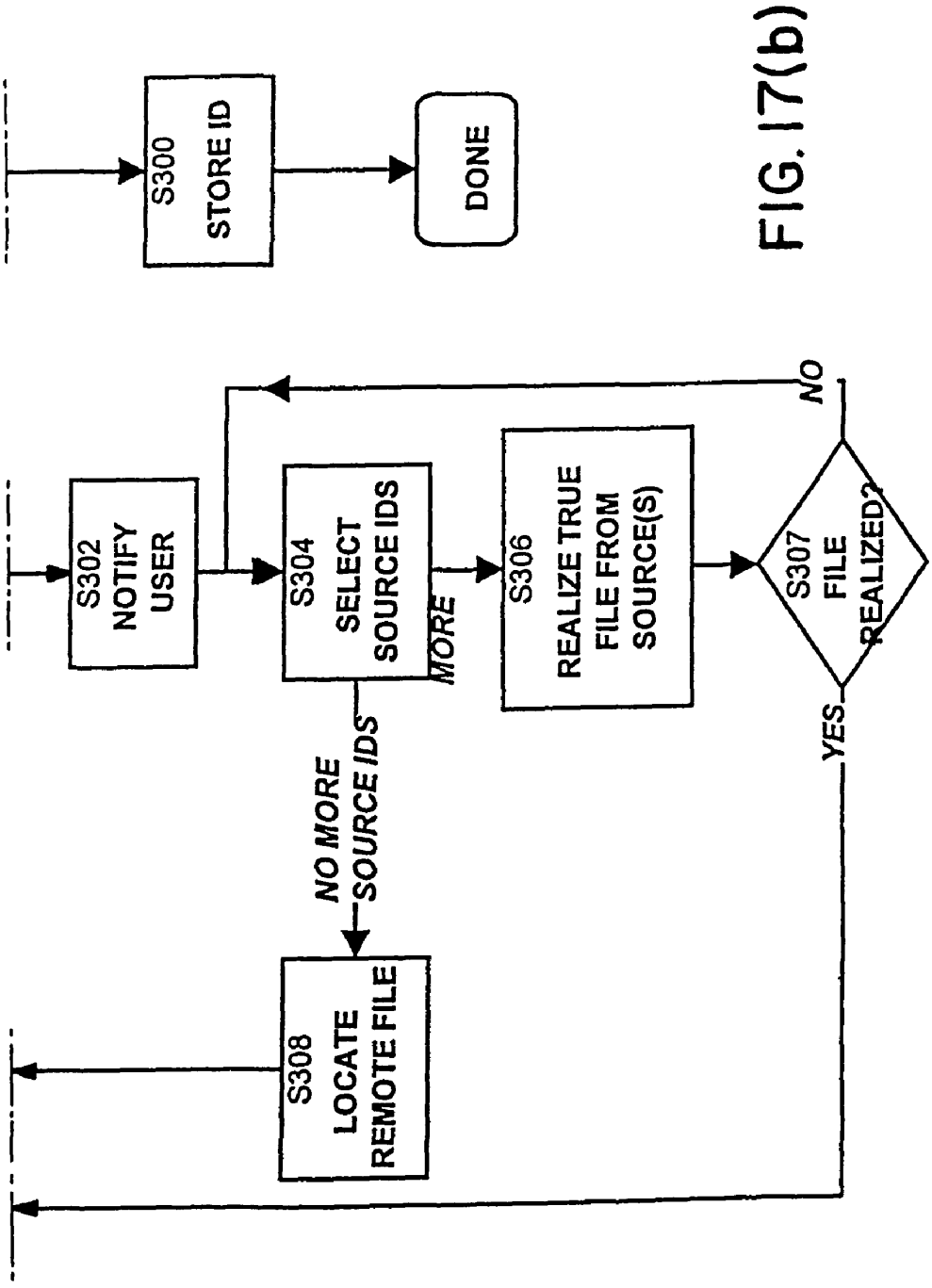
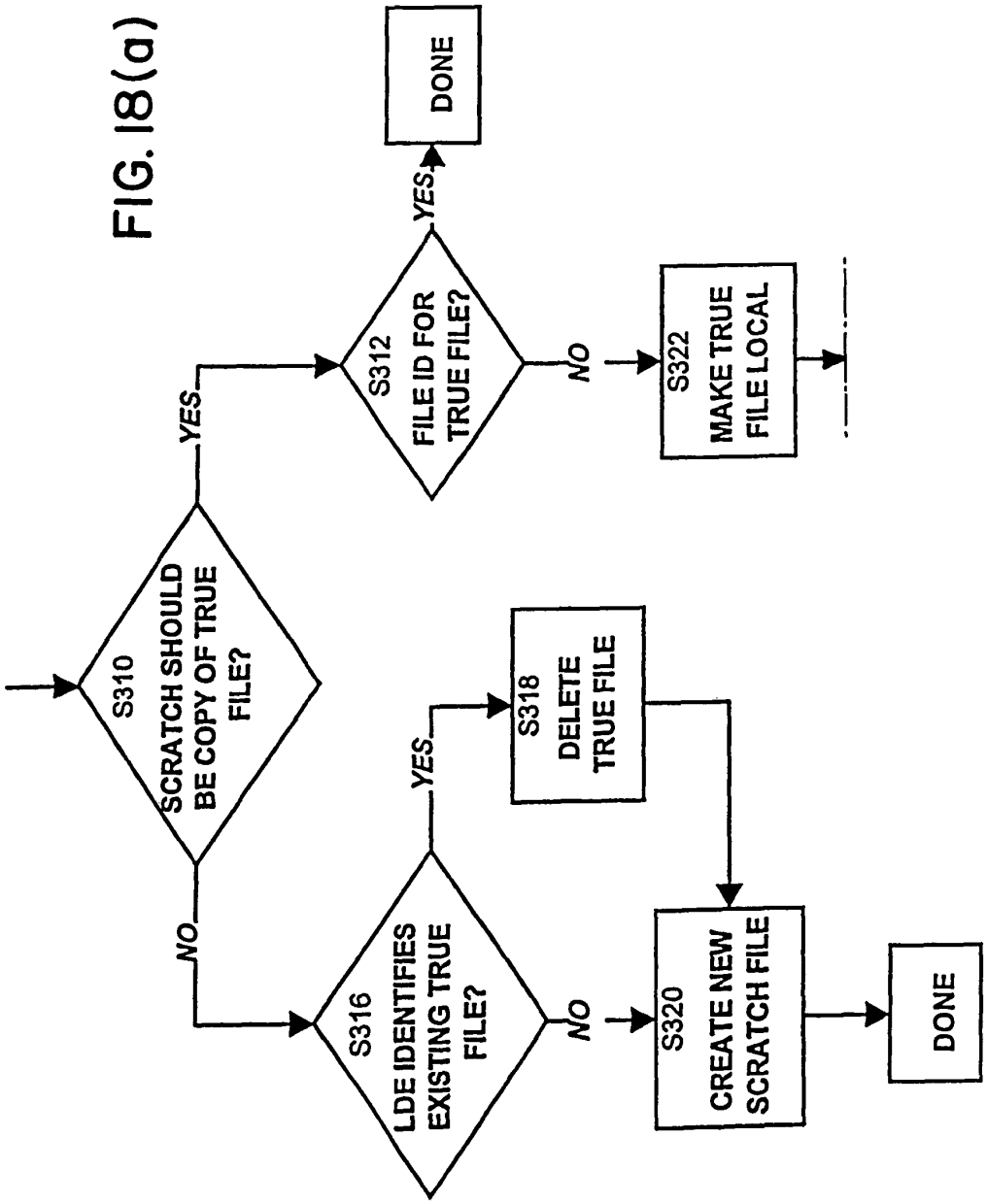

S300
STORE ID
DONE
S302
NOTIFY USER
S304
SELECT SOURCE IDS
MORE
S306
REALIZE TRUE FILE FROM SOURCE(S)
S307
FILE REALIZED?
NO
YES
NO MORE SOURCE IDS
S308
LOCATE REMOTE FILE

FIG. 17(b)

FIG. 18(a)


S310
SCRATCH SHOULD BE COPY OF TRUE FILE?
YES
NO
S312
FILE ID FOR TRUE FILE?
YES
NO
DONE
S322
MAKE TRUE FILE LOCAL
S316
LDE IDENTIFIES EXISTING TRUE FILE?
YES
NO
S318
DELETE TRUE FILE
S320
CREATE NEW SCRATCH FILE
DONE

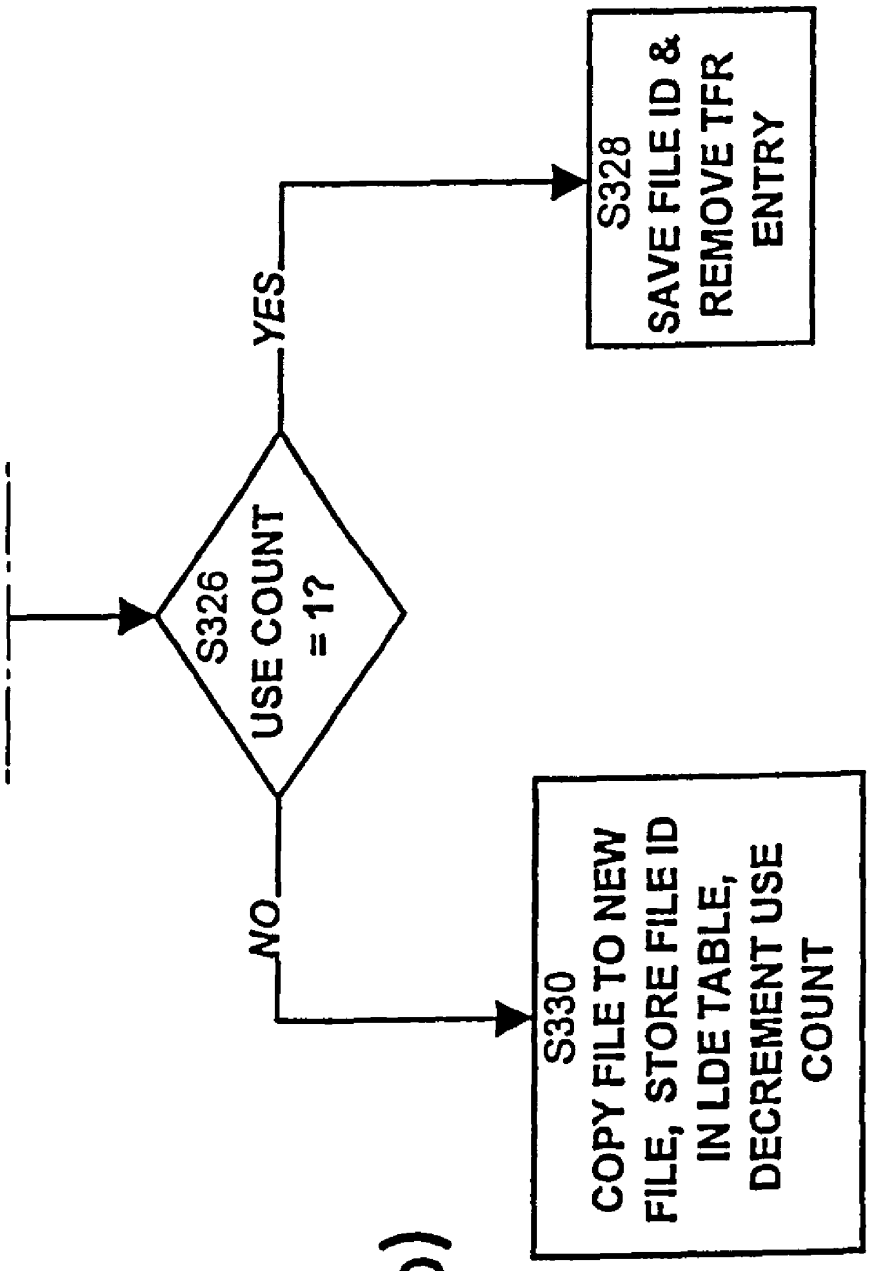
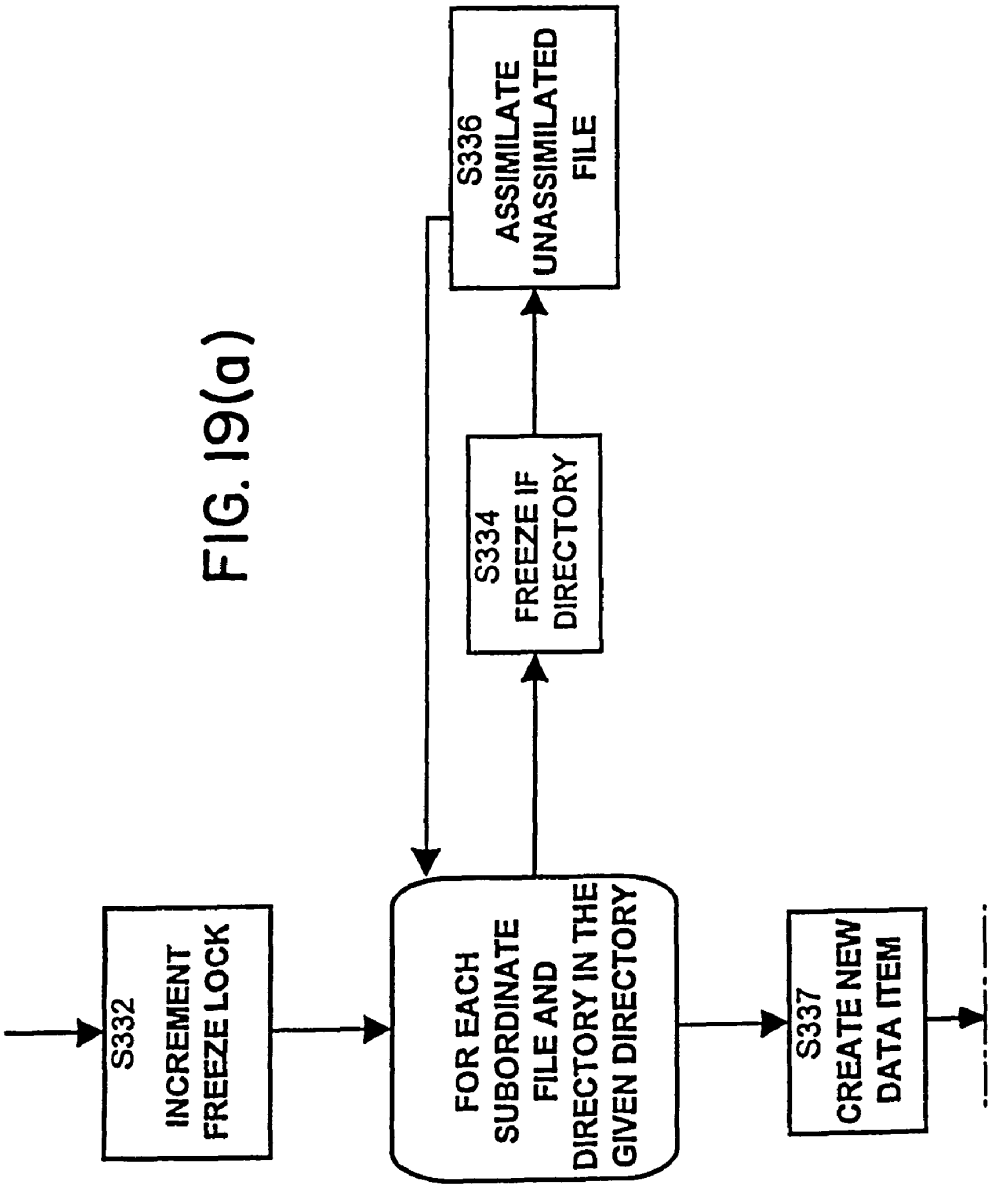

S326
USE COUNT
= 1?
YES
NO
S328
SAVE FILE ID &
REMOVE TFR
ENTRY
S330
COPY FILE TO NEW
FILE, STORE FILE ID
IN LDE TABLE,
DECREMENT USE
COUNT

FIG. 18(b)

FIG. 19(a)


S332
INCREMENT FREEZE LOCK
FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S334
FREEZE IF DIRECTORY
S336
ASSIMILATE UNASSIMILATED FILE
S337
CREATE NEW DATA ITEM

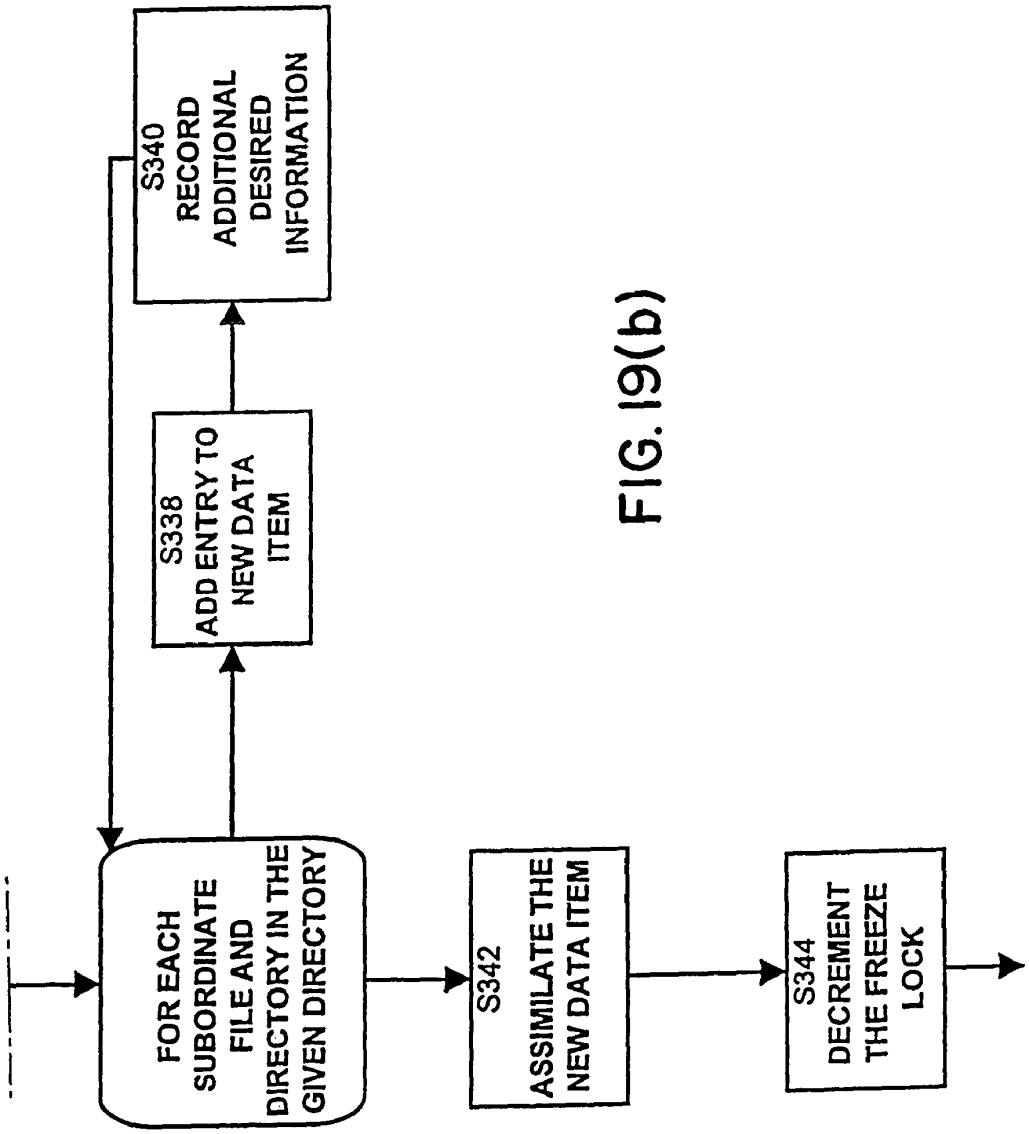

FOR EACH SUBORDINATE FILE AND DIRECTORY IN THE GIVEN DIRECTORY
S338 ADD ENTRY TO NEW DATA ITEM
S340 RECORD ADDITIONAL DESIRED INFORMATION
S342 ASSIMILATE THE NEW DATA ITEM
S344 DECREMENT THE FREEZE LOCK

FIG. 19(b)

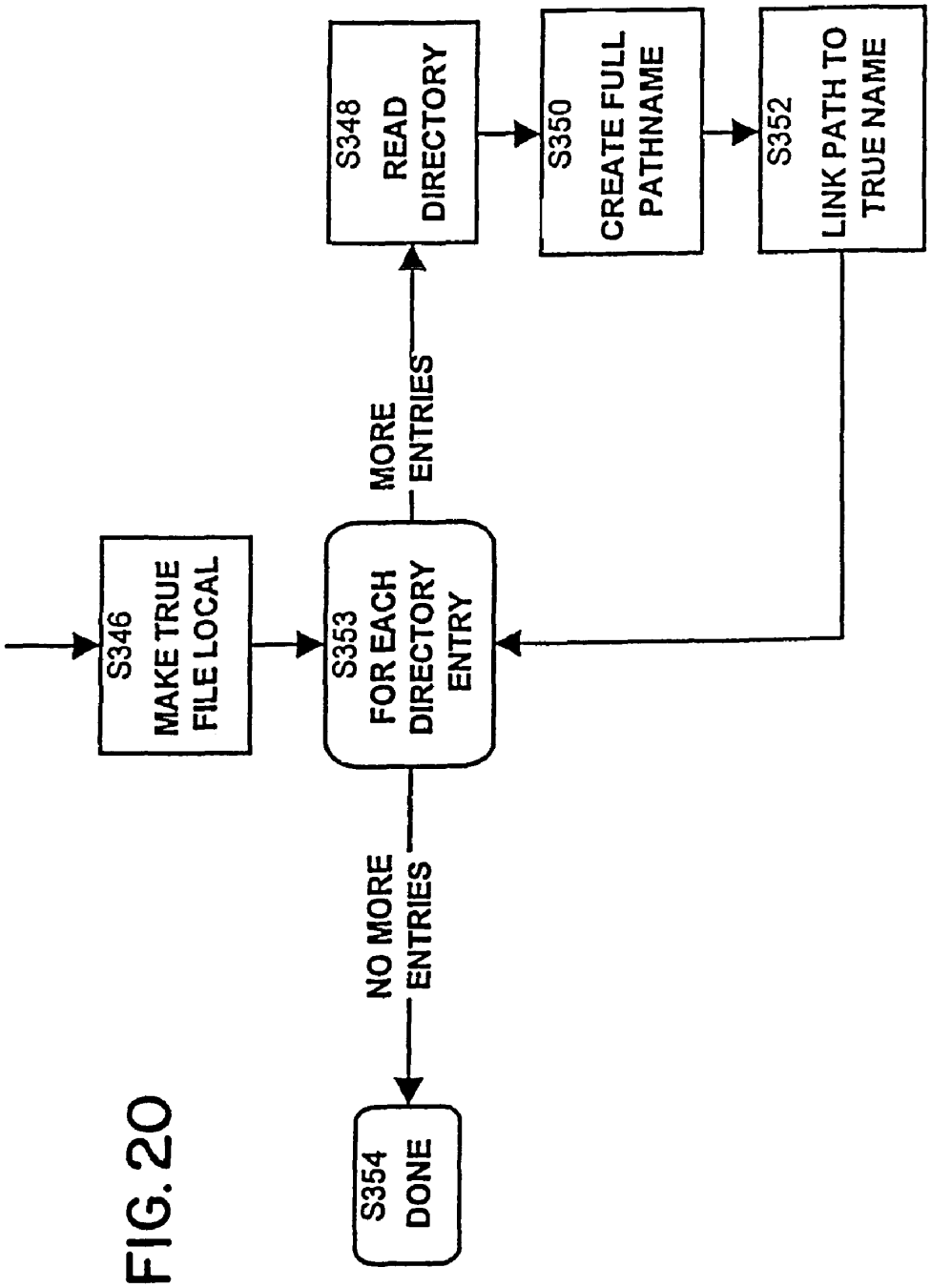

S346
MAKE TRUE FILE LOCAL
S353
FOR EACH DIRECTORY ENTRY
MORE ENTRIES
S348
READ DIRECTORY
S350
CREATE FULL PATHNAME
S352
LINK PATH TO TRUE NAME
NO MORE ENTRIES
S354
DONE

FIG. 20

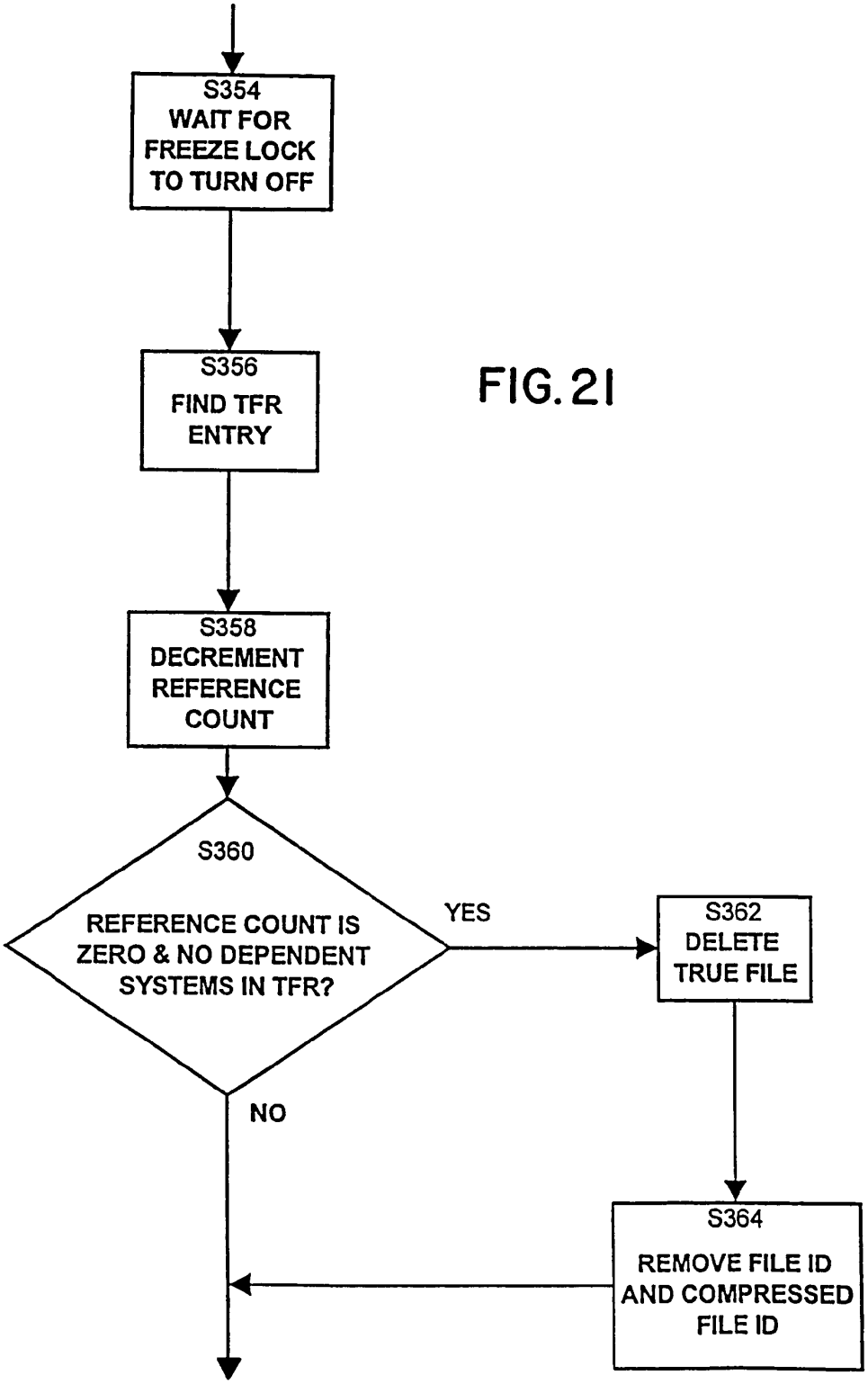

S354
WAIT FOR FREEZE LOCK TO TURN OFF
S356
FIND TFR ENTRY
S358
DECREMENT REFERENCE COUNT
S360
REFERENCE COUNT IS ZERO & NO DEPENDENT SYSTEMS IN TFR?
YES
NO
S362
DELETE TRUE FILE
S364
REMOVE FILE ID AND COMPRESSED FILE ID

FIG. 21

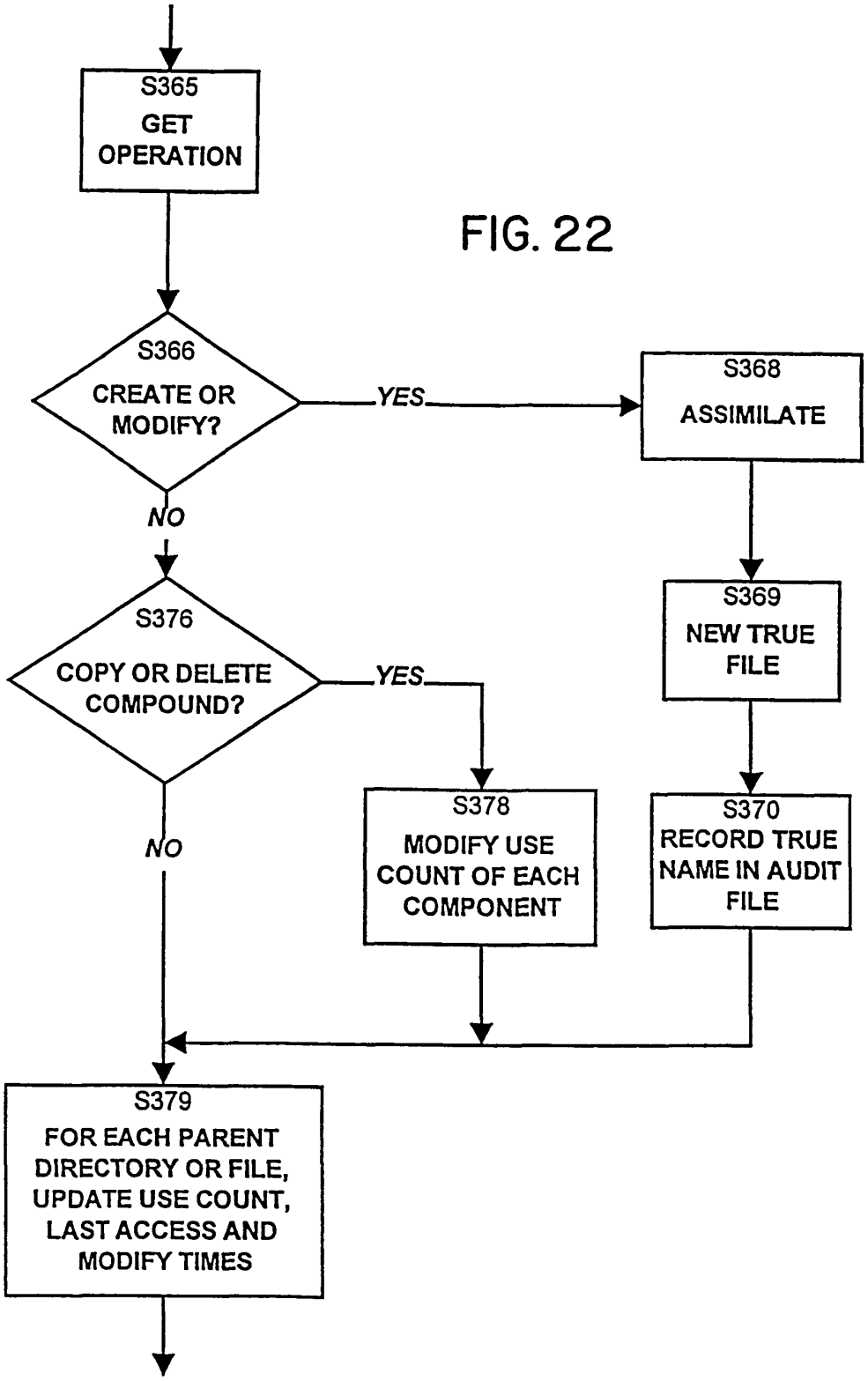
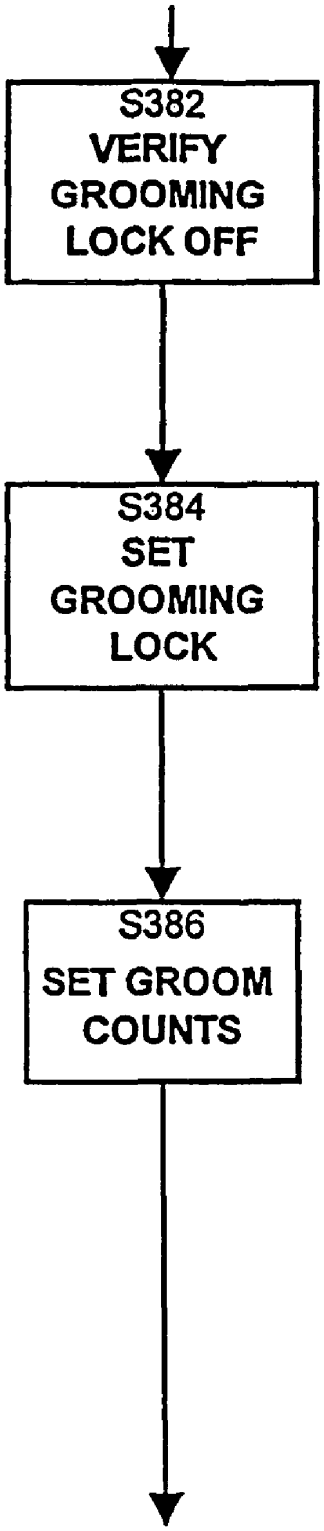

S365
GET OPERATION
S366
CREATE OR MODIFY?
YES
S368
ASSIMILATE
NO
S369
NEW TRUE FILE
S376
COPY OR DELETE COMPOUND?
YES
S378
MODIFY USE COUNT OF EACH COMPONENT
S370
RECORD TRUE NAME IN AUDIT FILE
NO
S379
FOR EACH PARENT DIRECTORY OR FILE, UPDATE USE COUNT, LAST ACCESS AND MODIFY TIMES

FIG. 22

FIG. 23


S382
VERIFY
GROOMING
LOCK OFF
S384
SET
GROOMING
LOCK
S386
SET GROOM
COUNTS

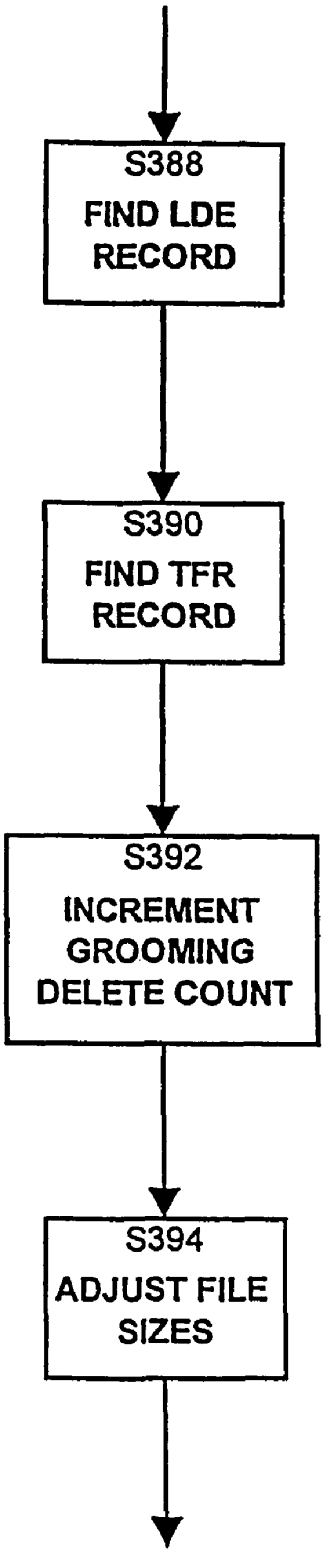

S388
FIND LDE RECORD
S390
FIND TFR RECORD
S392
INCREMENT GROOMING DELETE COUNT
S394
ADJUST FILE SIZES

FIG. 24

FIG. 25

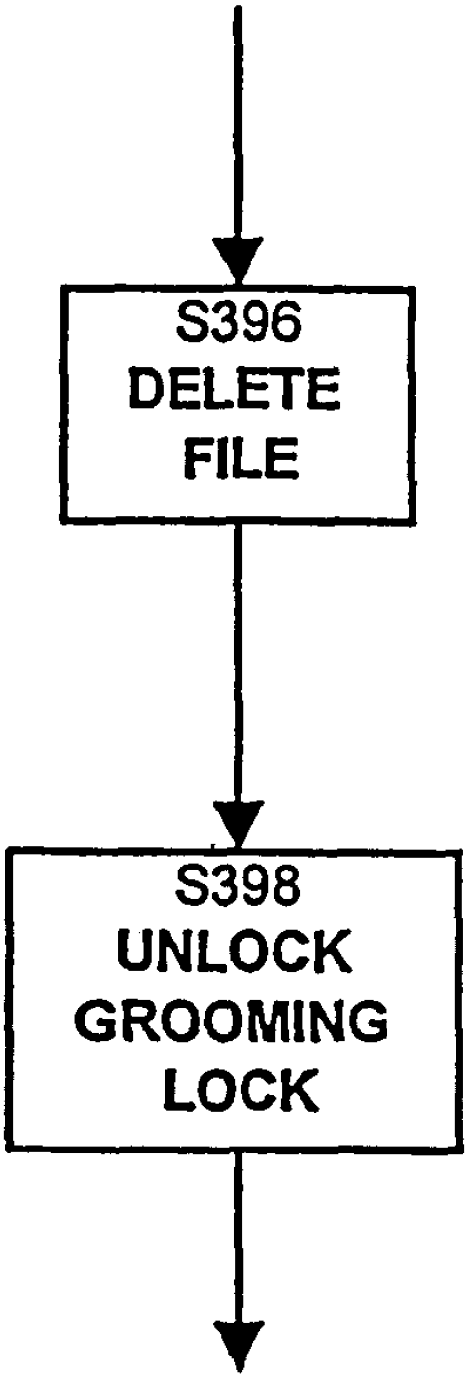
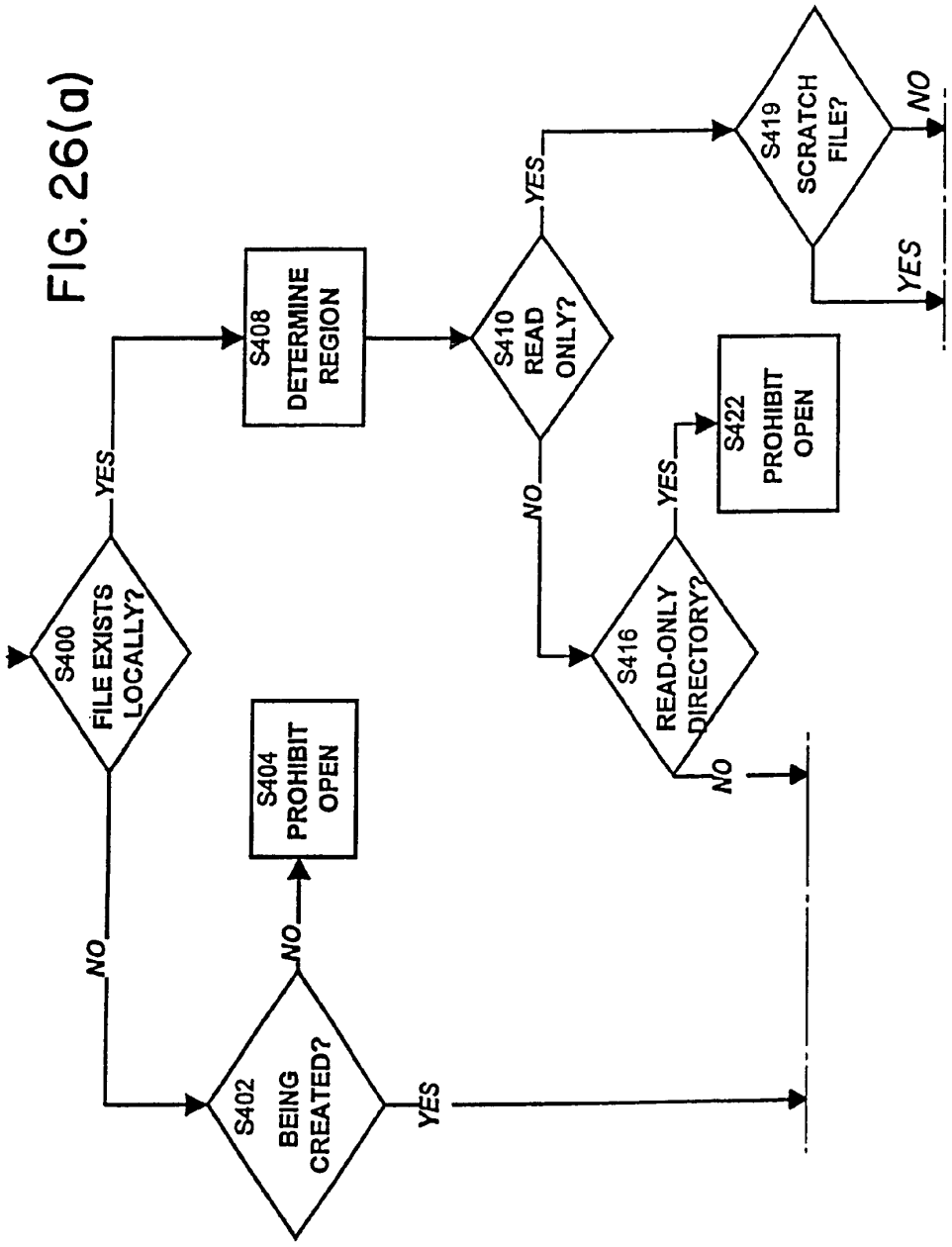


FIG. 26(a)


S400
FILE EXISTS LOCALLY?
YES
NO
S402
BEING CREATED?
YES
NO
S404
PROHIBIT OPEN
S408
DETERMINE REGION
S410
READ ONLY?
YES
NO
S416
READ-ONLY DIRECTORY?
YES
NO
S422
PROHIBIT OPEN
S419
SCRATCH FILE?
YES
NO

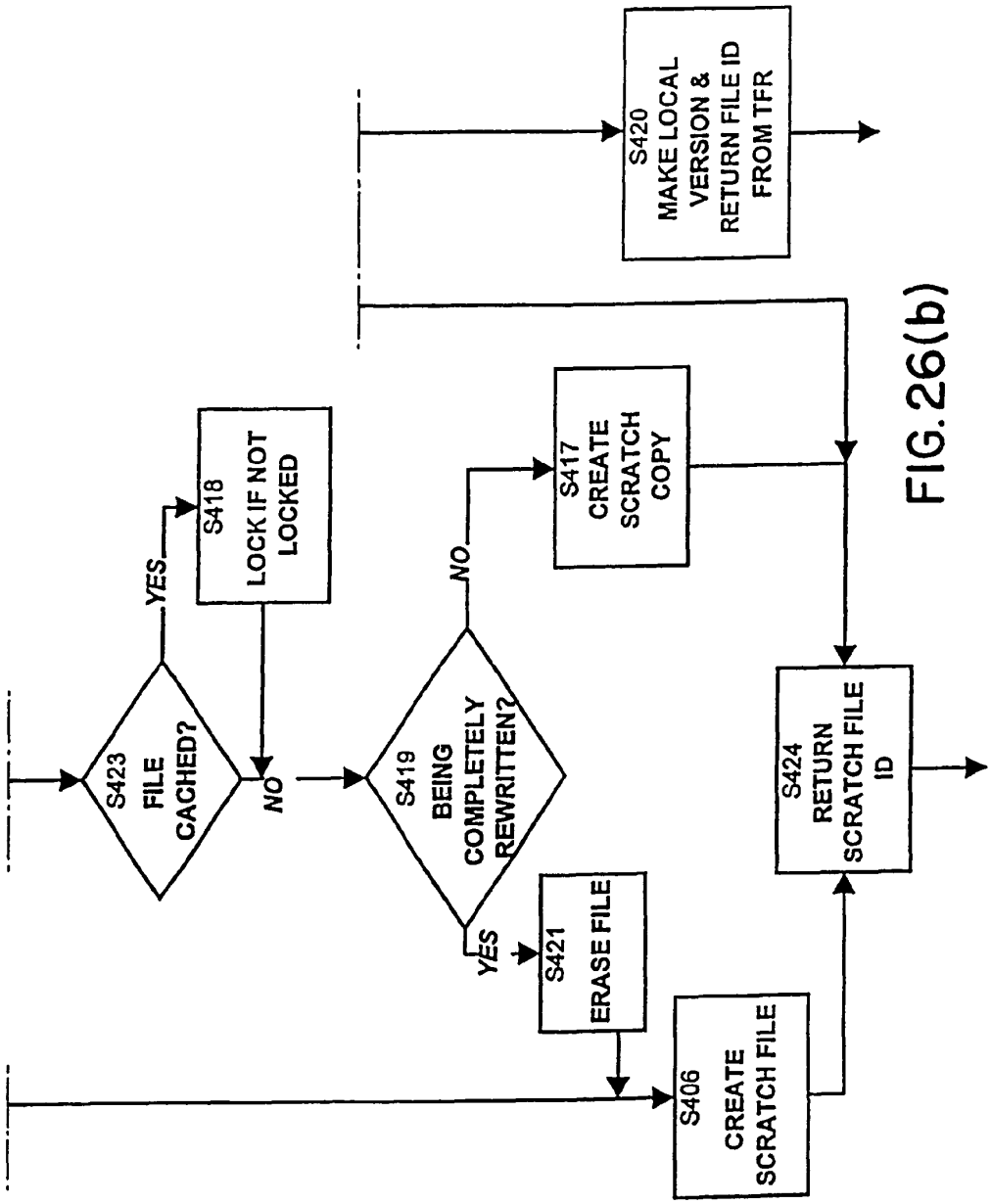

S420
MAKE LOCAL VERSION & RETURN FILE ID FROM TFR
S423
FILE CACHED?
YES
S418
LOCK IF NOT LOCKED
NO
S419
BEING COMPLETELY REWRITTEN?
NO
S417
CREATE SCRATCH COPY
YES
S421
ERASE FILE
S406
CREATE SCRATCH FILE
S424
RETURN SCRATCH FILE ID

FIG. 26(b)

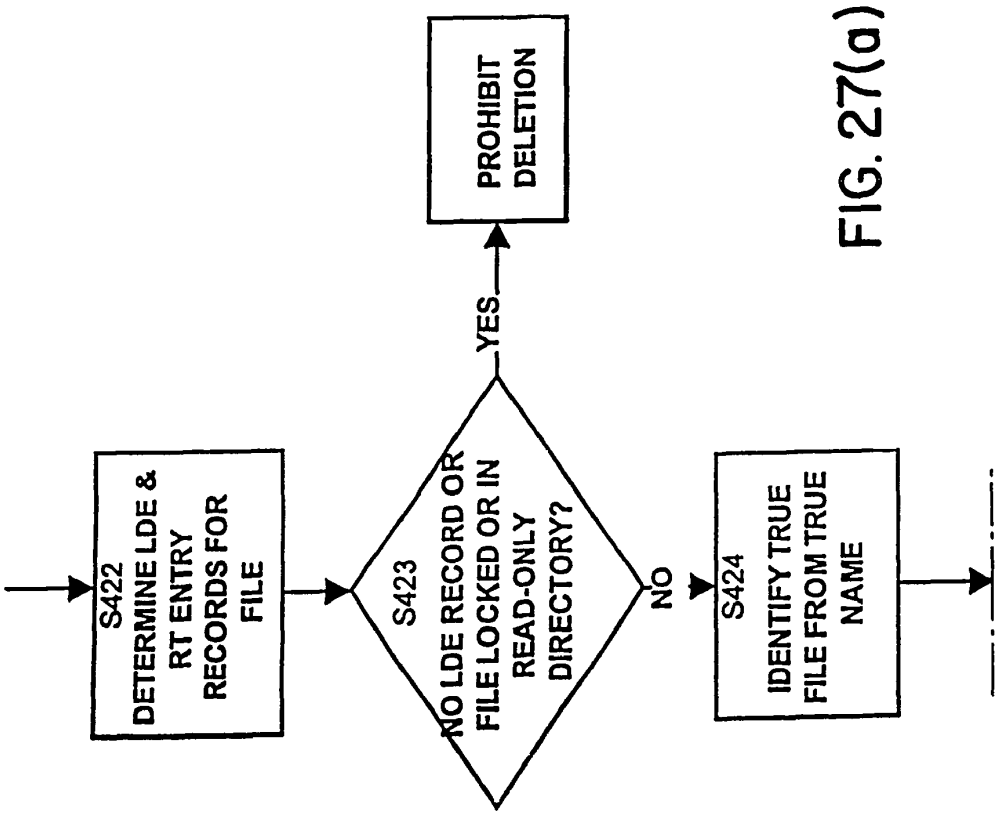

S422
DETERMINE LDE & RT ENTRY RECORDS FOR FILE
S423
NO LDE RECORD OR FILE LOCKED OR IN READ-ONLY DIRECTORY?
YES
PROHIBIT DELETION
NO
S424
IDENTIFY TRUE FILE FROM TRUE NAME

FIG. 27(a)

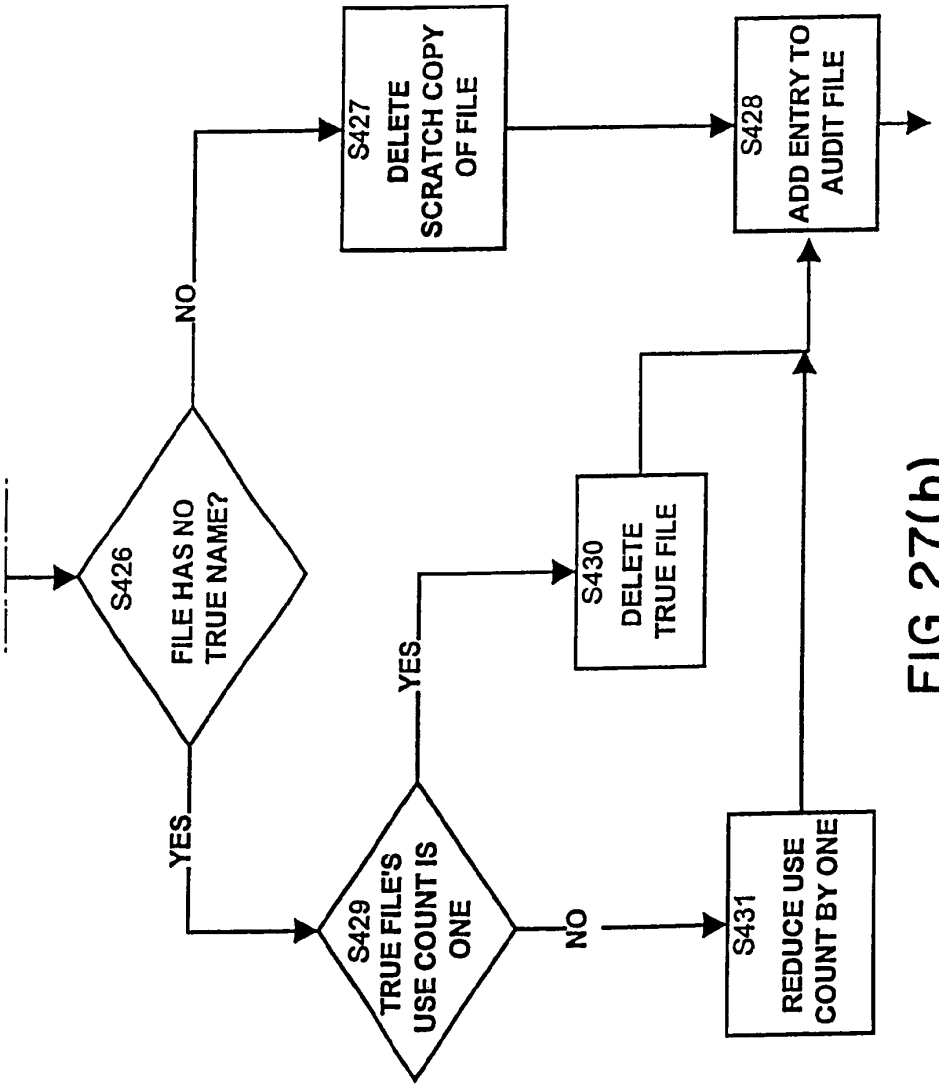
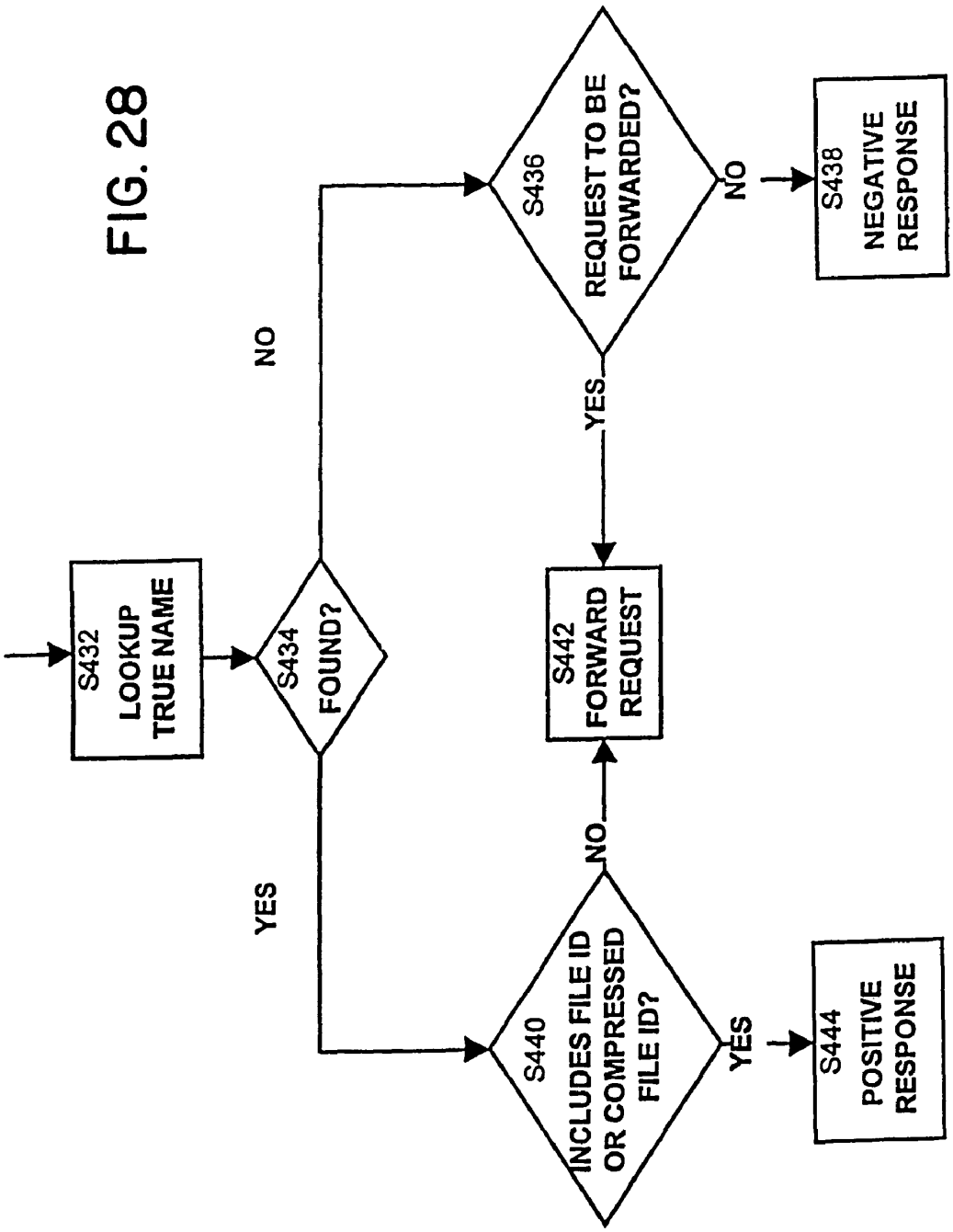

S426
FILE HAS NO TRUE NAME?
NO
YES
S427
DELETE SCRATCH COPY OF FILE
S428
ADD ENTRY TO AUDIT FILE
S429
TRUE FILE'S USE COUNT IS ONE
YES
NO
S430
DELETE TRUE FILE
S431
REDUCE USE COUNT BY ONE

FIG. 27(b)

FIG. 28


S432
LOOKUP TRUE NAME
S434
FOUND?
NO
YES
S436
REQUEST TO BE FORWARDED?
NO
S438
NEGATIVE RESPONSE
YES
S442
FORWARD REQUEST
S440
INCLUDES FILE ID OR COMPRESSED FILE ID?
NO
YES
S444
POSITIVE RESPONSE

ACCESSING DATA IN A DATA PROCESSING SYSTEM

RELATED APPLICATIONS

This is a continuation of and claims priority to application Ser. no. 09/987,723, filed Nov. 15, 2001, now U.S. Pat. No. 6,928,442, issued Aug. 9, 2005 (the contents of which are hereby incorporated herein by reference), which is a continuation of application Ser. No. 09/283,160, filed Apr. 1, 1999, now U.S. Pat. No. 6,415,280, which is a division of application Ser. No. 08/960,079, filed Oct. 24, 1997, now U.S. Pat. No. 5,978,791 filed Oct. 24, 2001 which is a continuation of Ser. No. 08/425,160, filed Apr. 11, 1995, now abandoned.

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to data processing systems and, more particularly, to data processing systems wherein data items are identified by substantially unique identifiers which depend on all of the data in the data items and only on the data in the data items.

2. Background of the Invention

Data processing (DP) systems, computers, networks of computers, or the like, typically offer users and programs various ways to identify the data in the systems.

Users typically identify data in the data processing system by giving the data some form of name. For example, a typical operating system (OS) on a computer provides a file system in which data items are named by alphanumeric identifiers. Programs typically identify data in the data processing system using a location or address. For example, a program may identify a record in a file or database by using a record number which serves to locate that record.

In all but the most primitive operating systems, users and programs are able to create and use collections of named data items, these collections themselves being named by identifiers. These named collections can then, themselves, be made part of other named collections. For example, an OS may provide mechanisms to group files (data items) into directories (collections). These directories can then, themselves be made part of other directories. A data item may thus be identified relative to these nested directories using a sequence of names, or a so-called pathname, which defines a path through the directories to a particular data item (file or directory).

As another example, a database management system may group data records (data items) into tables and then group these tables into database files (collections). The complete address of any data record can then be specified using the database file name, the table name, and the record number of that data record.

Other examples of identifying data items include: identifying files in a network file system, identifying objects in an object-oriented database, identifying images in an image database, and identifying articles in a text database.

In general, the terms "data" and "data item" as used herein refer to sequences of bits. Thus a data item may be the contents of a file, a portion of a file, a page in memory, an object in an object-oriented program, a digital message, a digital scanned image, a part of a video or audio signal, or any other entity which can be represented by a sequence of bits. The term "data processing" herein refers to the processing of data items, and is sometimes dependent on the type of data item being processed. For example, a data processor for a digital image may differ from a data processor for an audio signal.

In all of the prior data processing systems the names or identifiers provided to identify data items (the data items being files, directories, records in the database, objects in object-oriented programming, locations in memory or on a physical device, or the like) are always defined relative to a specific context. For instance, the file identified by a particular file name can only be determined when the directory containing the file (the context) is known. The file identified by a pathname can be determined only when the file system (context) is known. Similarly, the addresses in a process address space, the keys in a database table, or domain names on a global computer network such as the Internet are meaningful only because they are specified relative to a context.

In prior art systems for identifying data items there is no direct relationship between the data names and the data item. The same data name in two different contexts may refer to different data items, and two different data names in the same context may refer to the same data item.

In addition, because there is no correlation between a data name and the data it refers to, there is no a priori way to confirm that a given data item is in fact the one named by a data name. For instance, in a DP system, if one processor requests that another processor deliver a data item with a given data name, the requesting processor cannot, in general, verify that the data delivered is the correct data (given only the name). Therefore it may require further processing, typically on the part of the requester, to verify that the data item it has obtained is, in fact, the item it requested.

A common operation in a DP system is adding a new data item to the system. When a new data item is added to the system, a name can be assigned to it only by updating the context in which names are defined. Thus such systems require a centralized mechanism for the management of names. Such a mechanism is required even in a multi-processing system when data items are created and identified at separate processors in distinct locations, and in which there is no other need for communication when data items are added.

In many data processing systems or environments, data items are transferred between different locations in the system. These locations may be processors in the data processing system, storage devices, memory, or the like. For example, one processor may obtain a data item from another processor or from an external storage device, such as a floppy disk, and may incorporate that data item into its system (using the name provided with that data item).

However, when a processor (or some location) obtains a data item from another location in the DP system, it is possible that this obtained data item is already present in the system (either at the location of the processor or at some other location accessible by the processor) and therefore a duplicate of the data item is created. This situation is common in a network data processing environment where proprietary software products are installed from floppy disks onto several processors sharing a common file server. In these systems, it is often the case that the same product will be installed on several systems, so that several copies of each file will reside on the common file server.

In some data processing systems in which several processors are connected in a network, one system is designated as a cache server to maintain master copies of data items, and other systems are designated as cache clients to copy local copies of the master data items into a local cache on an as-needed basis. Before using a cached item, a cache client must either reload the cached item, be informed of changes to the cached item, or confirm that the master item corresponding to the cached item has not changed. In other words, a cache client must synchronize its data items with those on the

cache server. This synchronization may involve reloading data items onto the cache client. The need to keep the cache synchronized or reload it adds significant overhead to existing caching mechanisms.

In view of the above and other problems with prior art systems, it is therefore desirable to have a mechanism which allows each processor in a multiprocessor system to determine a common and substantially unique identifier for a data item, using only the data in the data item and not relying on any sort of context.

It is further desirable to have a mechanism for reducing multiple copies of data items in a data processing system and to have a mechanism which enables the identification of identical data items so as to reduce multiple copies. It is further desirable to determine whether two instances of a data item are in fact the same data item, and to perform various other systems' functions and applications on data items without relying on any context information or properties of the data item.

It is also desirable to provide such a mechanism in such a way as to make it transparent to users of the data processing system, and it is desirable that a single mechanism be used to address each of the problems described above.

SUMMARY OF THE INVENTION

This invention provides, in a data processing system, a method and apparatus for identifying a data item in the system, where the identity of the data item depends on all of the data in the data item and only on the data in the data item. Thus the identity of a data item is independent of its name, origin, location, address, or other information not derivable directly from the data, and depends only on the data itself.

This invention further provides an apparatus and a method for determining whether a particular data item is present in the system or at a location in the system, by examining only the data identities of a plurality of data items.

Using the method or apparatus of the present invention, the efficiency and integrity of a data processing system can be improved. The present invention improves the design and operation of a data storage system, file system, relational database, object-oriented database, or the like that stores a plurality of data items, by making possible or improving the design and operation of at least some or all of the following features:

the system stores at most one copy of any data item at a given location, even when multiple data names in the system refer to the same contents;

the system avoids copying data from source to destination locations when the destination locations already have the data;

the system provides transparent access to any data item by reference only to its identity and independent of its present location, whether it be local, remote, or offline;

the system caches data items from a server, so that only the most recently accessed data items need be retained;

when the system is being used to cache data items, problems of maintaining cache consistency are avoided;

the system maintains a desired level of redundancy of data items in a network of servers, to protect against failure by ensuring that multiple copies of the data items are present at different locations in the system;

the system automatically archives data items as they are created or modified;

the system provides the size, age, and location of groups of data items in order to decide whether they can be safely removed from a local file system;

the system can efficiently record and preserve any collection of data items;

the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items;

the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of the data items;

the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM, the inventory is independent of other properties of the data items such as their name, location, and date of creation;

the system allows closely related sets of data items, such as matching or corresponding directories on disconnected computers, to be periodically resynchronized with one another;

the system can verify that data retrieved from another location is the desired or requested data, using only the data identifier used to retrieve the data;

the system can prove possession of specific data items by content without disclosing the content of the data items, for purposes of later legal verification and to provide anonymity;

the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes.

Other objects, features, and characteristics of the present invention as well as the methods of operation and functions of the related elements of structure, and the combination of parts and economies of manufacture, will become more apparent upon consideration of the following description and the appended claims with reference to the accompanying drawings, all of which form a part of this specification.

BRIEF DESCRIPTION OF THE DRAWINGS

FIGS. **1**(*a*) and **1**(*b*) depict a typical data processing system in which a preferred embodiment of the present invention operates;

FIG. **2** depicts a hierarchy of data items stored at any location in such a data processing system;

FIGS. **3-9** depict data structures used to implement an embodiment of the present invention; and

FIGS. **10**(*a*)-**28** are flow charts depicting operation of various aspects of the present invention.

DETAILED DESCRIPTION OF THE PRESENTLY PREFERRED EXEMPLARY EMBODIMENTS

An embodiment of the present invention is now described with reference to a typical data processing system **100**, which, with reference to FIGS. **1**(*a*) and **1**(*b*), includes one or more processors (or computers) **102** and various storage devices **104** connected in some way, for example by a bus **106**.

Each processor **102** includes a CPU **108**, a memory **110** and one or more local storage devices **112**. The CPU **108**, memory **110**, and local storage device **112** may be internally connected, for example by a bus **114**. Each processor **102** may also include other devices (not shown), such as a keyboard, a display, a printer, and the like.

In a data processing system **100**, wherein more than one processor **102** is used, that is, in a multiprocessor system, the processors may be in one of various relationships. For example, two processors **102** may be in a client/server client/client, or a server/server relationship. These inter-processor relationships may be dynamic, changing depending on particular situations and functions. Thus, a particular processor

102 may change its relationship to other processors as needed, essentially setting up a peer-to-peer relationship with other processors. In a peer-to-peer relationship, sometimes a particular processor **102** acts as a client processor, whereas at other times the same processor acts as a server processor. In other words, there is no hierarchy imposed on or required of processors **102**.

In a multiprocessor system, the processors **102** may be homogeneous or heterogeneous. Further, in a multiprocessor data processing system **100**, some or all of the processors **102** may be disconnected from the network of processors for periods of time. Such disconnection may be part of the normal operation of the system **100** or it may be because a particular processor **102** is in need of repair.

Within a data processing system **100**, the data may be organized to form a hierarchy of data storage elements, wherein lower level data storage elements are combined to form higher level elements. This hierarchy can consist of, for example, processors, file systems, regions, directories, data files, segments, and the like. For example, with reference to FIG. **2**, the data items on a particular processor **102** may be organized or structured as a file system **116** which comprises regions **117**, each of which comprises directories **118**, each of which can contain other directories **118** or files **120**. Each file **120** being made up of one or more data segments **122**.

In a typical data processing system, some or all of these elements can be named by users given certain implementation specific naming conventions, the name (or pathname) of an element being relative to a context. In the context of a data processing system **100**, a pathname is fully specified by a processor name, a filesystem name, a sequence of zero or more directory names identifying nested directories, and a final file name. (Usually the lowest level elements, in this case segments **122**, cannot be named by users.)

In other words, a file system **116** is a collection of directories **118**. A directory **118** is a collection of named files **120**—both data files **120** and other directory files **118**. A file **120** is a named data item which is either a data file (which may be simple or compound) or a directory file **118**. A simple file **120** consists of a single data segment **122**. A compound file **120** consists of a sequence of data segments **122**. A data segment **122** is a fixed sequence of bytes. An important property of any data segment is its size, the number of bytes in the sequence.

A single processor **102** may access one or more file systems **116**, and a single storage device **104** may contain one or more file systems **116**, or portions of a file system **116**. For instance, a file system **116** may span several storage devices **104**.

In order to implement controls in a file system, file system **116** may be divided into distinct regions, where each region is a unit of management and control. A region consists of a given directory **118** and is identified by the pathname (user defined) of the directory.

In the following, the term "location", with respect to a data processing system **100**, refers to any of a particular processor **102** in the system, a memory of a particular processor, a storage device, a removable storage medium (such as a floppy disk or compact disk), or any other physical location in the system. The term "local" with respect to a particular processor **102** refers to the memory and storage devices of that particular processor.

In the following, the terms "True Name", "data identity" and "data identifier" refer to the substantially unique data identifier for a particular data item. The term "True File" refers to the actual file segment, or data item identified by a True Name.

A file system for a data processing system **100** is now described which is intended to work with an existing operating system by augmenting some of the operating system's file management system codes. The embodiment provided relies on the standard file management primitives for actually storing to and retrieving data items from disk, but uses the mechanisms of the present invention to reference and access those data items.

The processes and mechanisms (services) provided in this embodiment are grouped into the following categories: primitive mechanisms, operating system mechanisms, remote mechanisms, background mechanisms, and extended mechanisms.

Primitive mechanisms provide fundamental capabilities used to support other mechanisms. The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

Operating system mechanisms provide typical familiar file system mechanisms, while maintaining the data structures required to offer the mechanisms of the, present invention. Operating system mechanisms are designed to augment existing operating systems, and in this way to make the present invention compatible with, and generally transparent to, existing applications. The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

Remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation. The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

Background mechanisms are intended to run occasionally and at a low priority. These provide automated management capabilities with respect to the present invention. The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;

3. Check for Expired Links; and
4. Verify Region; and
5. Groom Source List.

Extended mechanisms run within application programs over the operating system. These mechanisms provide solutions to specific problems and applications. The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at location;
7. Verify True File;
8. Track for accounting purposes; and
9. Track for licensing purposes.

The file system herein described maintains sufficient information to provide a variety of mechanisms not ordinarily offered by an operating system, some of which are listed and described here. Various processing performed by this embodiment of the present invention will now be described in greater detail.

In some embodiments, some files **120** in a data processing system **100** do not have True Names because they have been recently received or created or modified, and thus their True Names have not yet been computed. A file that does not yet have a True Name is called a scratch file. The process of assigning a True Name to a file is referred to as assimilation, and is described later. Note that a scratch file may have a user provided name.

Some of the processing performed by the present invention can take place in a background mode or on a delayed or as-needed basis. This background processing is used to determine information that is not immediately required by the system or which may never be required. As an example, in some cases a scratch file is being changed at a rate greater than the rate at which it is useful to determine its True Name. In these cases, determining the True Name of the file can be postponed or performed in the background.

Data Structures

The following data structures, stored in memory **110** of one of more processors **102** are used to implement the mechanisms described herein. The data structures can be local to each processor **102** of the system **100**, or they can reside on only some of the processors **102**.

The data structures described are assumed to reside on individual peer processors **102** in the data processing system **100**. However, they can also be shared by placing them on a remote, shared file server (for instance, in a local area network of machines). In order to accommodate sharing data structures, it is necessary that the processors accessing the shared database use the appropriate locking techniques to ensure that changes to the shared database do not interfere with one another but are appropriately serialized. These locking techniques are well understood by ordinarily skilled programmers of distributed applications.

It is sometimes desirable to allow some regions to be local to a particular processor **102** and other regions to be shared among processors **102**. (Recall that a region is a unit of file system management and control consisting of a given directory identified by the pathname of the directory.) In the case of local and shared regions, there would be both local and shared versions of each data structure. Simple changes to the processes described below must be made to ensure that appropriate data structures are selected for a given operation.

The local directory extensions (LDE) table **124** is a data structure which provides information about files **120** and directories **118** in the data processing system **100**. The local directory extensions table **124** is indexed by a pathname or contextual name (that is, a user provided name) of a file and includes the True Name for most files. The information in local directory extension table **124** is in addition to that provided by the native file system of the operating system.

The True File registry (TFR) **126** is a data store for listing actual data items which have True Names, both files **120** and segments **122**. When such data items occur in the True File registry **126** they are known as True Files. True Files are identified in True File registry **126** by their True Names or identities. The table True File registry **126** also stores location, dependency, and migration information about True Files.

The region table (RT) **128** defines areas in the network storage which are to be managed separately. Region table **128** defines the rules for access to and migration of files **120** among various regions with the local file system **116** and remote peer file systems.

The source table (ST) **130** is a list of the sources of True Files other than the current True File registry **126**. The source table **130** includes removable volumes and remote processors.

The audit file (AF) **132** is a list of records indicating changes to be made in local or remote files, these changes to be processed in background.

The accounting log (AL) **134** is a log of file transactions used to create accounting information in a manner which preserves the identity of files being tracked independent of their name or location.

The license table (LT) **136** is a table identifying files, which may only be used by licensed users, in a manner independent of their name or location, and the users licensed to use them.

Detailed Descriptions of the Data Structures

The following table summarizes the fields of an local directory extensions table entry, as illustrated by record **138** in FIG. **3**.

Field	Description
Region ID	identifies the region in which this file is contained.
Pathname	the user provided name or contextual name of the file or directory, relative to the region in which it occurs.
True Name	the computed True Name or identity of the file or directory. This True Name is not always up to date, and it is set to a special value when a file is modified and is later recomputed in the background.
Type	indicates whether the file is a data file or a directory.
Scratch File ID	the physical location of the file in the file system, when no True Name has been calculated for the file. As noted above, such a file is called a scratch file.
Time of last access	the last access time to this file. If this file is a directory, this is the last access time to any file in the directory.
Time of last modification	the time of last change of this file. If this file is a directory, this is the last modification time of any file in the directory.
Safe flag	indicates that this file (and, if this file is a directory, all of its subordinate files) have been backed up on some other system, and it is therefore safe to remove them.
Lock flag	indicates whether a file is locked, that is, it is being modified by the local processor or a remote processor. Only one processor may modify a file at a time.
Size	the full size of this directory (including all subordinate files), if all files in it were fully expanded and duplicated. For a file that is not a directory this is the size of the actual True File.
Owner	the identity of the user who owns this file, for accounting and license tracking purposes.

Each record of the True File registry **126** has the fields shown in the True File registry record **140** in FIG. **4**. The True File registry **126** consists of the database described in the table below as well as the actual True Files identified by the True File IDs below.

Field	Description
True Name	computed True Name or identity of the file.
Compressed File ID	compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.
Grooming delete count	tentative count of how many references have been selected for deletion during a grooming operation.
Time of last access	most recent date and time the content of this file was accessed.
Expiration	date and time after which this file may be deleted by this server.
Dependent processors	processor IDs of other processors which contain references to this True File.
Source IDs	source ID(s) of zero or more sources from which this file or data item may be retrieved.
True File ID	identity or disk location of the actual physical representation of the file or file segment. It is sufficient to use a filename in the registration directory of the underlying operating system. The True File ID is absent if the actual file is not currently present at the current location.
Use count	number of other records on this processor which identify this True File.

A region table **128**, specified by a directory pathname, records storage policies which allow files in the file system to be stored, accessed and migrated in different ways. Storage policies are programmed in a configurable way using a set of rules described below.

Each region table record **142** of region table **128** includes the fields described in the following table (with reference to FIG. **5**):

Field	Description
Region ID	internally used identifier for this region.
Region file system	file system on the local processor of which this region is a part.
Region pathname	a pathname relative to the region file system which defines the location of this region. The region consists of all files and directories subordinate to this pathname, except those in a region subordinate to this region.
Mirror processor(s)	zero or more identifiers of processors which are to keep mirror or archival copies of all files in the current region. Multiple mirror processors can be defined to form a mirror group.
Mirror duplication count	number of copies of each file in this region that should be retained in a mirror group.
Region status	specifies whether this region is local to a single processor 102, shared by several processors 102 (if, for instance, it resides on a shared file server), or managed by a remote processor.
Policy	the migration policy to apply to this region. A single region might participate in several policies. The policies are as follows (parameters in brackets are specified as part of the policy): region is a cached version from [processor ID]; region is a member of a mirror set defined by [processor ID]. region is to be archived on [processor ID]. region is to be backed up locally, by placing new copies in [region ID]. region is read only and may not be changed. region is published and expires on [date]. Files in this region should be compressed.

A source table **130** identifies a source location for True Files. The source table **130** is also used to identify client processors making reservations on the current processor. Each source record **144** of the source table **130** includes the fields summarized in the following table, with reference to FIG. **6**:

Field	Description
source ID	internal identifier used to identify a particular source.
source type	type of source location: Removable Storage Volume Local Region Cache Server Mirror Group Server Cooperative Server Publishing Server Client
source rights	includes information about the rights of this processor, such as whether it can ask the local processor to store data items for it.
source availability	measurement of the bandwidth, cost, and reliability of the connection to this source of True Files. The availability is used to select from among several possible sources.
source location	information on how the local processor is to access the source. This may be, for example, the name of a removable storage volume, or the processor ID and region path of a region on a remote processor.

The audit file **132** is a table of events ordered by timestamp, each record **146** in audit file **132** including the fields summarized in the following table (with reference to FIG. **7**):

Field	Description
Original Name	path of the file in question.
Operation	whether the file was created, read, written, copied or deleted.
Type	specifies whether the source is a file or a directory.
Processor ID	ID of the remote processor generating this event (if not local).
Timestamp	time and date file was closed (required only for accessed/modified files).
Pathname	Name of the file (required only for rename).
True Name	computed True Name of the file. This is used by remote systems to mirror changes to the directory and is filled in during background processing.

Each record **148** of the accounting log **134** records an event which may later be used to provide information for billing mechanisms. Each accounting log entry record **148** includes at least the information summarized in the following table, with reference to FIG. **8**:

Field	Description
date of entry	date and time of this log entry.
type of entry	Entry types include create file, delete file, and transmit file.
True Name	True Name of data item in question.
owner	identity of the user responsible for this action.

Each record **150** of the license table **136** records a relationship between a licensable data item and the user licensed to

have access to it. Each license table record **150** includes the information summarized in the following table, with reference to FIG. **9**:

Field	Description
True Name	True Name of a data item subject to license validation.
licensee	identity of a user authorized to have access to this object.

Various other data structures are employed on some or all of the processors **102** in the data processing system **100**. Each processor **102** has a global freeze lock (GFL) **152** (FIG. **1**), which is used to prevent synchronization errors when a directory is frozen or copied. Any processor **102** may include a special archive directory (SAD) **154** into which directories may be copied for the purposes of archival. Any processor **102** may include a special media directory (SMD) **156**, into which the directories of removable volumes are stored to form a media inventory. Each processor has a grooming lock **158**, which is set during a grooming operation. During this period the grooming delete count of True File registry entries **140** is active, and no True Files should be deleted until grooming is complete. While grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

Primitive Mechanisms

The first of the mechanisms provided by the present invention, primitive mechanisms, are now described. The mechanisms described here depend on underlying data management mechanisms to create, copy, read, and delete data items in the True File registry **126**, as identified by a True File ID. This support may be provided by an underlying operating system or disk storage manager.

The following primitive mechanisms are described:

1. Calculate True Name;
2. Assimilate Data Item;
3. New True File;
4. Get True Name from Path;
5. Link Path to True Name;
6. Realize True File from Location;
7. Locate Remote File;
8. Make True File Local;
9. Create Scratch File;
10. Freeze Directory;
11. Expand Frozen Directory;
12. Delete True File;
13. Process Audit File Entry;
14. Begin Grooming;
15. Select For Removal; and
16. End Grooming.

1. Calculate True Name

A True Name is computed using a function, MD, which reduces a data block B of arbitrary length to a relatively small, fixed size identifier, the True Name of the data block, such that the True Name of the data block is virtually guaranteed to represent the data block B and only data block B.

The function MD must have the following properties:

1. The domain of the function MD is the set of all data items. The range of the function MD is the set of True Names.
2. The function MD must take a data item of arbitrary length and reduce it to an integer value in the range 0 to N−1, where N is the cardinality of the set of True Names. That is, for an arbitrary length data block B, $0 \leqq MD(B) \leqq N$.
3. The results of MD(B) must be evenly and randomly distributed over the range of N, in such a way that simple or regular changes to B are virtually guaranteed to produce a different value of MD(B).
4. It must be computationally difficult to find a different value B' such that MD(B)=MD(B').
5. The function MD(B) must be efficiently computed.

A family of functions with the above properties are the so-called message digest functions, which are used in digital security systems as techniques for authentification of data. These functions (or algorithms) include MD4, MD5, and SHA.

In the presently preferred embodiments, either MD5 or SHA is employed as the basis for the computation of True Names. Whichever of these two message digest functions is employed, that same function must be employed on a system-wide basis.

It is impossible to define a function having a unique output for each possible input when the number of elements in the range of the function is smaller than the number of elements in its domain. However, a crucial observation is that the actual data items that will be encountered in the operation of any system embodying this invention form a very sparse subset of all the possible inputs.

A colliding set of data items is defined as a set wherein, for one or more pairs x and y in the set, MD(x)=MD(y). Since a function conforming to the requirements for MD must evenly and randomly distribute its outputs, it is possible, by making the range of the function large enough, to make the probability arbitrarily small that actual inputs encountered in the operation of an embodiment of this invention will form a colliding set.

To roughly quantify the probability of a collision, assume that there are no more than 2^{30} storage devices in the world, and that each storage device has an average of at most 2^{20} different data items. Then there are at most 2^{50} data items in the world. If the outputs of MD range between 0 and 2^{128}, it can be demonstrated that the probability of a collision is approximately 1 in 2^{29}. Details on the derivation of these probability values are found, for example, in P. Flajolet and A. M. Odlyzko, "Random Mapping Statistics," *Lecture Notes in Computer Science* 434: *Advances in Cryptology—Eurocrypt '89 Proceedings*, Springer-Verlag, pp. 329-354.

Note that for some less preferred embodiments of the present invention, lower probabilities of uniqueness may be acceptable, depending on the types of applications and mechanisms used. In some embodiments it may also be useful to have more than one level of True Names, with some of the True Names having different degrees of uniqueness. If such a scheme is implemented, it is necessary to ensure that less unique True Names are not propagated in the system. While the invention is described herein using only the True Name of a data item as the identifier for the data item, other preferred embodiments use tagged, typed, categorized or classified data items and use a combination of both the True Name and the tag, type, category or class of the data item as an identifier. Examples of such categorizations are files, directories, and segments; executable files and data files, and the like. Examples of classes are classes of objects in an object-oriented system. In such a system, a lower degree of True Name uniqueness is acceptable over the entire universe of data items, as long as sufficient uniqueness is provided per category of data items. This is because the tags provide an additional level of uniqueness.

A mechanism for calculating a True Name given a data item is now described, with reference to FIGS. **10**(*a*) and **10**(*b*).

A simple data item is a data item whose size is less than a particular given size (which must be defined in each particular implementation of the invention). To determine the True Name of a simple data item, with reference to FIG. **10**(*a*), first compute the MD function (described above) on the given simple data item (Step S**212**). Then append to the resulting 128 bits, the byte length modulo **32** of the data item (Step S**214**). The resulting 160-bit value is the True Name of the simple data item.

A compound data item is one whose size is greater than the particular given size of a simple data item. To determine the True Name of an arbitrary (simple or compound) data item, with reference to FIG. **10**(*b*), first determine if the data item is a simple or a compound data item (Step S**216**). If the data item is a simple data item, then compute its True Name in step S**218** (using steps S**212** and S**214** described above), otherwise partition the data item into segments (Step S**220**) and assimilate each segment (Step S**222**) (the primitive mechanism, Assimilate a Data Item, is described below), computing the True Name of the segment. Then create an indirect block consisting of the computed segment True Names (Step S**224**). An indirect block is a data item which consists of the sequence of True Names of the segments. Then, in step S**226**, assimilate the indirect block and compute its True Name. Finally, replace the final thirty-two (32) bits of the resulting True Name (that is, the length of the indirect block) by the length modulo **32** of the compound data item (Step S**228**). The result is the True Name of the compound data item.

Note that the compound data item may be so large that the indirect block of segment True Names is itself a compound data item. In this case the mechanism is invoked recursively until only simple data items are being processed.

Both the use of segments and the attachment of a length to the True Name are not strictly required in a system using the present invention, but are currently considered desirable features in the preferred embodiment.

2. Assimilate Data Item

A mechanism for assimilating a data item (scratch file or segment) into a file system, given the scratch file ID of the data item, is now described with reference to FIG. **11**. The purpose of this mechanism is to add a given data item to the True File registry **126**. If the data item already exists in the True File registry **126**, this will be discovered and used during this process, and the duplicate will be eliminated.

Thereby the system stores at most one copy of any data item or file by content, even when multiple names refer to the same content.

First, determine the True Name of the data item corresponding to the given scratch File ID using the Calculate True Name primitive mechanism (Step S**230**). Next, look for an entry for the True Name in the True File registry **126** (Step S**232**) and determine whether a True Name entry, record **140**, exists in, the True File registry **126**. If the entry record includes a corresponding True File ID or compressed File ID (Step S**237**), delete the file with the scratch File ID (Step S**238**). Otherwise store the given True File ID in the entry record (step S**239**).

If it is determined (in step S**232**) that no True Name entry exists in the True File registry **126**, then, in Step S**236**, create a new entry in the True File registry **126** for this True Name. Set the True Name of the entry to the calculated True Name, set the use count for the new entry to one, store the given True File ID in the entry and set the other fields of the entry as appropriate.

Because this procedure may take some time to compute, it is intended to run in background after a file has ceased to change. In the meantime, the file is considered an unassimilated scratch file.

3. New True File

The New True File process is invoked when processing the audit file **132**, some time after a True File has been assimilated (using the Assimilate Data Item primitive mechanism). Given a local directory extensions table entry record **138** in the local directory extensions table **124**, the New True File process can provide the following steps (with reference to FIG. **12**), depending on how the local processor is configured:

First, in step S**238**, examine the local directory extensions table entry record **138** to determine whether the file is locked by a cache server. If the file is locked, then add the ID of the cache server to the dependent processor list of the True File registry table **126**, and then send a message to the cache server to update the cache of the current processor using the Update Cache remote mechanism (Step **242**).

If desired, compress the True File (Step S**246**), and, if desired, mirror the True File using the Mirror True File background mechanism (Step S**248**).

4. Get True Name from Path

The True Name of a file can be used to identify a file by contents, to confirm that a file matches its original contents, or to compare two files. The mechanism to get a True Name given the pathname of a file is now described with reference to FIG. **13**.

First, search the local directory extensions table **124** for the entry record **138** with the given pathname (Step S**250**). If the pathname is not found, this process fails and no True Name corresponding to the given pathname exists. Next, determine whether the local directory extensions table entry record **138** includes a True Name (Step S**252**), and if so, the mechanism's task is complete. Otherwise, determine whether the local directory extensions table entry record **138** identifies a directory (Step S**254**), and if so, freeze the directory (Step S**256**) (the primitive mechanism Freeze Directory is described below).

Otherwise, in step S**258**, assimilate the file (using the Assimilate Data Item primitive mechanism) defined by the File ID field to generate its True Name and store its True Name in the local directory extensions entry record. Then return the True Name identified by the local directory extensions table **124**.

5. Link Path to True Name

The mechanism to link a path to a True Name provides a way of creating a new directory entry record identifying an existing, assimilated file. This basic process may be used to copy, move, and rename files without a need to copy their contents. The mechanism to link a path to a True Name is now described with reference to FIG. **14**.

First, if desired, confirm that the True Name exists locally by searching for it in the True Name registry or local directory extensions table **135** (Step S**260**). Most uses of this mechanism will require this form of validation. Next, search for the path in the local directory extensions table **135** (Step S**262**). Confirm that the directory containing the file named in the path already exists (Step S**264**). If the named file itself exists, delete the File using the Delete True File operating system mechanism (see below) (Step S**268**).

Then, create an entry record in the local directory extensions with the specified path (Step S**270**) and update the entry record and other data structures as follows: fill in the True Name field of the entry with the specified True Name; increment the use count for the True File registry entry record **140** of the corresponding True Name; note whether the entry is a

directory by reading the True File to see if it contains a tag (magic number) indicating that it represents a frozen directory (see also the description of the Freeze Directory primitive mechanism regarding the tag); and compute and set the other fields of the local directory extensions appropriately. For instance, search the region table **128** to identify the region of the path, and set the time of last access and time of last modification to the current time.

6. Realize True File from Location

This mechanism is used to try to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. This mechanism is now described with reference to FIG. **15**.

First, in step S**272**, determine whether the location specified is a processor. If it is determined that the location specified is a processor, then send a Request True File message (using the Request True File remote mechanism) to the remote processor and wait for a response (Step S**274**). If a negative response is received or no response is received after a timeout period, this mechanism fails. If a positive response is received, enter the True File returned in the True File registry **126** (Step S**276**). (If the file received was compressed, enter the True File ID in the compressed File ID field.)

If, on the other hand, it is determined in step S**272** that the location specified is not a processor, then, if necessary, request the user or operator to mount the indicated volume (Step S**278**). Then (Step S**280**) find the indicated file on the given volume and assimilate the file using the Assimilate Data Item primitive mechanism. If the volume does not contain a True File registry **126**, search the media inventory to find the path of the file on the volume. If no such file can be found, this mechanism fails.

At this point, whether or not the location is determined (in step S**272**) to be a processor, if desired, verify the True File (in step S**282**).

7. Locate Remote File

This mechanism allows a processor to locate a file or data item from a remote source of True Files, when a specific source is unknown or unavailable. A client processor system may ask one of several or many sources whether it can supply a data object with a given True Name. The steps to perform this mechanism are as follows (with reference to FIGS. **16**(*a*) and **16**(*b*)).

The client processor **102** uses the source table **145** to select one or more source processors (Step S**284**). If no source processor can be found, the mechanism fails. Next, the client processor **102** broadcasts to the selected sources a request to locate the file with the given True Name using the Locate True File remote mechanism (Step S**286**). The request to locate may be augmented by asking to propagate this request to distant servers. The client processor then waits for one or more servers to respond positively (Step S**288**). After all servers respond negatively, or after a timeout period with no positive response, the mechanism repeats selection (Step S**284**) to attempt to identify alternative sources. If any selected source processor responds, its processor ID is the result of this mechanism. Store the processor ID in the source field of the True File registry entry record **140** of the given True Name (Step S**290**).

If the source location of the True Name is a different processor or medium than the destination (Step S**290***a*), perform the following steps:

(i) Look up the True File registry entry record **140** for the corresponding True Name, and add the source location ID to the list of sources for the True Name (Step S**290***b*); and

(ii) If the source is a publishing system, determine the expiration date on the publishing system for the True Name and add that to the list of sources. If the source is not a publishing system, send a message to reserve the True File on the source processor (Step S**290***c*).

Source selection in step S**284** may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost, and ignoring previously selected processors which did not respond in step S**288**.

8. Make True File Local

This mechanism is used when a True Name is known and a locally accessible copy of the corresponding file or data item is required. This mechanism makes it possible to actually read the data in a True File. The mechanism takes a True Name and returns when there is a local, accessible copy of the True File in the True File registry **126**. This mechanism is described here with reference to the flow chart of FIGS. **17**(*a*) and **17**(*b*).

First, look in the True File registry **126** for a True File entry record **140** for the corresponding True Name (Step S**292**). If no such entry is found this mechanism fails. If there is already a True File ID for the entry (Step S**294**), this mechanism's task is complete. If there is a compressed file ID for the entry (Step S**296**), decompress the file corresponding to the file ID (Step S**298**) and store the decompressed file ID in the entry (Step S**300**). This mechanism is then complete.

If there is no True File ID for the entry (Step S**294**) and there is no compressed file ID for the entry (Step S**296**), then continue searching for the requested file. At this time it may be necessary to notify the user that the system is searching for the requested file.

If there are one or more source IDs, then select an order in which to attempt to realize the source ID (Step S**304**). The order may be based on optimizations involving general availability of the source, access time, bandwidth, and transmission cost. For each source in the order chosen, realize the True File from the source location (using the Realize True File from Location primitive mechanism), until the True File is realized (Step S**306**). If it is realized, continue with step S**294**. If no known source can realize the True File, use the Locate Remote File primitive mechanism to attempt to find the True File (Step S**308**). If this succeeds, realize the True File from the identified source location and continue with step S**296**.

9. Create Scratch File

A scratch copy of a file is required when a file is being created or is about to be modified. The scratch copy is stored in the file system of the underlying operating system. The scratch copy is eventually assimilated when the audit file record entry **146** is processed by the Process Audit File Entry primitive mechanism. This Create Scratch File mechanism requires a local directory extensions table entry record **138**. When it succeeds, the local directory extensions table entry record **138** contains the scratch file ID of a scratch file that is not contained in the True File registry **126** and that may be modified. This mechanism is now described with reference to FIGS. **18**(*a*) and **18**(*b*).

First determine whether the scratch file should be a copy of the existing True File (Step S**310**). If so, continue with step S**312**. Otherwise, determine whether the local directory extensions table entry record **138** identifies an existing True File (Step S**316**), and if so, delete the True File using the Delete True File primitive mechanism (Step S**318**). Then create a new, empty scratch file and store its scratch file ID in the local directory extensions table entry record **138** (step S**320**). This mechanism is then complete.

If the local directory extensions table entry record **138** identifies a scratch file ID (Step S**312**), then the entry already has a scratch file, so this mechanism succeeds.

If the local directory extensions table entry record **138** identifies a True File (S**316**), and there is no True File ID for the True File (S**312**), then make the True File local using the Make True File Local primitive mechanism (Step S**322**). If there is still no True File ID, this mechanism fails.

There is now a local True File for this file. If the use count in the corresponding True File registry entry record **140** is one (Step S**326**), save the True File ID in the scratch file ID of the local directory extensions table entry record **138**, and remove the True File registry entry record **140** (Step S**328**). (This step makes the True File into a scratch file.) This mechanism's task is complete.

Otherwise, if the use count in the corresponding True File registry entry record **140** is not one (in step S**326**), copy the file with the given True File ID to a new scratch file, using the Read File OS mechanism and store its file ID in the local directory extensions table entry record **138** (Step S**330**), and reduce the use count for the True File by one. If there is insufficient space to make a copy, this mechanism fails.

10. Freeze Directory

This mechanism freezes a directory in order to calculate its True Name. Since the True Name of a directory is a function of the files within the directory, they must not change during the computation of the True Name of the directory. This mechanism requires the pathname of a directory to freeze. This mechanism is described with reference to FIGS. **19**(*a*) and **19**(*b*).

In step S**332**, add one to the global freeze lock. Then search the local directory extensions table **124** to find each subordinate data file and directory of the given directory, and freeze each subordinate directory found using the Freeze Directory primitive mechanism (Step S**334**). Assimilate each unassimilated data file in the directory using the Assimilate Data Item primitive mechanism (Step S**336**). Then create a data item which begins with a tag or marker (a "magic number") being a unique data item indicating that this data item is a frozen directory (Step S**337**). Then list the file name and True Name for each file in the current directory (Step S**338**). Record any additional information required, such as the type, time of last access and modification, and size (Step S**340**). Next, in step S**342**, using the Assimilate Data Item primitive mechanism, assimilate the data item created in step S**338**. The resulting True Name is the True Name of the frozen directory. Finally, subtract one from the global freeze lock (Step S**344**).

11. Expand Frozen Directory

This mechanism expands a frozen directory in a given location. It requires a given pathname into which to expand the directory, and the True Name of the directory and is described with reference to FIG. **20**.

First, in step S**346**, make the True File with the given True Name local using the Make True File Local primitive mechanism. Then read each directory entry in the local file created in step S**346** (Step S**348**). For each such directory entry, do the following:

Create a full pathname using the given pathname and the file name of the entry (Step S**350**); and

link the created path to the True Name (Step S**352**) using the Link Path to True Name primitive mechanism.

12. Delete True File

This mechanism deletes a reference to a True Name. The underlying True File is not removed from the True File registry **126** unless there are no additional references to the file. With reference to FIG. **21**, this mechanism is performed as follows:

If the global freeze lock is on, wait until the global freeze lock is turned off (Step S**354**). This prevents deleting a True File while a directory which might refer to it is being frozen. Next, find the True File registry entry record **140** given the True Name (Step S**356**). If the reference count field of the True File registry **126** is greater than zero, subtract one from the reference count field (Step S**358**). If it is determined (in step S**360**) that the reference count field of the True File registry entry record **140** is zero, and if there are no dependent systems listed in the True File registry entry record **140**, then perform the following steps:

(i) If the True File is a simple data item, then delete the True File, otherwise,

(ii) (the True File is a compound data item) for each True Name in the data item, recursively delete the True File corresponding to the True Name (Step S**362**).

(iii) Remove the file indicated by the True File ID and compressed file ID from the True File registry **126**, and remove the True File registry entry record **140** (Step S**364**).

13. Process Audit File Entry

This mechanism performs tasks which are required to maintain information in the local directory extensions table **124** and True File registry **126**, but which can be delayed while the processor is busy doing more time-critical tasks. Entries **142** in the audit file **132** should be processed at a background priority as long as there are entries to be processed. With reference to FIG. **22**, the steps for processing an entry are as follows:

Determine the operation in the entry **142** currently being processed (Step S**365**). If the operation indicates that a file was created or written (Step S**366**), then assimilate the file using the Assimilate Data Item primitive mechanism (Step S**368**), use the New True File primitive mechanism to do additional desired processing (such as cache update, compression, and mirroring) (Step S**369**), and record the newly computed True Name for the file in the audit file record entry (Step S**370**).

Otherwise, if the entry being processed indicates that a compound data item or directory was copied (or deleted) (Step S**376**), then for each component True Name in the compound data item or directory, add (or subtract) one to the use count of the True File registry entry record **140** corresponding to the component True Name (Step S**378**).

In all cases, for each parent directory of the given file, update the size, time of last access, and time of last modification, according to the operation in the audit record (Step S**379**).

Note that the audit record is not removed after processing, but is retained for some reasonable period so that it may be used by the Synchronize Directory extended mechanism to allow a disconnected remote processor to update its representation of the local system.

14. Begin Grooming

This mechanism makes it possible to select a set of files for removal and determine the overall amount of space to be recovered. With reference to FIG. **23**, first verify that the global grooming lock is currently unlocked (Step S**382**). Then set the global grooming lock, set the total amount of space freed during grooming to zero and empty the list of files selected for deletion (Step S**384**). For each True File in the True File registry **126**, set the delete count to zero (Step S**386**).

15. Select For Removal

This grooming mechanism tentatively selects a pathname to allow its corresponding True File to be removed. With reference to FIG. **24**, first find the local directory extensions table entry record **138** corresponding to the given pathname (Step S**388**). Then find the True File registry entry record **140** corresponding to the True File name in the local directory extensions table entry record **138** (Step S**390**). Add one to the

grooming delete count in the True File registry entry record **140** and add the pathname to a list of files selected for deletion (Step S**392**). If the grooming delete count of the True File registry entry record **140** is equal to the use count of the True File registry entry record **140**, and if the there are no entries in the dependency list of the True File registry entry record **140**, then add the size of the file indicated by the True File ID and or compressed file ID to the total amount of space freed during grooming (Step. S**394**).

16. End Grooming

This grooming mechanism ends the grooming phase and removes all files selected for removal. With reference to FIG. **25**, for each file in the list of files selected for deletion, delete the file (Step S**396**) and then unlock the global grooming lock (Step S**398**).

Operating System Mechanisms

The next of the mechanisms provided by the present invention, operating system mechanisms, are now described.

The following operating system mechanisms are described:

1. Open File;
2. Close File;
3. Read File;
4. Write File;
5. Delete File or Directory;
6. Copy File or Directory;
7. Move File or Directory;
8. Get File Status; and
9. Get Files in Directory.

1. Open File

A mechanism to open a file is described with reference to FIGS. **26**(*a*) and **26**(*b*). This mechanism is given as input a pathname and the type of access required for the file (for example, read, write, read/write, create, etc.) and produces either the File ID of the file to be opened or an indication that no file should be opened. The local directory extensions table record **138** and region table record **142** associated with the opened file are associated with the open file for later use in other processing functions which refer to the file, such as read, write, and close.

First, determine whether or not the named file exists locally by examining the local directory extensions table **124** to determine whether there is an entry corresponding to the given pathname (Step S**400**). If it is determined that the file name does not exist locally, then, using the access type, determine whether or not the file is being created by this opening process (Step S**402**). If the file is not being created, prohibit the open (Step S**404**). If the file is being created, create a zero-length scratch file using an entry in local directory extensions table **124** and produce the scratch file ID of this scratch file as the result (Step S**406**).

If, on the other hand, it is determined in step S**400** that the file name does exist locally, then determine the region in which the file is located by searching the region table **128** to find the record **142** with the longest region path which is a prefix of the file pathname (Step S**408**). This record identifies the region of the specified file.

Next, determine using the access type, whether the file is being opened for writing or whether it is being opened only for reading (Step S**410**). If the file is being opened for reading only, then, if the file is a scratch file (Step S**419**), return the scratch File ID of the file (Step S**424**). Otherwise get the True Name from the local directory extensions table **124** and make a local version of the True File associated with the True Name using the Make. True File Local primitive mechanism, and then return the True File ID associated with the True Name (Step S**420**).

If the file is not being opened for reading only (Step S**410**), then, if it is determined by inspecting the region table entry record **142** that the file is in a read-only directory (Step S**416**), then prohibit the opening (Step S**422**).

If it is determined by inspecting the region table **128** that the file is in a cached region (Step S**423**), then send a Lock Cache message to the corresponding cache server, and wait for a return message (Step S**418**). If the return message says the file is already locked, prohibit the opening.

If the access type indicates that the file being modified is being rewritten completely (Step S**419**), so that the original data will not be required, then Delete the File using the Delete File OS mechanism (Step S**421**) and perform step S**406**. Otherwise, make a scratch copy of the file (Step S**417**) and produce the scratch file ID of the scratch file as the result (Step S**424**).

2. Close File

This mechanism takes as input the local directory extensions table entry record **138** of an open file and the data maintained for the open file. To close a file, add an entry to the audit file indicating the time and operation (create, read or write). The audit file processing (using the Process Audit File Entry primitive mechanism) will take care of assimilating the file and thereby updating the other records.

3. Read File

To read a file, a program must provide the offset and length of the data to be read, and the location of a buffer into which to copy the data read.

The file to be read from is identified by an open file descriptor which includes a File ID as computed by the Open File operating system mechanism defined above. The File ID may identify either a scratch file or a True File (or True File segment). If the File ID identifies a True File, it may be either a simple or a compound True File. Reading a file is accomplished by the following steps:

In the case where the File ID identifies a scratch file or a simple True File, use the read capabilities of the underlying operating system.

In the case where the File ID identifies a compound file, break the read operation into one or more read operations on component segments as follows:

A. Identify the segment(s) to be read by dividing the specified file offset and length each by the fixed size of a segment (a system dependent parameter), to determine the segment number and number of segments that must be read.

B. For each segment number computed above, do the following:

i. Read the compound True File index block to determine the True Name of the segment to be read.

ii. Use the Realize True File from Location primitive mechanism to make the True File segment available locally. (If that mechanism fails, the Read File mechanism fails).

iii. Determine the File ID of the True File specified by the True Name corresponding to this segment.

iv. Use the Read File mechanism (recursively) to read from this segment into the corresponding location in the specified buffer.

4. Write File

File writing uses the file ID and data management capabilities of the underlying operating system. File access (Make File Local described-above) can be deferred until the first read or write.

5. Delete File or Directory

The process of deleting a file, for a given pathname, is described here with reference to FIGS. **27**(*a*) and **27**(*b*).

First, determine the local directory extensions table entry record **138** and region table entry record **142** for the file (Step S**422**). If the file has no local directory extensions table entry record **138** or is locked or is in a read-only region, prohibit the deletion.

Identify the corresponding True File given the True Name of the file being deleted using the True File registry **126** (Step S**424**). If the file has no True Name, (Step S**426**) then delete the scratch copy of the file based on its scratch file ID in the local directory extensions table **124** (Step S**427**), and continue with step S**428**.

If the file has a True Name and the True File's use count is one (Step S**429**), then delete the True File (Step S**430**), and continue with step S**428**.

If the file has a True Name and the True File's use count is greater than one, reduce its use count by one (Step S**431**). Then proceed with step S**428**.

In Step S**428**, delete the local directory extensions table entry record, and add an entry to the audit file **132** indicating the time and the operation performed (delete).

6. Copy File or Directory

A mechanism is provided to copy a file or directory given a source and destination processor and pathname. The Copy File mechanism does not actually copy the data in the file, only the True Name of the file. This mechanism is performed as follows:

(A) Given the source path, get the True Name from the path. If this step fails, the mechanism fails.

(B) Given the True Name and the destination path, link the destination path to the True Name.

(C) If the source and destination processors have different True File registries, find (or, if necessary, create) an entry for the True Name in the True File registry table **126** of the destination processor. Enter into the source ID field of this new entry the source processor identity.

(D) Add an entry to the audit file **132** indicating the time and operation performed (copy).

This mechanism addresses capability of the system to avoid copying data from a source location to a destination location when the destination already has the data. In addition, because of the ability to freeze a directory, this mechanism also addresses capability of the system immediately to make a copy of any collection of files, thereby to support an efficient version control mechanisms for groups of files.

7. Move File or Directory

A mechanism is described which moves (or renames) a file from a source path to a destination path. The move operation, like the copy operation, requires no actual transfer of data, and is performed as follows:

(A) Copy the file from the source path to the destination path.

(B) If the source path is different from the destination path, delete the source path.

8. Get File Status

This mechanism takes a file pathname and provides information about the pathname. First the local directory extensions table entry record **138** corresponding to the pathname given is found. If no such entry exists, then this mechanism fails, otherwise, gather information about the file and its corresponding True File from the local directory extensions table **124**. The information can include any information shown in the data structures, including the size, type, owner, True Name, sources, time of last access, time of last modification, state (local or not, assimilated or not, compressed or not), use count, expiration date, and reservations.

9. Get Files in Directory

This mechanism enumerates the files in a directory. It is used (implicitly) whenever it is necessary to determine whether a file exists (is present) in a directory. For instance, it is implicitly used in the Open File, Delete File, Copy File or Directory, and Move File operating system mechanisms, because the files operated on are referred to by pathnames containing directory names. The mechanism works as follows:

The local directory extensions table **124** is searched for an entry **138** with the given directory pathname. If no such entry is found, or if the entry found is not a directory, then this mechanism fails.

If there is a corresponding True File field in the local directory extensions table record, then it is assumed that the True File represents a frozen directory. The Expand Frozen Directory primitive mechanism is used to expand the existing True File into directory entries in the local directory extensions table.

Finally, the local directory extensions table **124** is again searched, this time to find each directory subordinate to the given directory. The names found are provided as the result.

Remote Mechanisms

The remote mechanisms provided by the present invention are now described. Recall that remote mechanisms are used by the operating system in responding to requests from other processors. These mechanisms enable the capabilities of the present invention in a peer-to-peer network mode of operation.

In a presently preferred embodiment, processors communicate with each other using a remote procedure call (RPC) style interface, running over one of any number of communication protocols such as IPX/SPX or TCP/IP. Each peer processor which provides access to its True File registry **126** or file regions, or which depends on another peer processor, provides a number of mechanisms which can be used by its peers.

The following remote mechanisms are described:

1. Locate True File;
2. Reserve True File;
3. Request True File;
4. Retire True File;
5. Cancel Reservation;
6. Acquire True File;
7. Lock Cache;
8. Update Cache; and
9. Check Expiration Date.

1. Locate True File

This mechanism allows a remote processor to determine whether the local processor contains a copy of a specific True File. The mechanism begins with a True Name and a flag indicating whether to forward requests for this file to other servers. This mechanism is now described with reference to FIG. **28**.

First determine if the True File is available locally or if there is some indication of where the True File is located (for example, in the Source IDs field). Look up the requested True Name in the True File registry **126** (Step S**432**).

If a True File registry entry record **140** is not found for this True Name (Step S**434**), and the flag indicates that the request is not to be forwarded (Step S**436**), respond negatively (Step S**438**). That is, respond to the effect that the True File is not available.

One the other hand, if a True File registry entry record **140** is not found (Step S**434**), and the flag indicates that the request for this True File is to be forwarded (Step S**436**), then forward a request for this True File to some other processors in the system (Step S**442**). If the source table for the current

processor identifies one or more publishing servers which should have a copy of this True File then forward the request to each of those publishing servers (Step S**436**).

If a True File registry entry record **140** is found for the required True File (Step S**434**), and if the entry includes a True File ID or Compressed File ID (Step S**440**), respond positively (Step S**444**). If the entry includes a True File ID then this provides the identity or disk location of the actual physical representation of the file or file segment required. If the entry include a Compressed File ID, then a compressed version of the True File may be stored instead of, or in addition to, an uncompressed version. This field provides the identity of the actual representation of the compressed version of the file.

If the True File registry entry record **140** is found (Step S**434**) but does not include a True File ID (the File ID is absent if the actual file is not currently present at the current location) (Step S**440**), and if the True File registry entry record **140** includes one or more source processors, and if the request can be forwarded, then forward the request for this True File to one or more of the source processors (Step S**444**).

2. Reserve True File

This mechanism allows a remote processor to indicate that it depends on the local processor for access to a specific True File. It takes a True Name as input. This mechanism is described here.

(A) Find the True File registry entry record **140** associated with the given True File. If no entry exists, reply negatively.

(B) If the True File registry entry record **140** does not include a True File ID or compressed File ID, and if the True File registry entry record **140** includes no source IDs for removable storage volumes, then this processor does not have access to a copy of the given file. Reply negatively.

(C) Add the ID of the sending processor to the list of dependent processors for the True File registry entry record **140**. Reply positively, with an indication of whether the reserved True File is on line or off line.

3. Request True File

This mechanism allows a remote processor to request a copy of a True File from the local processor. It requires a True Name and responds positively by sending a True File back to the requesting processor. The mechanism operates as follows:

(A) Find the True File registry entry record **140** associated with the given True Name. If there is no such True File registry entry record **140**, reply negatively.

(B) Make the True File local using the Make True File Local primitive mechanism. If this mechanism fails, the Request True File mechanism also fails.

(C) Send the local True File in either it is uncompressed or compressed form to the requesting remote processor. Note that if the True File is a compound file, the components are not sent.

(D) If the remote file is listed in the dependent process list of the True File registry entry record **140**, remove it.

4. Retire True File

This mechanism allows a remote processor to indicate that it no longer plans to maintain a copy of a given True File. An alternate source of the True File can be specified, if, for instance, the True File is being moved from one server to another. It begins with a True Name, a requesting processor ID, and an optional alternate source. This mechanism operates as follows:

(A) Find a True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Find the requesting processor on the source list and, if it is there, remove it.

(C) If an alternate source is provided, add it to the source list for the True File registry entry record **140**.

(D) If the source list of the True File registry entry record **140** has no items in it, use the Locate Remote File primitive mechanism to search for another copy of the file. If it fails, raise a serious error.

5. Cancel Reservation

This mechanism allows a remote processor to indicate that it no longer requires access to a True File stored on the local processor. It begins with a True Name and a requesting processor ID and proceeds as follows:

(A) Find the True Name entry in the True File registry **126**. If there is no entry for this True Name, this mechanism's task is complete.

(B) Remove the identity of the requesting processor from the list of dependent processors, if it appears.

(C) If the list of dependent processors becomes zero and the use count is also zero, delete the True File.

6. Acquire True File

This mechanism allows a remote processor to insist that a local processor make a copy of a specified True File. It is used, for example, when a cache client wants to write through a new version of a file. The Acquire True File mechanism begins with a data item and an optional True Name for the data item and proceeds as follows:

(A) Confirm that the requesting processor has the right to require the local processor to acquire data items. If not, send a negative reply.

(B) Make a local copy of the data item transmitted by the remote processor.

(C) Assimilate the data item into the True File registry of the local processor.

(D) If a True Name was provided with the file, the True Name calculation can be avoided, or the mechanism can verify that the file received matches the True Name sent.

(E) Add an entry in the dependent processor list of the true file registry record indicating that the requesting processor depends on this copy of the given True File.

(F) Send a positive reply.

7. Lock Cache

This mechanism allows a remote cache client to lock a local file so that local users or other cache clients cannot change it while the remote processor is using it. The mechanism begins with a pathname and proceeds as follows:

(A) Find the local directory extensions table entry record **138** of the specified pathname. If no such entry exists, reply negatively.

(B) If an local directory extensions table entry record **138** exists and is already locked, reply negatively that the file is already locked.

(C) If an local directory extensions table entry record **138** exists and is not locked, lock the entry. Reply positively.

8. Update Cache

This mechanism allows a remote cache client to unlock a local file and update it with new contents. It begins with a pathname and a True Name. The file corresponding to the True Name must be accessible from the remote processor. This mechanism operates as follows:

Find the local directory extensions table entry record **138** corresponding to the given pathname. Reply negatively if no such entry exists or if the entry is not locked.

Link the given pathname to the given True Name using the Link Path to True Name primitive mechanism.

Unlock the local directory extensions table entry record **138** and return positively.

9. Check Expiration Date

Return current or new expiration date and possible alternative source to caller.

Background Processes and Mechanisms

The background processes and mechanisms provided by the present invention are now described. Recall that background mechanisms are intended to run occasionally and at a low priority to provide automated management capabilities with respect to the present invention.

The following background mechanisms are described:

1. Mirror True File;
2. Groom Region;
3. Check for Expired Links;
4. Verify Region; and
5. Groom Source List.

1. Mirror True File

This mechanism is used to ensure that files are available in alternate locations in mirror groups or archived on archival servers. The mechanism depends on application-specific migration/archival criteria (size, time since last access, number of copies required, number of existing alternative sources) which determine under what conditions a file should be moved. The Mirror True File mechanism operates as follows, using the True File specified, perform the following steps:

(A) Count the number of available locations of the True File by inspecting the source list of the True File registry entry record **140** for the True File. This step determines how many copies of the True. File are available in the system.

(B) If the True File meets the specified migration criteria, select a mirror group server to which a copy of the file should be sent. Use the Acquire True File remote mechanism to copy the True File to the selected mirror group server. Add the identity of the selected system to the source list for the True File.

2. Groom Region

This mechanism is used to automatically free up space in a processor by deleting data items that may be available elsewhere. The mechanism depends on application-specific grooming criteria (for instance, a file may be removed if there is an alternate online source for it, it has not been accessed in a given number of days, and it is larger than a given size). This mechanism operates as follows:

Repeat the following steps (i) to (iii) with more aggressive grooming criteria until sufficient space is freed or until all grooming criteria have been exercised. Use grooming information to determine how much space has been freed. Recall that, while grooming is in effect, grooming information includes a table of pathnames selected for deletion, and keeps track of the amount of space that would be freed if all of the files were deleted.

(i) Begin Grooming (using the primitive mechanism).

(ii) For each pathname in the specified region, for the True File corresponding to the pathname, if the True File is present, has at least one alternative source, and meets application specific grooming criteria for the region, select the file for removal (using the primitive mechanism).

(iii) End Grooming (using the primitive mechanism).

If the region is used as a cache, no other processors are dependent on True Files to which it refers, and all such True Files are mirrored elsewhere. In this case, True Files can be removed with impunity. For a cache region, the grooming criteria would ordinarily eliminate the least recently accessed True Files first. This is best done by sorting the True Files in the region by the most recent access time before performing step (ii) above. The application specific criteria would thus be to select for removal every True File encountered (beginning with the least recently used) until the required amount of free space is reached.

3. Check for Expired Links

This mechanism is used to determine whether dependencies on published files should be refreshed. The following steps describe the operation of this mechanism:

For each pathname in the specified region, for each True File corresponding to the pathname, perform the following step:

If the True File registry entry record **140** corresponding to the True File contains at least one source which is a publishing server, and if the expiration date on the dependency is past or close, then perform the following steps:

(A) Determine whether the True File registry entry record contains other sources which have not expired.

(B) Check the True Name expiration of the server. If the expiration date has been extended, or an alternate source is suggested, add the source to the True File registry entry record **140**.

(C) If no acceptable alternate source was found in steps (A) or (B) above, make a local copy of the True File.

(D) Remove the expired source.

4. Verify Region

This mechanism can be used to ensure that the data items in the True File registry **126** have not been damaged accidentally or maliciously. The operation of this mechanism is described by the following steps:

(A) Search the local directory extensions table **124** for each pathname in the specified region and then perform the following steps:

(i) Get the True File name corresponding to the pathname;

(ii) If the True File registry entry **140** for the True File does not have a True File ID or compressed file ID, ignore it.

(iii) Use the Verify True File mechanism (see extended mechanisms below) to confirm that the True File specified is correct.

5. Groom Source List

The source list in a True File entry should be groomed sometimes to ensure there are not too many mirror or archive copies. When a file is deleted or when a region definition or its mirror criteria are changed, it may be necessary to inspect the affected True Files to determine whether there are too many mirror copies. This can be done with the following steps:

For each affected True File,

(A) Search the local directory extensions table to find each region that refers to the True File.

(B) Create a set of "required sources", initially empty.

(C) For each region found,

(a) determine the mirroring criteria for that region,

(b) determine which sources for the True File satisfy the mirroring criteria, and

(c) add these sources to the set of required sources.

(D) For each source in the True File registry entry, if the source identifies a remote processor (as opposed to removable media), and if the source is not a publisher, and if the source is not in the set of required sources, then eliminate the source, and use the Cancel Reservation remote mechanism to eliminate the given processor from the list of dependent processors recorded at the remote processor identified by the source.

Extended Mechanisms

The extended mechanisms provided by the present invention are now described. Recall that extended mechanisms run within application programs over the operating system to provide solutions to specific problems and applications.

The following extended mechanisms are described:

1. Inventory Existing Directory;
2. Inventory Removable, Read-only Files;
3. Synchronize Directories;
4. Publish Region;
5. Retire Directory;
6. Realize Directory at Location;
7. Verify True File;
8. Track for Accounting Purposes; and
9. Track for Licensing Purposes.

1. Inventory Existing Directory

This mechanism determines the True Names of files in an existing on-line directory in the underlying operating system. One purpose of this mechanism is to install True Name mechanisms in an existing file system. An effect of such an installation is to eliminate immediately all duplicate files from the file system being traversed. If several file systems are inventoried in a single True File registry, duplicates across the volumes are also eliminated.

(A) Traverse the underlying file system in the operating system. For each file encountered, excluding directories, perform the following:

(i) Assimilate the file encountered (using the Assimilate File primitive mechanism). This process computes its True Name and moves its data into the True File registry **126**.

(ii) Create a pathname consisting of the path to the volume directory and the relative path of the file on the media. Link this path to the computed True Name using the Link Path to True Name primitive mechanism.

2. Inventory Removable, Read-Only Files

A system with access to removable, read-only media volumes (such as WORM disks and CD-ROMs) can create a usable inventory of the files on these disks without having to make online copies. These objects can then be used for archival purposes, directory overlays, or other needs. An operator must request that an inventory be created for such a volume.

This mechanism allows for maintaining inventories of the contents of files and data items on removable media, such as diskettes and CD-ROMs, independent of other properties of the files such as name, location, and date of creation.

The mechanism creates an online inventory of the files on one or more removable volumes, such as a floppy disk or CD-ROM, when the data on the volume is represented as a directory. The inventory service uses a True Name to identify each file, providing a way to locate the data independent of its name, date of creation, or location.

The inventory can be used for archival of data (making it possible to avoid archiving data when that data is already on a separate volume), for grooming (making it possible to delete infrequently accessed files if they can be retrieved from removable volumes), for version control (making it possible to generate a new version of a CD-ROM without having to copy the old version), and for other purposes.

The inventory is made by creating a volume directory in the media inventory in which each file named identifies the data item on the volume being inventoried. Data items are not copied from the removable volume during the inventory process.

An operator must request that an inventory be created for a specific volume. Once created, the volume directory can be frozen or copied like any other directory. Data items from either the physical volume or the volume directory can be accessed using the Open File operating system mechanism which will cause them to be read from the physical volume using the Realize True File from Location primitive mechanism.

To create an inventory the following steps are taken:

(A) A volume directory in the media inventory is created to correspond to the volume being inventoried. Its contextual name identifies the specific volume.

(B) A source table entry **144** for the volume is created in the source table **130**. This entry **144** identifies the physical source volume and the volume directory created in step (A).

(C) The file system on the volume is traversed. For each file encountered, excluding directories, the following steps are taken:

(i) The True Name of the file is computed. An entry is created in the True Name registry **124**, including the True Name of the file using the primitive mechanism. The source field of the True Name registry entry **140** identifies the source table entry **144**.

(ii) A pathname is created consisting of the path to the volume directory and the relative path of the file on the media. This path is linked to the computed True Name using Link Path to True Name primitive mechanism.

(D) After all files have been inventoried, the volume directory is frozen. The volume directory serves as a table of contents for the volume. It can be copied using the Copy File or Directory primitive mechanism to create an "overlay" directory which can then be modified, making it possible to edit a virtual copy of a read-only medium.

3. Synchronize Directories

Given two versions of a directory derived from the same starting point, this mechanism creates a new, synchronized version which includes the changes from each. Where a file is changed in both versions, this mechanism provides a user exit for handling the discrepancy. By using True Names, comparisons are instantaneous, and no copies of files are necessary.

This mechanism lets a local processor synchronize a directory to account for changes made at a remote processor. Its purpose is to bring a local copy of a directory up to date after a period of no communication between the local and remote processor. Such a period might occur if the local processor were a mobile processor detached from its server, or if two distant processors were run independently and updated nightly.

An advantage of the described synchronization process is that it does not depend on synchronizing the clocks of the local and remote processors. However, it does require that the local processor track its position in the remote processor's audit file.

This mechanism does not resolve changes made simultaneously to the same file at several sites. If that occurs, an external resolution mechanism such as, for example, operator intervention, is required.

The mechanism takes as input a start time, a local directory pathname, a remote processor name, and a remote directory pathname name, and it operates by the following steps:

(A) Request a copy of the audit file **132** from the remote processor using the Request True File remote mechanism.

(B) For each entry **146** in the audit file **132** after the start time, if the entry indicates a change to a file in the remote directory, perform the following steps:

(i) Compute the pathname of the corresponding file in the local directory. Determine the True Name of the corresponding file.

(ii) If the True Name of the local file is the same as the old True Name in the audit file, or if there is no local file and the audit entry indicates a new file is being created, link the new True Name in the audit file to the local pathname using the Link Path to True Name primitive mechanism.

(iii) Otherwise, note that there is a problem with the synchronization by sending a message to the operator or to

a problem resolution program, indicating the local pathname, remote pathname, remote processor, and time of change.

(C) After synchronization is complete, record the time of the final change. This time is to be used as the new start time the next time this directory is synchronized with the same remote processor.

4. Publish Region

The publish region mechanism allows a processor to offer the files in a region to any client processors for a limited period of time.

The purpose of the service is to eliminate any need for client processors to make reservations with the publishing processor. This in turn makes it possible for the publishing processor to service a much larger number of clients.

When a region is published, an expiration date is defined for all files in the region, and is propagated into the publishing system's True File registry entry record **140** for each file.

When a remote file is copied, for instance using the Copy File operating system mechanism, the expiration date is copied into the source field of the client's True File registry entry record **140**. When the source is a publishing system, no dependency need be created.

The client processor must occasionally and in background, check for expired links, to make sure it still has access to these files. This is described in the background mechanism Check for Expired Links.

5. Retire Directory

This mechanism makes it possible to eliminate safely the True Files in a directory, or at least dependencies on them, after ensuring that any client processors depending on those files remove their dependencies. The files in the directory are not actually deleted by this process. The directory can be deleted with the Delete File operating system mechanism.

The mechanism takes the pathname of a given directory, and optionally, the identification of a preferred alternate source processor for clients to use. The mechanism performs the following steps:

(A) Traverse the directory. For each file in the directory, perform the following steps:

(i) Get the True Name of the file from its path and find the True File registry entry **140** associated with the True Name.

(ii) Determine an alternate source for the True File. If the source IDs field of the TFR entry includes the preferred alternate source, that is the alternate source. If it does not, but includes some other source, that is the alternate source. If it contains no alternate sources, there is no alternate source.

(iii) For each dependent processor in the True File registry entry **140**, ask that processor to retire the True File, specifying an alternate source if one was determined, using the remote mechanism.

6. Realize Directory at Location

This mechanism allows the user or operating system to force copies of files from some source location to the True File registry **126** at a given location. The purpose of the mechanism is to ensure that files are accessible in the event the source location becomes inaccessible. This can happen for instance if the source or given location are on mobile computers, or are on removable media, or if the network connection to the source is expected to become unavailable, or if the source is being retired. This mechanism is provided in the following steps for each file in the given directory, with the exception of subdirectories:

(A) Get the local directory extensions table entry record **138** given the pathname of the file. Get the True Name of the local directory extensions table entry record **138**. This service assimilates the file if it has not already been assimilated.

(B) Realize the corresponding True File at the given location. This service causes it to be copied to the given location from a remote system or removable media.

7. Verify True File

This mechanism is used to verify that the data item in a True File registry **126** is indeed the correct data item given its True Name. Its purpose is to guard against device errors, malicious changes, or other problems.

If an error is found, the system has the ability to "heal" itself by finding another source for the True File with the given name. It may also be desirable to verify that the error has not propagated to other systems, and to log the problem or indicate it to the computer operator. These details are not described here.

To verify a data item that is not in a True File registry **126**, use the Calculate True Name primitive mechanism described above.

The basic mechanism begins with a True Name, and operates in the following steps:

(A) Find the True File registry entry record **140** corresponding to the given True Name.

(B) If there is a True File ID for the True File registry entry record **140** then use it. Otherwise, indicate that no file exists to verify.

(C) Calculate the True Name of the data item given the file ID of the data item.

(D) Confirm that the calculated True Name is equal to the given True Name.

(E) If the True Names are not equal, there is an error in the True File registry **126**. Remove the True File ID from the True File registry entry record **140** and place it somewhere else. Indicate that the True File registry entry record **140** contained an error.

8. Track for Accounting Purposes

This mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another. The mechanism can be used as a basis for a value-based accounting system in which charges are based on the identity of the data stored or transmitted, rather than simply on the number of bits.

This mechanism allows the system to track possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. True names make it possible to identify each file briefly yet uniquely for this purpose.

Tracking the identities of files requires maintaining an accounting log **134** and processing it for accounting or billing purposes. The mechanism operates in the following steps:

(A) Note every time a file is created or deleted, for instance by monitoring audit entries in the Process Audit File Entry primitive mechanism. When such an event is encountered, create an entry **148** in the accounting log **134** that shows the responsible party and the identity of the file created or deleted.

(B) Every time a file is transmitted for instance when a file is copied with a Request True File remote mechanism or an Acquire True File remote mechanism, create an entry in the accounting log **134** that shows the responsible party, the identity of the file, and the source and destination processors.

(C) Occasionally run an accounting program to process the accounting log **134**, distributing the events to the account records of each responsible party. The account records can eventually be summarized for billing purposes.

9. Track for Licensing Purposes

This mechanism ensures that licensed files are not used by unauthorized parties. The True Name provides a safe way to identify licensed material. This service allows proof of possession of specific files according to their contents without disclosing their contents.

Enforcing use of valid licenses can be active (for example, by refusing to provide access to a file without authorization) or passive (for example, by creating a report of users who do not have proper authorization).

One possible way to perform license validation is to perform occasional audits of employee systems. The service described herein relies on True Names to support such an audit, as in the following steps:

(A) For each licensed product, record in the license table **136** the True Name of key files in the product (that is, files which are required in order to use the product, and which do not occur in other products) Typically, for a software product, this would include the main executable image and perhaps other major files such as clip-art, scripts, or online help. Also record the identity of each system which is authorized to have a copy of the file.

(B) Occasionally, compare the contents of each user processor against the license table **136**. For each True Name in the license table do the following:

(i) Unless the user processor is authorized to have a copy of the file, confirm that the user processor does not have a copy of the file using the Locate True File mechanism.

(ii) If the user processor is found to have a file that it is not authorized to have, record the user processor and True Name in a license violation table.

The System in Operation

Given the mechanisms described above, the operation of a typical DP system employing these mechanisms is now described in order to demonstrate how the present invention meets its requirements and capabilities.

In operation, data items (for example, files, database records, messages, data segments, data blocks, directories, instances of object classes, and the like) in a DP system employing the present invention are identified by substantially unique identifiers (True Names), the identifiers depending on all of the data in the data items and only on the data in the data items. The primitive mechanisms Calculate True Name and Assimilate Data Item support this property. For any given data item, using the Calculate True Name primitive mechanism, a substantially unique identifier or True Name for that data item can be determined.

Further, in operation of a DP system incorporating the present invention, multiple copies of data items are avoided (unless they are required for some reason such as backups or mirror copies in a fault-tolerant system). Multiple copies of data items are avoided even when multiple names refer to the same data item. The primitive mechanisms Assimilate Data Items and New True File support this property. Using the Assimilate Data Item primitive mechanism, if a data item already exists in the system, as indicated by an entry in the True File registry **126**, this existence will be discovered by this mechanism, and the duplicate data item (the new data item) will be eliminated (or not added). Thus, for example, if a data file is being copied onto a system from a floppy disk, if, based on the True Name of the data file, it is determined that the data file already exists in the system (by the same or some other name), then the duplicate copy will not be installed. If the data item was being installed on the system by some name other than its current name, then, using the Link Path to True Name primitive mechanism, the other (or new) name can be linked to the already existing data item.

In general, the mechanisms of the present invention operate in such a way as to avoid recreating an actual data item at a location when a copy of that data item is already present at that location. In the case of a copy from a floppy disk, the data item (file) may have to be copied (into a scratch file) before it can be determined that it is a duplicate. This is because only one processor is involved. On the other hand, in a multiprocessor environment or DP system, each processor has a record of the True Names of the data items on that processor. When a data item is to be copied to another location (another processor) in the DP system, all that is necessary is to examine the True Name of the data item prior to the copying. If a data item with the same True Name already exists at the destination location (processor), then there is no need to copy the data item. Note that if a data item which already exists locally at a destination location is still copied to the destination location (for example, because the remote system did not have a True Name for the data item or because it arrives as a stream of un-named data), the Assimilate Data Item primitive mechanism will prevent multiple copies of the data item from being created.

Since the True Name of a large data item (a compound data item) is derived from and based on the True Names of components of the data item, copying of an entire data item can be avoided. Since some (or all) of the components of a large data item may already be present at a destination location, only those components which are not present there need be copied. This property derives from the manner in which True Names are determined.

When a file is copied by the Copy File or Directory operating system mechanism, only the True Name of the file is actually replicated.

When a file is opened (using the Open File operating system mechanism), it uses the Make True File Local primitive mechanism (either directly or indirectly through the Create Scratch File primitive mechanism) to create a local copy of the file. The Open File operating system mechanism uses the Make True File Local primitive mechanism, which uses the Realize True File from Location primitive mechanism, which, in turn uses the Request True File remote mechanism.

The Request True File remote mechanism copies only a single data item from one processor to another. If the data item is a compound file, its component segments are not copied, only the indirect block is copied. The segments are copied only when they are read (or otherwise needed).

The Read File operating system mechanism actually reads data. The Read File mechanism is aware of compound files and indirect blocks, and it uses the Realize True File from Location primitive mechanism to make sure that component segments are locally available, and then uses the operating system file mechanisms to read data from the local file.

Thus, when a compound file is copied from a remote system, only its True Name is copied. When it is opened, only its indirect block is copied. When the corresponding file is read, the required component segments are realized and therefore copied.

In operation data items can be accessed by reference to their identities (True Names) independent of their present location. The actual data item or True File corresponding to a given data identifier or True Name may reside anywhere in the system (that is, locally, remotely, offline, etc). If a required True File is present locally, then the data in the file can be accessed. If the data item is not present locally, there are a number of ways in which it can be obtained from wherever it is present. Using the source IDs field of the True File registry table, the location(s) of copies of the True File corresponding to a given True Name can be determined. The Realize True

File from Location primitive mechanism tries to make a local copy of a True File, given its True Name and the name of a source location (processor or media) that may contain the True File. If, on the other hand, for some reason it is not known where there is a copy of the True File, or if the processors identified in the source IDs field do not respond with the required True File, the processor requiring the data item can make a general request for the data item using the Request True File remote mechanism from all processors in the system that it can contact.

As a result, the system provides transparent access to any data item by reference to its data identity, and independent of its present location.

In operation, data items in the system can be verified and have their integrity checked. This is from the manner in which True Names are determined. This can be used for security purposes, for instance, to check for viruses and to verify that data retrieved from another location is the desired, and requested data. For example, the system might store the True Names of all executable applications on the system and then periodically redetermine the True Names of each of these applications to ensure that they match the stored True Names. Any change in a True Name potentially signals corruption in the system and can be further investigated. The Verify Region background mechanism and the Verify. True File extended mechanisms provide direct support for this mode of operation. The Verify. Region mechanism is used to ensure that the data items in the True File registry have not been damaged accidentally or maliciously. The Verify True File mechanism verifies that a data item in a True File registry is indeed the correct data item given its True Name.

Once a processor has determined where (that is, at which other processor or location) a copy of a data item is in the DP system, that processor might need that other processor or location to keep a copy of that data item. For example, a processor might want to delete local copies of data items to make space available locally while knowing that it can rely on retrieving the data from somewhere else when needed. To this end the system allows a processor to Reserve (and cancel the reservation of) True Files at remote locations (using the remote mechanism). In this way the remote locations are put on notice that another location is relying on the presence of the True File at their location.

A DP system employing the present invention can be made into a fault-tolerant system by providing a certain amount of redundancy of data items at multiple locations in the system. Using the Acquire True File and Reserve True File remote mechanisms, a particular processor can implement its own form of fault-tolerance by copying data items to other processors and then reserving them there. However, the system also provides the Mirror True File background mechanism to mirror (make copies) of the True File available elsewhere in the system. Any degree of redundancy (limited by the number of processors or locations in the system) can be implemented. As a result, this invention maintains a desired degree or level of redundancy in a network of processors, to protect against failure of any particular processor by ensuring that multiple copies of data items exist at different locations.

The data structures used to implement various features and mechanisms of this invention store a variety of useful information which can be used, in conjunction with the various mechanisms, to implement storage schemes and policies in a DP system employing the invention. For example, the size, age and location of a data item (or of groups of data items) is provided. This information can be used to decide how the data items should be treated. For example, a processor may implement a policy of deleting local copies of all data items over a certain age if other copies of those data items are present elsewhere in the system. The age (or variations on the age) can be determined using the time of last access or modification in the local directory extensions table, and the presence of other copies of the data item can be determined either from the Safe Flag or the source IDs, or by checking which other processors in the system have copies of the data item and then reserving at least one of those copies.

In operation, the system can keep track of data items regardless of how those items are named by users (or regardless of whether the data items even have names). The system can also track data items that have different names (in different or the same location) as well as different data items that have the same name. Since a data item is identified by the data in the item, without regard for the context of the data, the problems of inconsistent naming in a DP system are overcome.

In operation, the system can publish data items, allowing other, possibly anonymous, systems in a network to gain access to the data items and to rely on the availability of these data items. True Names are globally unique identifiers which can be published simply by copying them. For example, a user might create a textual representation of a file on system A with True Name N (for instance as a hexadecimal string), and post it on a computer bulletin board. Another user on system B could create a directory entry F for this True Name N by using the Link Path to True Name primitive mechanism. (Alternatively, an application could be developed which hides the True Name from the users, but provides the same public transfer service.)

When a program on system B attempts to open pathname F linked to True Name N, the Locate Remote File primitive mechanism would be used, and would use the Locate True File remote mechanism to search for True Name N on one or more remote processors, such as system A. If system B has access to system A, it would be able to realize the True File (using the Realize True File from Location primitive mechanism) and use it locally. Alternatively, system B could find True Name N by accessing any publicly available True Name server, if the server could eventually forward the request to system A.

Clients of a local server can indicate that they depend on a given True File (using the Reserve True File remote mechanism) so that the True File is not deleted from the server registry as long as some client requires access to it. (The Retire True File remote mechanism is used to indicate that a client no longer needs a given True File.)

A publishing server, on the other hand, may want to provide access to many clients, and possibly anonymous ones, without incurring the overhead of tracking dependencies for each client. Therefore, a public server can provide expiration dates for True Files in its registry. This allows client systems to safely maintain references to a True File on the public server. The Check For Expired Links background mechanism allows the client of a publishing server to occasionally confirm that its dependencies on the publishing server are safe.

In a variation of this aspect of the invention, a processor that is newly connected (or reconnected after some absence) to the system can obtain a current version of all (or of needed) data in the system by requesting it from a server processor. Any such processor can send a request to update or resynchronize all of its directories (starting at a root directory), simply by using the Synchronize Directories extended mechanism on the needed directories.

Using the accounting log or some other user provided mechanism, a user can prove the existence of certain data items at certain times. By publishing (in a public place) a list

of all True Names in the system on a given day (or at some given time), a user can later refer back to that list to show that a particular data item was present in the system at the time that list was published. Such a mechanism is useful in tracking, for example, laboratory notebooks or the like to prove dates of conception of inventions. Such a mechanism also permits proof of possession of a data item at a particular date and time.

The accounting log file can also track the use of specific data items and files by content for accounting purposes. For instance, an information utility company can determine the data identities of data items that are stored and transmitted through its computer systems, and use these identities to provide bills to its customers based on the identities of the data items being transmitted (as defined by the substantially unique identifier). The assignment of prices for storing and transmitting specific True Files would be made by the information utility and/or its data suppliers; this information would be joined periodically with the information in the accounting log file to produce customer statements.

Backing up data items in a DP system employing the present invention can be done based on the True Names of the data items. By tracking backups using True Names, duplication in the backups is prevented. In operation, the system maintains a backup record of data identifiers of data items already backed up, and invokes the Copy File or Directory operating system mechanism to copy only those data items whose data identifiers are not recorded in the backup record. Once a data item has been backed up, it can be restored by retrieving it from its backup location, based on the identifier of the data item. Using the backup record produced by the backup to identify the data item, the data item can be obtained using, for example, the Make True File Local primitive mechanism.

In operation, the system can be used to cache data items from a server, so that only the most recently accessed data items need be retained. To operate in this way, a cache client is configured to have a local registry (its cache) with a remote Local Directory Extensions table (from the cache server). Whenever a file is opened (or read), the Local Directory Extensions table is used to identify the True Name, and the Make True File Local primitive mechanism inspects the local registry. When the local registry already has a copy, the file is already cached. Otherwise, the Locate True File remote mechanism is used to get a copy of the file. This mechanism consults the cache server and uses the Request True File remote mechanism to make a local copy, effectively loading the cache.

The Groom Cache background mechanism flushes the cache, removing the least-recently-used files from the cache client's True File registry. While a file is being modified on a cache client, the Lock Cache and Update Cache remote mechanisms prevent other clients from trying to modify the same file.

In operation, when the system is being used to cache data items, the problems of maintaining cache consistency are avoided.

To access a cache and to fill it from its server, a key is required to identify the data item desired. Ordinarily, the key is a name or address (in this case, it would be the pathname of a file). If the data associated with such a key is changed, the client's cache becomes inconsistent; when the cache client refers to that name, it will retrieve the wrong data. In order to maintain cache consistency it is necessary to notify every client immediately whenever a change occurs on the server.

By using an embodiment of the present invention, the cache key uniquely identifies the data it represents. When the data associated with a name changes, the key itself changes. Thus, when a cache client wishes to access the modified data associated with a given file name, it will use a new key (the True Name of the new file) rather than the key to the old file contents in its cache. The client will always request the correct data, and the old data in its cache will be eventually aged and flushed by the Groom Cache background mechanism.

Because it is not necessary to immediately notify clients when changes on the cache server occur, the present invention makes it possible for a single server to support a much larger number of clients than is otherwise possible.

In operation, the system automatically archives data items as they are created or modified. After a file is created or modified, the Close File operating system mechanism creates an audit file record, which is eventually processed by the Process Audit File Entry primitive mechanism. This mechanism uses the New True File primitive mechanism for any file which is newly created, which in turn uses the Mirror True File background mechanism if the True File is in a mirrored or archived region. This mechanism causes one or more copies of the new file to be made on remote processors.

In operation, the system can efficiently record and preserve any collection of data items. The Freeze Directory primitive mechanism creates a True File which identifies all of the files in the directory and its subordinates. Because this True File includes the True Names of its constituents, it represents the exact contents of the directory tree at the time it was frozen. The frozen directory can be copied with its components preserved.

The Acquire True File remote mechanism (used in mirroring and archiving) preserves the directory tree structure by ensuring that all of the component segments and True Files in a compound data item are actually copied to a remote system. Of course, no transfer is necessary for data items already in the registry of the remote system.

In operation, the system can efficiently make a copy of any collection of data items, to support a version control mechanism for groups of the data items.

The Freeze Directory primitive mechanism is used to create a collection of data items. The constituent files and segments referred to by the frozen directory are maintained in the registry, without any need to make copies of the constituents each time the directory is frozen.

Whenever a pathname is traversed, the Get Files in Directory operating system mechanism is used, and when it encounters a frozen directory, it uses the Expand Frozen Directory primitive mechanism.

A frozen directory can be copied from one pathname to another efficiently, merely by copying its True Name. The Copy File operating system mechanism is used to copy a frozen directory.

Thus it is possible to efficiently create copies of different versions of a directory, thereby creating a record of its history (hence a version control system).

In operation, the system can maintain a local inventory of all the data items located on a given removable medium, such as a diskette or CD-ROM. The inventory is independent of other properties of the data items such as their name, location, and date of creation.

The Inventory Existing. Directory extended mechanism provides a way to create True File Registry entries for all of the files in a directory. One use of this inventory is as a way to pre-load a True File registry with backup record information. Those files in the registry (such as previously installed software) which are on the volumes inventoried need not be backed up onto other volumes.

The Inventory Removable, Read-only Files extended mechanism not only determines the True Names for the files

on the medium, but also records directory entries for each file in a frozen directory structure. By copying and modifying this directory, it is possible to create an on line patch, or small modification of an existing read-only file. For example, it is possible to create an online representation of a modified CD-ROM, such that the unmodified files are actually on the CD-ROM, and only the modified files are online.

In operation, the system tracks possession of specific data items according to content by owner, independent of the name, date, or other properties of the data item, and tracks the uses of specific data items and files by content for accounting purposes. Using the Track for Accounting Purposes extended mechanism provides a way to know reliably which files have been stored on a system or transmitted from one system to another.

True Names in Relational and Object-Oriented Databases

Although the preferred embodiment of this invention has been presented in the context of a file system, the invention of True Names would be equally valuable in a relational or object-oriented database. A relational or object-oriented database system using True Names would have similar benefits to those of the file system employing the invention. For instance, such a database would permit efficient elimination of duplicate records, support a cache for records, simplify the process of maintaining cache consistency, provide location-independent access to records, maintain archives and histories of records, and synchronize with distant or disconnected systems or databases.

The mechanisms described above can be easily modified to serve in such a database environment. The True Name registry would be used as a repository of database records. All references to records would be via the True Name of the record. (The Local Directory Extensions table is an example of a primary index that uses the True Name as the unique identifier of the desired records.)

In such a database, the operations of inserting, updating, and deleting records would be implemented by first assimilating records into the registry, and then updating a primary key index to map the key of the record to its contents by using the True Name as a pointer to the contents.

The mechanisms described in the preferred embodiment, or similar mechanisms, would be employed in such a system. These mechanisms could include, for example, the mechanisms for calculating true names, assimilating, locating, realizing, deleting, copying, and moving True Files for mirroring True Files, for maintaining a cache of True Files, for grooming True Files, and other mechanisms based on the use of substantially unique identifiers.

While the invention has been described in connection with what is presently considered to be the most practical and preferred embodiments, it is to be understood that the invention is not to be limited to the disclosed embodiment, but on the contrary, is intended to cover various modifications and equivalent arrangements included within the spirit and scope of the appended claims.

What is claimed:

1. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps of:

(A) obtaining a plurality of identifiers, each of said identifiers in said plurality of identifiers corresponding to at least one of a plurality of data items, each of said identifiers in said plurality of identifiers being based, at least in part, on a first given function of at least some of the data that comprise the contents of a corresponding one of the plurality of data items, wherein two identical data items of said plurality of data items have identical identifiers;

(B) responsive to a request, the request including at least a specific name for a particular sequence of bits, the specific name having been determined, at least in part, using a second given function of the particular sequence of bits, wherein two identical sequences of bits have the same name as determined using the second given function, and wherein the first given function is the same as the second given function that was used to determine the specific name for the particular sequence of bits, hardware in combination with software, ascertaining whether or not the specific name for the particular sequence of bits corresponds to an identifier in said plurality of identifiers; and,

(C) based at least in part on said ascertaining in step (B), selectively permitting at least one copy of the particular sequence of bits to be distributed across or accessed by or from a plurality of computers in a network, wherein a copy of the particular sequence of bits is not permitted to be distributed or accessed without authorization, as determined based, at least in part, on whether or not the specific name for the particular sequence of bits corresponds to an identifier in said plurality of identifiers.

2. A method as in claim **1**, wherein a copy of the particular sequence of bits is not permitted to be distributed on behalf of unlicensed parties or unauthorized parties.

3. A method as in claim **1**, wherein distribution of or accessing an unauthorized copy of a sequence of bits is not allowed.

4. A method as recited in claim **3** further comprising:

permitting at least one copy of the particular sequence of bits to be distributed in a network when it is not determined that the particular sequence of bits is unauthorized.

5. The method of claim **1** wherein distribution of or accessing an unlicensed copy of a sequence of bits is not allowed.

6. The method of claim **1** wherein the specific name for the particular sequence of bits corresponds to an identifier of the plurality of identifiers when the specific name for the particular sequence of bits exactly matches the identifier of the plurality of identifiers.

7. A method as recited in claim **1** wherein the specific name of the particular sequence of bits is based, at least in part, on a function of all of the bits of the particular sequence of bits.

8. The method of claim **1** further comprising:

denying permission for the particular sequence of bits to be distributed in a network when it is determined that access to the particular sequence of bits is not authorized.

9. The method of claim **1** further comprising:

maintaining the list plurality of identifiers.

10. The method of claim **9** wherein said maintaining the plurality of identifiers comprises:

adding new identifiers to said plurality of identifiers.

11. The method of claim **1** further comprising:

using the specific name for the particular sequence of bits to determine whether the particular sequence of bits is present in a data processing system.

12. The method of claim **1** wherein the particular sequence of bits represent data selected from the group comprising: a file, a portion of a file, a page in memory, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and a identifier of a data item.

13. A computer-implemented method, implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) obtaining a specific name for a particular sequence of bits, the specific name having been determined at least in part as a first given function of at least some of the sequence of bits, wherein two identical sequences of bits will have the same name, as determined using the first given function; and

(B) ascertaining, by hardware in combination with software, whether or not the specific name for the particular sequence of bits corresponds to an identifier in a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each of said plurality of identifiers being based, at least in part, on a second given function of the contents of a corresponding one of the plurality of data items, wherein two identical data items have identical identifiers, as determined by said second given function, and wherein the second given function is the same as the first given function; and

(C) based at least in part on said ascertaining in step (B), selectively allowing a copy of the particular sequence of bits to be distributed to or provided or accessed by or from at least one of the computers in said plurality of computers, wherein a copy of the sequence of bits is not to be distributed or provided or accessed without authorization, as determined based, at least in part, on whether or not the specific name for the particular sequence of bits corresponds to one of said plurality of identifiers.

14. A method as in claim 13, further comprising:

maintaining accounting information relating to at least some data items in the system; and

using the accounting information as a basis for charges based on an identity of the at least some data items.

15. A method as in claim 13, further comprising:

collecting information relating to the particular sequence of bits.

16. The method of claim 13 wherein the specific name for the particular sequence of bits is based, at least in part, on a size or length of the particular sequence of bits.

17. The method of claim 13 wherein the specific name for the particular sequence of bits comprises a digital fingerprint of the particular sequence of bits.

18. The method of claim 13 wherein the specific name for the particular sequence of bits is a True Name of the particular sequence of bits.

19. The method of any one of claims 1 and 13 wherein the specific name for the particular sequence of bits is a based on all of the bits and only the bits in the particular sequence of bits.

20. The method of claim 13 further comprising:

obtaining a copy of the particular sequence of bits; and

wherein the step (A) of obtaining the specific name for the particular sequence of bits comprises:

hardware in combination with software, determining the specific name for the particular sequence of bits using the copy of the particular sequence of bits.

21. The method of claim 13 further comprising:

obtaining the plurality of identifiers.

22. The method of claim 13 wherein, when contents of the particular sequence of bits represent an audio signal or a portion of an audio signal, the specific name of the particular sequence of bits is a function of at least some of the data comprising the audio signal; and

when contents of the particular sequence of bits represent a video signal or a portion of a video signal, the specific name of the particular sequence of bits is a function of at least some of the data comprising the video signal.

23. The method of any one of claims 1 and 13, wherein the specific name for the particular sequence of bits comprises a request for the particular sequence of bits.

24. The method of any one of claims 1 and 13, wherein the specific name for the particular sequence of bits was determined using only bits in the sequence of bits.

25. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) hardware in combination with software, determining a first content-dependent name for a particular sequence of bits, at least in part by applying a particular function to at least some of the particular sequence of bits, said particular function comprising a message digest function or a hash function, wherein two identical sequences of bits will have the same content-dependent name as determined using said particular function;

(B) ascertaining whether or not said first content-dependent name for the particular sequence of bits corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each identifier of said plurality of identifiers being based, at least in part, on a first given function of the data that comprise the contents of a corresponding one of the plurality of data items, wherein said first given function comprises the particular function used to determine the first content-dependent name for said particular sequence of bits; and,

(C) based at least in part on said ascertaining in step (B), selectively allowing a copy of the particular sequence of bits to be provided to or accessed by or from at least one of the computers in a network of computers, wherein a copy of the sequence of bits is not to be provided or accessed without authorization, as determined, at least in part, based on whether or not said first content-dependent name of the particular sequence of bits corresponds to one of the plurality of identifiers.

26. The method as in 25 wherein the plurality of identifiers is in a table comprising said plurality of identifiers.

27. The method as in 25 wherein the plurality of identifiers is in a database comprising said plurality of identifiers.

28. The method of claim 25 wherein the first content-dependent name for the particular sequence of bits is based, at least in part, on a size or length of the particular sequence of bits.

29. The method of claim 25 wherein the first content-dependent name for the particular sequence of bits comprises a digital fingerprint of the particular sequence of bits.

30. The method of claim 25 wherein the first content-dependent name for the particular sequence of bits is a True Name of the particular sequence of bits.

31. The method of claim 22 wherein the first content-dependent name for the particular sequence of bits is a based on all of the bits and only the bits in the particular sequence of bits.

32. The method of claim 25 where said first content-dependent name of the particular sequence of bits corresponds to an identifier of the first plurality of identifiers when said first content-dependent name of the particular sequence of bits exactly matches the identifier of the first plurality of identifiers.

33. The method of claim 25 further comprising:

(B2) obtaining a second content-dependent name for said at least one particular sequence of bits, the second content-dependent name being based at least in part on a

second function of at least some data comprising the particular sequence of bits, said second content-dependent name being distinct from said first content-dependent name; and

(C2) ascertaining whether or not said second content-dependent name for the at least one particular sequence of bits corresponds to an identifier of a second plurality of identifiers, each identifier of said second plurality of identifiers being based, at least in part, on a second given function of at least some of the data that comprise the contents of a corresponding one of a plurality of data items, said second given function corresponding to said second function, said second given function being distinct from said first given function;

wherein said selectively allowing in step (C) is based, at least in part, based on whether or not said second content-dependent name of the particular sequence of bits corresponds to one of the second plurality of identifiers.

34. The method of claim **25**, wherein the first content-dependent name for the particular sequence of bits was determined using all of the bits in the sequence of bits.

35. The method of claim **25**, wherein the first content-dependent name for the particular sequence of bits was determined using only bits in the sequence of bits.

36. The method of any one of claims **1**, **13**, and **25** wherein particular sequence of bits comprise a file.

37. The method of claim **25** operable in a system comprising one or more zones.

38. The method of claim **37**, wherein at least some of said one or more zones are regions.

39. The method of claim **37**, wherein at least one of said one or more zones is used for access control.

40. The method of claim **37**, wherein at least one of said one or more zones resides on a shared file server.

41. The method of claim **37** wherein the particular sequence of bits is associated with a particular zone, and wherein the act of selectively allowing in (C) is based, at least in part, on at least one policy associated with the particular zone.

42. The method of claim **41** wherein the particular zone is determined using at least the first content-dependent name of the particular sequence of bits.

43. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps of:

(A) hardware in combination with software, determining a first content-dependent digital identifier for a particular sequence of bits, said determining using at least a first given function of some data, wherein the data used by the first given function to determine the first content-dependent digital identifier for the particular sequence of bits comprises at least some of the bits in the sequence of bits, wherein two identical sequences of bits will have the same digital identifier as determined by the first given function; and

(B) selectively allowing the particular sequence of bits to be provided to or accessed by or from at least one computer of a network of computers, wherein a copy of the sequence of bits is not allowed to be provided or accessed without authorization, as resolved, based, at least in part, on whether or not said first content-dependent digital identifier for the particular sequence of bits corresponds to an entry in a first database, said first database comprising a first plurality of identifiers, each of said identifiers in said first plurality of identifiers being based, at least in part, on a function of the data that comprise the contents of a corresponding one of a plurality of data items, and wherein the first given function used in step (A) to determine the content-dependent digital identifier of the specific sequence of bits is the same as the function that was used to determine the first plurality of identifiers in the first database.

44. The method of claim **43** where said first content-dependent digital identifier for the particular sequence of bits corresponds to an entry in said first database when said first content-dependent digital identifier for the particular sequence of bits exactly matches the entry in the first database.

45. The method of claim **43** further comprising:

(B2,) obtaining a second content-dependent digital identifier for the particular sequence of bits, said second content-dependent digital identifier being based on at least a second given function of the particular sequence of bits;

and wherein, in step (B), said selectively allowing is resolved, based, at least in part, on one or more of:

(i) whether or not said first content-dependent digital identifier for the particular sequence of bits corresponds to an entry in the first database, and

(ii) whether or not said second content-dependent digital identifier for the particular sequence of bits corresponds to an entry in a second database, said second database comprising a second plurality of identifiers, each of said identifiers in said second plurality of identifiers being based, at least in part, on a second function of the data that comprise the contents of a corresponding one of a plurality of data items, and wherein the second given function used to determine the second content-dependent digital identifier of the particular sequence of bits corresponds to the second function that was used to determine the identifiers in the second database.

46. The method of claim **43**, wherein the first content-dependent digital identifier for the particular sequence of bits was determined using all of the bits in the sequence of bits.

47. The method of claim **43**, wherein the first content-dependent digital identifier for the particular sequence of bits was determined using only bits in the sequence of bits.

48. The method as in any one of claims **1**, **13** and **43** wherein the first given function is a cryptographic hash function.

49. The method of claim **43** further comprising the step of: obtaining a copy of the particular sequence of bits.

50. A method as recited in any of claims **1**, **13**, and **4**, wherein access to said sequence of bits comprises one or more of (a) copying the sequence of bits to or from at least one of a plurality of computers; (b) providing the sequence of bits to at least one of a plurality of computers; (c) reading the sequence of bits; (d) copying the sequence of bits; (e) distributing the sequence of bits; (f) modifying the sequence of bits; (g) storing the sequence of bits; (h) opening the sequence of bits; (i) publishing the sequence of bits; (j) writing the sequence of bits; (k) moving the sequence of bits; and (l) deleting the sequence of bits.

51. The method of claim **43** operable in a system comprising one or more zones.

52. The method of claim **51**, wherein at least some of said one or more zones are regions.

53. The method of claim **51**, wherein at least one of said one or more zones is used for access control.

54. The method of claim **51**, wherein at least one of said one or more zones resides on a shared file server.

55. The method of claim **51**, wherein the particular sequence of bits is associated with a particular zone, and

wherein the act of selectively allowing in (B) is based, at least in part, on at least one policy associated with the particular zone.

56. The method of claim **55** wherein the particular zone is determined using at least the first content-dependent digital identifier of the particular sequence of bits.

57. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) selectively permitting distribution of a set of data items in a network of servers by, for at least one particular data item in said set of data items:

(a1) said particular data item comprising a particular sequence of bits representing at least one of: a file, a portion of a file, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and an identifier of a data item,

the particular data item having at least one contextual name specifying at least one location of the particular data item;

hardware in combination with software, determining another name for the particular data item, the other name for the particular data item being distinct from the at least one contextual name of the particular data item, the other name for the particular data item comprising a content-dependent digital identifier for the particular data item which was determined at least in part using a given function of some data, said given function comprising a message digest function or a hash function, and where the data used by the given function to determine the content-dependent digital identifier for the particular data item comprises at least some of the bits in the particular sequence of bits comprising the particular data item, wherein two identical data items will have the same digital identifier as determined by the given function;

(a2) ascertaining whether or not the content-dependent digital identifier for the particular data item corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each identifier of said plurality of identifiers being based, at least in part, on a second given function of the data that comprise the contents of a corresponding one of the plurality of data items, wherein two identical data items will have the same identifier as determined using said second given function, and wherein the second given function is the same as the given function used to determine the content-dependent digital identifier for the particular data item; and,

(a3) based at least in part on said ascertaining in (a2), selectively permitting of a copy of the particular data item to be provided or accessed from at least one of the servers of the network of servers, said permitting being based at least in part on the content-dependent digital identifier of the particular data item, and wherein a copy of the particular data item is not allowed to be provided or accessed without authorization, as resolved based, at least in part, on whether or not the content-dependent digital identifier of the particular data item corresponds to one of the plurality of identifiers.

58. The method of claim **57** where said content-dependent digital identifier for the particular data item corresponds to one of a plurality of identifiers when said content-dependent digital identifier for the particular data item matches the one of a plurality of identifiers.

59. The method of claim **57** wherein said selectively permitting a copy of the particular data item to be accessed in step (a3) comprises selectively permitting one or more of:

(a) copying the particular data item; (b) providing the particular data item to at least one of a plurality of computers; (c) reading the particular data item; (d) distribution of the particular data item; (e) modification of the particular data item; (f) storage of the particular data item; (g) opening of the particular data item; (h) publishing the particular data item; (i) writing the particular data item; (k) moving the particular data item; and (l) deleting the particular data item.

60. The method of claim **57**, wherein the other name for the particular data item is determined using all of the bits in the sequence of bits comprising the particular data item.

61. The method of claim **57**, wherein the other name for the particular data item is determined using only the sequence of bits comprising the particular data item.

62. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) obtaining a content-based name for a particular sequence of bits, said content-based name comprising at least a value having been determined at least in part by applying a first given function comprising a message digest function or hash function to at least a portion of the particular sequence of bits, wherein two identical sequences of bits will have the same content-based name as determined by said first given function,

said particular sequence of bits comprising data representing at least one of: a file, a portion of a file, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and an identifier of a data item; and

(B) selectively permitting distribution of at least one copy of the particular sequence of bits across a network of servers by:

(b1) hardware in combination with software, ascertaining whether or not the content-based name of the particular sequence of bits corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each of said identifiers being based, at least in part, on a second given function of data that comprise the contents of a corresponding one of the plurality of data items, wherein the second given function is the same as the first given function; and

(b2) based at least in part on said ascertaining in (b1), selectively permitting distribution of at least one copy of the particular sequence of bits in the network of computers, wherein a copy of the particular sequence of bits is not to be provided without authorization, as determined based, at least in part, on whether or not said content-based name of the particular sequence of bits corresponds to one of said plurality of identifiers.

63. A method as in any one of claims **25**, **57**, and **62** wherein the message digest function or hash function is selected from the functions: MD4, MD5, and SHA.

64. The method of claim **62** where said content-based name of the particular sequence of bits corresponds to an entry on said list of identifiers

when said content-based name of the particular sequence of bits matches the entry on said list of identifiers.

65. The method of claim 62, wherein the content-based name for the particular sequence of bits is determined using all of the bits in the particular sequence of bits.

66. The method of claim 62, wherein the content-based name for the particular sequence of bits is determined using only the bits in the particular sequence of bits.

67. The method of claim 62 operable in a system comprising one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization is based, at least in part, on at least one policy associated with the particular zone.

68. The method of claim 67, wherein at least some of said one or more zones are regions.

69. The method of claim 67 wherein the particular zone is determined using at least the content-based name of the particular sequence of bits.

70. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) for a particular data item in a set of data items, said particular data item comprising a corresponding particular sequence of bits:

(a1) hardware in combination with software, determining one or more content-dependent digital identifiers for said particular data item, each said content-dependent digital identifier being based at least in part on a given function of at least some of the bits in the particular sequence of bits of the particular data item, wherein two identical data items will have the same digital identifiers as determined using said given function; and

(a2) selectively permitting the particular data item to be made available for access and to be provided to or accessed by or from at least some of the computers in a network of computers, wherein the data item is not to be made available for access or provided without authorization, as resolved based, at least in part, on whether or not at least one of said one or more content-dependent digital identifiers for said particular data item corresponds to an entry in one or more databases,

each of said one or more databases comprising a plurality of identifiers, each of said identifiers in each said database corresponding to at least one data item of a plurality of data items, and each of said identifiers in each said database being based, at least in part, on at least some of the data in a corresponding data item.

71. A method as in claim 70, further comprising:

maintaining accounting information relating to at least some of said data items; and

using the accounting information as a basis for a system in which charges are based on an identity of data items.

72. A method as in claim 71, wherein the maintaining of accounting information includes at least some of activities selected from:

(a) tracking which data items have been stored on a computer; and

(b) tracking which data items have been transmitted from a computer.

73. The method of claim 70 where at least one of said one or more content-dependent digital identifiers corresponds to an entry in one or more databases when at least one of said one or more of said content-dependent digital identifiers matches an entry in one or more of said databases.

74. The method of claim 70 wherein at least one of said one or more content-dependent digital identifiers for said particular data item is based, at least in part, on a size or length of the particular data item.

75. The method of claim 70 wherein at least two content-dependent digital identifiers for the particular data item have different degrees of uniqueness.

76. The method of claim 70, wherein at least one of the one or more digital identifiers for the particular data item is determined using all of the bits in the sequence of bits comprising the particular data item.

77. The method of claim 70, wherein at least one of the one or more digital identifiers for the particular data item is determined using only bits in the sequence of bits comprising the particular data item.

78. A method as recited in claim 70 wherein access to said particular data item comprises one or more of: (a) copying the said particular data item to or from at least one of a plurality of computers; (b) providing the particular data item to at least one of a plurality of computers; (c) reading the said particular data item; (d) copying said particular data item; (e) distributing said particular data item; (f) modifying said particular data item; (g) storing said particular data item; (h) opening said particular data item; (i) publishing said particular data item; (j) writing said particular data item; (k) moving said particular data item; and (l) deleting said particular data item.

79. The method of claim 70 operable in a system comprising one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the authorization is based, at least in part, on at least one policy associated with the particular zone.

80. The method of claim 79, wherein at least some of said one or more zones are regions.

81. The method of claim 79 wherein the particular zone is determined using at least one of the content-dependent digital identifiers for said particular data item.

82. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps of:

(A) obtaining a request to distribute a particular data item in a network of computers, said particular data item comprising a particular sequence of bits;

(B) in response to the request, hardware in combination with software, obtaining a substantially unique identifier for the particular data item, said substantially unique identifier being based at least in part on a first function of at least some of the bits in the particular sequence of bits of the particular data item, wherein two identical data items will have the same identifier based on said first function; and

(C) ascertaining whether or not the substantially unique identifier of the particular data item corresponds to one of a plurality of identifiers, each of said identifiers being based, at least in part, on a given function of the data that comprise the contents of a corresponding one of a plurality of data items, wherein the given function used to determine said plurality of identifiers is the same as the first function that was used to determine the substantially unique identifier of the particular data item; and,

(D) based at least in part on said ascertaining in step (C), selectively allowing the particular data item to be distributed to at least some of the computers in the network of computers, wherein a copy of the data item is not allowed to be distributed without authorization, and wherein the request to distribute the particular file across the network of computers is resolved based, at least in

part, on whether or not the substantially unique identifier of the data item corresponds to one of the plurality of identifiers; and

(E) collecting information regarding the particular data item.

83. A method as in claim **82**, further comprising:

using the information collected in step (E) as a basis for a system in which charges are based on an identity of data items.

84. A method as in claim **82**, further comprising the step of:

maintaining accounting information relating to at least some data items,

wherein the maintaining of accounting information includes at least some of activities selected from:

(a) tracking which data items have been stored at a location; and

(b) tracking which data items have been transmitted from a location.

85. The method as in any one of claims **1**, **13**, **62**, and **82** wherein the plurality of identifiers is in table comprising said plurality of identifiers.

86. The method as in any one of claims **1**, **13**, **62**, and **82** wherein the plurality of identifiers is in a database comprising said plurlaity of identifiers.

87. The method of claim **86** wherein said database maps each said identifier in said database to information relating to a corresponding data item.

88. The method of claim **82** where

the substantially unique identifier of the particular data item corresponds to one of a plurality of identifiers when the substantially unique identifier of the particular data item exactly matches the one of the plurality of identifiers.

89. The method of claim **82** wherein the information collected regarding the particular data item in step (E) includes at least one of:

(a) information about whether or not the particular data item was read; (b) information about content of the particular data item, (c) information about an owner of the particular data item, (d) information about a type of particular data item, (e) information about a contextual name of the particular data item, (f) information about whether or not the particular data item was copied; (g) a name of the particular data item; (h) information about an identity of a requestor; (i) a timestamp; and (j) information about whether or not the particular data item was created.

90. A method as recited in claim **82** wherein at least some of the information collected is maintained for accounting or billing purposes.

91. The method of claim **82**, wherein the substantially unique identifier for the particular data item is determined using all of the bits in the sequence of bits comprising the particular data item.

92. The method of claim **82**, wherein the substantially unique identifier for the particular data item is determined using only bits in the sequence of bits comprising the particular data item.

93. The method of claim **82** operable in a system comprising one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the authorization is based, at least in part, on at least one policy associated with the particular zone.

94. The method of claim **93**, wherein at least some of said one or more zones are regions.

95. The method of claim **93** wherein the particular zone is determined using at least the substantially unique identifier for the particular data item.

96. A computer-implemented method, operable in a data processing system and implemented at least in part by hardware in combination with software, the method comprising:

(A) in response to a request to allow distribution of a particular data item in a network of computers, hardware in combination with software, ascertaining whether or not a content-dependent particular name of the particular data item corresponds to an entry in a database,

said database comprising a plurality of identifiers, each of said identifiers in said database having been determined based, at least in part, as a first given function of the data that comprise the contents of a corresponding one of a plurality of data items, and

said particular data item comprising a particular sequence of bits, said content-dependent particular name for the particular data item being based, at least in part, on a second given function of said particular sequence of bits, and wherein two identical sequences of bits will have the same name, as determined using said second given function, and wherein the first given function is the same as the second given function; and

(B) based at least in part on said ascertaining in (A), selectively permitting the particular data item to be distributed in the network of computers; and

(C) collecting and maintaining information relating to the particular data item, wherein the information collected includes at least one of information selected from one or more of: (a) information about the content of the particular data item, (b) information about an owner of the particular data item, (c) information about a type of the particular data item, (d) information about a contextual name of the particular data item, (e) information about whether or not the particular data item was copied, (f) the content-based name of the particular data item, (g) information about an identity of a requestor, (h) a timestamp; (i) information about whether or not the particular data item was created, and (j) information about whether or not the particular data item was read.

97. A method as in claim **96**, wherein some of the computers communicate with each other using a TCP/IP communication protocol.

98. The method as in any one of claims **1-62**, **70**, **82**, and **96**, wherein the hardware comprises a processor and memory.

99. The method as in any one of claims **1-62**, **70**, **82**, and **96**, wherein the hardware comprises a computer.

100. The method of claim **96** wherein the identifiers in the database are identifiers of licensed content items.

101. The method of claim **96** wherein the database comprises a table or a list identifying data items which may only be used by licensed users.

102. The method of claim **96** wherein the database comprises a table identifying license information associated with data items, and wherein the table is indexed at least by the identifiers of data items.

103. The method of claim **96** wherein, wherein the database comprises a table containing a plurality of records, and wherein a record of the table records a relationship between a specific data item and users licensed to have access to that specific data item.

104. The method of claim **96** where

the content-dependent particular name of the particular data item corresponds to an entry in a database when the content-dependent particular name of the particular data item matches the entry in the database.

105. The method of claim 96, wherein the particular name for the particular data item is based on all the bits in the sequence of bits comprising the particular data item.

106. The method of claim 96, wherein the particular name for the particular data item is based on only bits in the sequence of bits comprising the particular data item.

107. The method as in any one of claims 1, 13, 25, 43, 57, 62, 70, 82 and 96, wherein the particular sequence of bits is an arbitrary sequence of bits.

108. The method as in any one of claims 1, 13, 25, 43, 57, 62, 70, 82 and 96, wherein the particular sequence of bits comprises a file or a portion of a file.

109. The method of claim 96 wherein the data processing system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the selectively permitting in (B) is based, at least in part, on at least one policy associated with the particular zone.

110. The method of claim 109, wherein at least some of said one or more zones are regions.

111. The method of claim 109 wherein the particular zone is determined using at least the content-dependent particular name of the particular data item.

112. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) obtaining a particular data item, said particular data item comprising multiple parts;

(B) hashing a plurality of said multiple parts to determine a plurality of part hash values for said particular data item, wherein identical parts have the same hash value; and

(C) by hardware in combination with software, applying a cryptographic hash function or message digest function to said plurality of part hash values to determine a data identifier for the particular data item, said data identifier for the particular data item being determined and depending on the plurality of part hash values for the particular data item, wherein two identical data items will have the same identifier; and

(D) ascertaining, using said data identifier for said particular data item, whether or not said particular data item corresponds to another data item of a plurality of data items, said ascertaining comprising determining whether or not said data identifier for said particular data item corresponds to any of a plurality of identifiers corresponding to said plurality of data items; and,

(E) based at least in part on said ascertaining in step (D), accessing or permitting access to the particular data item.

113. The method of claim 112 wherein access to said particular data item comprises one or more of: (a) copying the said particular data item to or from at least one of a plurality of computers; (b) providing the particular data item to at least one of a plurality of computers; (c) reading the said particular data item; (d) copying said particular data item; (e) distributing said particular data item; (f) modifying said particular data item; (g) storing said particular data item; (h) opening said particular data item; (i) publishing said particular data item; (j) writing said particular data item; (k) moving said particular data item; and (l) deleting said particular data item.

114. The method of claim 113 wherein the particular data item is a compound data item.

115. The method of claim 114 wherein the plurality of parts comprise a plurality of segments.

116. The method of claim 112 where each of the plurality of parts consists of a sequence of data segments.

117. The method of claim 112 wherein a majority of the plurality of parts have the same size.

118. The method of claim 112 wherein the particular data item comprises a file or a portion of a file.

119. The method of claim 112 wherein the cryptographic hash function used in step (C) uses a function selected from the functions: MD4, MD5, and SHA.

120. The method of claim 112 wherein the hashing in step (B) uses a single hash function.

121. The method of claim 120 wherein the single hash function is a message digest function or a cryptographic hash function.

122. The method of claim 112 wherein said particular data item corresponds to another data item of a plurality of data items when said particular data item is identical to the other data item.

123. The method of claim 112 wherein said particular data item corresponds to another data item of a plurality of data items when said data identifier for said particular data item exactly matches a corresponding data identifier for the other data item.

124. The method of claim 112 operable in a system comprising one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the accessing or permitting access to the particular data item in (E) is based, at least in part, on at least one policy associated with the particular zone.

125. The method of claim 124, wherein at least some of said one or more zones are regions.

126. The method of claim 124 wherein the particular zone is determined using at least the data identifier for said particular data item.

127. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising:

(A) hardware in combination with software, obtaining a first digital name for a particular sequence of bits, said first digital name being based at least in part on a first given function of some data, wherein the data used by the first given function to determine the first digital name for the particular sequence of bits comprises at least some of the bits in the sequence of bits, wherein two identical sequences of bits will have the same digital name as determined by the given function; and

(B) selectively allowing the particular sequence of bits to be provided to or accessed by or from at least one location, wherein a copy of the sequence of bits is not allowed to be provided or accessed without authorization, as resolved, based, at least in part, on whether or not said first digital name for the particular sequence of bits corresponds to an entry in a database, said database comprising a plurality of names, each of said names in said plurality of names being based, at least in part, on a function of the data that comprise the contents of a corresponding one of a plurality of data items, and wherein the function that was used to determine the first plurality of names in the database is the same as the first given function.

128. The method of claim 127 operable in a system comprising one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular

zone, and wherein the selectively allowing in (B) is based, at least in part, on at least one policy associated with the particular zone.

129. The method of claim **128**, wherein at least some of said one or more zones are regions.

130. The method of claim **128** wherein the particular zone is determined using at least the first digital name for the particular sequence of bits.

131. The method of claims **37**, **51**, **67**, **79**, **93**, **109**, **124**, **128**, wherein the system comprises an object addressable storage system.

132. A computer-implemented method comprising:

(A) hashing each of a plurality of parts to determine a corresponding plurality of part hash values, each of said parts comprising an arbitrary sequence of bits, and wherein identical parts have the same part hash value; and

(B) by hardware in combination with software, applying a given function to said plurality of part hash values to determine a digital value, said digital value being determined and depending on the plurality of part hash values, wherein two identical pluralities of part hash values will have the same digital value as determined using said given function; and

(C) ascertaining, using said digital value, whether or not said plurality of parts corresponds to at least one data item of a plurality of data items, based, at least in part, on whether or not said digital value corresponds to any of a plurality of values corresponding to said plurality of data items; and,

(D) based at least in part on said ascertaining in (C), providing information about said at least one data item when said digital value corresponds to said at least one data item.

133. The method of claim **132** wherein the plurality of parts comprise a plurality of segments.

134. The method of claim **132** wherein the given function is a hash or message digest function.

135. The method of claim **134** wherein the given function comprises a function selected from the functions: MD4, MD5, and SHA.

136. The method of claim **132** wherein the hashing in (A) uses a single hash function.

137. The method of claim **132** wherein said plurality of parts corresponds to at least one data item when said digital value is identical to a digital value of the at least one data item.

138. The method of claim **132** wherein said information provided about said at least one data item comprises information about a location having a replica of said at least one data item.

139. The method of claim **138**, wherein said location is a server in a network.

140. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising:

(A) by hardware in combination with software, ascertaining a digital value corresponding to a plurality of parts,

each of said parts comprising an arbitrary sequence of bits, each of said parts having a corresponding part value, the corresponding part value of each part having been determined by applying a first function to said each part, wherein identical parts have the same part value as determined by said first function, wherein said first function comprises a first hash function; and

the digital value having been determined using a second function, and depending on the plurality of part values, wherein two identical pluralities of part values will have the same digital value as determined using said second function, wherein said second function comprises a second hash function; and

(B) ascertaining, using said digital value, whether or not said plurality of parts corresponds to at least one data item of a plurality of data items, said ascertaining comprising determining whether or not said digital value corresponds to any of a plurality of values corresponding to said plurality of data items; and,

(C) based at least in part on said ascertaining in (B), when said digital value corresponds to said at least one data item, providing information about said at least one data item, wherein said information provided about said at least one data item comprises information about a location having a replica of said at least one data item, wherein said location comprises a server in a network.

141. A computer-implemented method implemented at least in part by hardware in combination with software, the method comprising the steps:

(A) obtaining a first name for a particular data item, said particular data item comprising multiple parts, wherein said first name is determined, at least in part, by applying a first function to a plurality of part values,

said plurality of part values having been determined by applying a second function to a plurality of said multiple parts, wherein said second function comprises a second hash function, and wherein identical parts have the same part value as determined using said second function,

said first name for the particular data item having been determined and depending on the plurality of part values, wherein two identical pluralities of parts will have the same name as determined using said first function,

wherein said first function comprises a first hash or message digest function; and

(B) by hardware in combination with software, ascertaining, using said first name for said particular data item, whether or not said particular data item corresponds to another data item of a plurality of data items, said ascertaining comprising determining whether or not said first name for said particular data item corresponds to any of a plurality of names corresponding to said plurality of data items; and,

(C) based at least in part on said ascertaining in (B), accessing or permitting access to the particular data item.

142. A system comprising hardware including at least one processor, and software, in combination with said hardware:

(a) to obtain a plurality of identifiers, each of said identifiers in said plurality of identifiers corresponding to at least one of a plurality of data items, each of said identifiers in said plurality of identifiers being based, at least in part, on a first given function of at least some of the data that comprise the contents of a corresponding one of the plurality of data items, wherein two identical data items of said plurality of data items have identical identifiers;

(b) to ascertain, responsive to a request, whether or not a specific name for a particular sequence of bits corresponds to an identifier in said plurality of identifiers,

the request including at least the specific name for the particular sequence of bits, the specific name having been determined, at least in part, using a second given function of the particular sequence of bits, wherein two identical sequences of bits have the same name as determined using the second given function, and wherein the first given function is the same as the second given

function that was used to determine the specific name for the particular sequence of bits, hardware in combination with software; and,

(c) selectively to permit at least one copy of the particular sequence of bits to be distributed across or accessed by or from a plurality of computers in a network, wherein a copy of the particular sequence of bits is not permitted to be distributed or accessed without authorization, as determined based, at least in part, on whether or not the specific name for the particular sequence of bits corresponds to an identifier in said plurality of identifiers.

143. The system of claim **142** wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization in (c) is based, at least in part, on at least one policy associated with the particular zone.

144. The system of claim **143**, wherein at least some of said one or more zones are regions.

145. The system of claim **143** wherein the particular zone is determined using at least the specific name for the particular sequence of bits.

146. A system comprising hardware including at least one processor, and software, in combination with said hardware:

(a) to obtain a specific name for a particular sequence of bits, the specific name having been determined at least in part as a first given function of at least some of the sequence of bits, wherein two identical sequences of bits will have the same name, as determined using the first given function; and

(b) to ascertain whether or not the specific name for the particular sequence of bits corresponds to an identifier in a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each of said plurality of identifiers being based, at least in part, on a second given function of the contents of a corresponding one of the plurality of data items, wherein two identical data items have identical identifiers, as determined by said second given function, and wherein the second given function is the same as the first given function; and

(c) selectively to allow a copy of the particular sequence of bits to be distributed to or provided or accessed by or from at least one of the computers in said plurality of computers, wherein a copy of the sequence of bits is not to be distributed or provided or accessed without authorization, as determined based, at least in part, on whether or not the specific name for the particular sequence of bits corresponds to one of said plurality of identifiers.

147. The system of claim **146** wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization in (c) is based, at least in part, on at least one policy associated with the particular zone.

148. The system of claim **147**, wherein at least some of said one or more zones are regions.

149. The system of claim **147** wherein the particular zone is determined using at least the specific name for the particular sequence of bits.

150. A system comprising hardware including at least one processor, and software, in combination with said hardware:

(a) to determine a first content-dependent name for a particular sequence of bits, at least in part by applying a particular function to at least some of the particular sequence of bits, said particular function comprising a message digest function or a hash function, wherein two identical sequences of bits will have the same content-dependent name as determined using said particular function;

(b) to ascertain whether or not said first content-dependent name for the particular sequence of bits corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each identifier of said plurality of identifiers being based, at least in part, on a first given function of the data that comprise the contents of a corresponding one of the plurality of data items, wherein said first given function comprises the particular function used to determine the first content-dependent name for said particular sequence of bits; and,

(c) selectively to allow a copy of the particular sequence of bits to be provided to or accessed by or from at least one of the computers in a network of computers, wherein a copy of the sequence of bits is not to be provided or accessed without authorization, as determined, at least in part, based on whether or not said first content-dependent name of the particular sequence of bits corresponds to one of the plurality of identifiers.

151. The system of claim **150** wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization in (c) is based, at least in part, on at least one policy associated with the particular zone.

152. The system of claim **151**, wherein at least some of said one or more zones are regions.

153. The system of claim **151** wherein the particular zone is determined using at least the first content-dependent name for the particular sequence of bits.

154. A system comprising hardware including at least one processor, and software, in combination with said hardware:

(a) to determine a first content-dependent digital identifier for a particular sequence of bits, said first content-dependent digital identifier being determined using at least a first given function of some data, wherein the data used by the first given function to determine the first content-dependent digital identifier for the particular sequence of bits comprises at least some of the bits in the sequence of bits, wherein two identical sequences of bits will have the same digital identifier as determined by the first given function; and

(b) selectively to allow the particular sequence of bits to be provided to or accessed by or from at least one computer of a network of computers, wherein a copy of the sequence of bits is not allowed to be provided or accessed without authorization, as resolved, based, at least in part, on whether or not said first content-dependent digital identifier for the particular sequence of bits corresponds to an entry in a first database, said first database comprising a first plurality of identifiers, each of said identifiers in said first plurality of identifiers being based, at least in part, on a function of the data that comprise the contents of a corresponding one of a plurality of data items, and wherein the first given function is the same as the function that was used to determine the first plurality of identifiers in the first database.

155. The system of claim **154** wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization in (b) is based, at least in part, on at least one policy associated with the particular zone.

156. The system of claim 155, wherein at least some of said one or more zones are regions.

157. The system of claim 155 wherein the particular zone is determined using at least the first content-dependent digital identifier for the particular sequence of bits.

158. A system comprising hardware including at least one processor, and software, in combination with said hardware:

(A) selectively to permit distribution of a set of data items in a network of servers by, for at least one particular data item in said set of data items:

(a1) said particular data item comprising a particular sequence of bits representing at least one of: a file, a portion of a file, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and an identifier of a data item,

the particular data item having at least one contextual name specifying at least one location of the particular data item;

to determine another name for the particular data item, the other name for the particular data item being distinct from the at least one contextual name of the particular data item, the other name for the particular data item comprising a content-dependent digital identifier for the particular data item which was determined at least in part using a given function of some data, said given function comprising a message digest function or a hash function, and where the data used by the given function to determine the content-dependent digital identifier for the particular data item comprises at least some of the bits in the particular sequence of bits comprising the particular data item, wherein two identical data items will have the same digital identifier as determined by the given function;

(a2) to ascertain whether or not the content-dependent digital identifier for the particular data item corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each identifier of said plurality of identifiers being based, at least in part, on a second given function of the data that comprise the contents of a corresponding one of the plurality of data items, wherein two identical data items will have the same identifier as determined using said second given function, and wherein the second given function is the same as the given function used to determine the content-dependent digital identifier for the particular data item; and,

(a3) selectively to permit of a copy of the particular data item to be provided or accessed from at least one of the servers of the network of servers, based at least in part on the content-dependent digital identifier of the particular data item, and wherein a copy of the particular data item is not allowed to be provided or accessed without authorization, as resolved based, at least in part, on whether or not the content-dependent digital identifier of the particular data item corresponds to one of the plurality of identifiers.

159. The system of claim 158 wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the authorization in (a3) is based, at least in part, on at least one policy associated with the particular zone.

160. The system of claim 159, wherein at least some of said one or more zones are regions.

161. The system of claim 159 wherein the particular zone is determined using at least the content-dependent digital identifier for the particular data item.

162. A system comprising hardware, including at least a processor, and software, in combination with said hardware:

(A) to obtain a content-based name for a particular sequence of bits, said content-based name comprising at least a value having been determined at least in part by applying a first given function comprising a message digest function or hash function to at least a portion of the particular sequence of bits, wherein two identical sequences of bits will have the same content-based name as determined by said first given function,

said particular sequence of bits comprising data representing at least one of: a file, a portion of a file, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and an identifier of a data item; and

(B) selectively to permit distribution of at least one copy of the particular sequence of bits across a network of servers by:

(b1) ascertaining whether or not the content-based name of the particular sequence of bits corresponds to one of a plurality of identifiers, said plurality of identifiers corresponding to a plurality of data items, each of said identifiers being based, at least in part, on a second given function of data that comprise the contents of a corresponding one of the plurality of data items, wherein the second given function is the same as the first given function; and

(b2) based at least in part on said ascertaining in (bl), selectively permitting distribution of at least one copy of the particular sequence of bits in the network of computers, wherein a copy of the particular sequence of bits is not to be provided without authorization, as determined based, at least in part, on whether or not said content-based name of the particular sequence of bits corresponds to one of said plurality of identifiers.

163. The system of claim 162 wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular sequence of bits is associated with a particular zone, and wherein the authorization in (b2) is based, at least in part, on at least one policy associated with the particular zone.

164. The system of claim 163, wherein at least some of said one or more zones are regions.

165. The system of claim 163 wherein the particular zone is determined using at least the content-based name for the particular sequence of bits.

166. A system comprising hardware, including at least a processor, and software, in combination with said hardware to:

(A) for a particular data item in a set of data items, said particular data item comprising a corresponding particular sequence of bits:

(a1) determine one or more content-dependent digital identifiers for said particular data item, each said content-dependent digital identifier being based at least in part on a given function of at least some of the bits in the particular sequence of bits of the particular data item, wherein two identical data items will have the same digital identifiers as determined using said given function; and

(a2) selectively permit the particular data item to be made available for access and to be provided to or accessed by or from at least some of the computers in

a network of computers, wherein the data item is not to be made available for access or provided without authorization, as resolved based, at least in part, on whether or not at least one of said one or more content-dependent digital identifiers for said particular data item corresponds to an entry in one or more databases, each of said one or more databases comprising a plurality of identifiers, each of said identifiers in each said database corresponding to at least one data item of a plurality of data items, and each of said identifiers in each said database being based, at least in part, on at least some of the data in a corresponding data item.

167. The system of any one of claims **146** to **166** wherein the hardware comprises one or more computers.

168. The system of claim **166** wherein the system comprises one or more zones, and wherein at least one of said one or more zones is used for access control, and wherein the particular data item is associated with a particular zone, and wherein the authorization in (a2) is based, at least in part, on at least one policy associated with the particular zone.

169. The system of claim **168**, wherein at least some of said one or more zones are regions.

170. The system of claim **168** wherein the particular zone is determined using at least one of the one or more content-dependent digital identifiers for the particular data item.

171. The system as in any one of claims **143**, **147**, **151**, **155**, **159**, **163**, and **168**, wherein the system comprises an object addressable storage system.

172. A computer-implemented method, operable in a computer system comprising at least one storage system, wherein

the at least one storage system comprises a plurality of zones, and wherein

each data item stored on the at least one storage system is associated with an object name, wherein

the object name for a data item is used to map to one of the zones on the at least one storage system on which the corresponding data item is to be stored, and wherein

each particular data item comprises a corresponding arbitrary sequence of bits, and wherein

the object name for each particular data item is based, at least in part, on a given function of at least some of the corresponding arbitrary sequence of bits of the particular data item, wherein

two identical data items will have the same name, as determined using the given function, wherein the given function comprises at least one hash function, and wherein

the method is implemented at least in part by hardware in combination with software, the method comprising:

(A) obtaining a specific object name for a first data item, wherein the first data item comprises a first arbitrary sequence of bits, the specific object name being based, at least in part, on the given function of at least some of the bits of the first data item; and

(B) hardware in combination with software, using said specific object name for said first data item, to ascertain a particular zone of said plurality of zones corresponding to said first data item; and

(C) selectively allowing access to or of the first data item, said allowing being determined based, at least in part, on at least one policy associated with the particular zone corresponding to said first data item.

173. The method of claim **172** wherein the given function comprises a cryptographic hash function.

174. The method of claim **173** wherein the at least one hash function is selected from the functions: MD4, MD5, and SHA.

175. The method of claim **172** wherein said access comprises one or more of:

(a) copying the first data item; (b) providing the first data item to at least one of a plurality of computers; (c) reading the first data item; (d) distribution of the first data item; (e) modification of the first data item; (f) storage of the first data item; (g) opening of the first data item; (h) publishing the first data item; (i) writing the first data item; (k) moving the first data item; and (l) deleting the first data item.

176. The method of claim **172** wherein the specific object name was obtained in (A) as part of a request to access the first data item.

177. The method of claim **172** wherein the particular sequence of bits represent data selected from the group comprising: a file, a portion of a file, a page in memory, a digital message, a portion of a digital message, a digital image, a portion of a digital image, a video signal, a portion of a video signal, an audio signal, a portion of an audio signal, a software product, a portion of a software product, and a identifier of a data item.

178. The method of claim **172** wherein at least some of said plurality of zones are regions.

* * * * *

US008099420C1

(12) **EX PARTE REEXAMINATION CERTIFICATE** (10860th)

United States Patent
Farber et al.

(10) **Number: US 8,099,420 C1**
(45) **Certificate Issued: *May 10, 2016**

(54) **ACCESSING DATA IN A DATA PROCESSING SYSTEM**

(75) Inventors: **David A. Farber**, Ojai, CA (US); **Ronald D. Lachman**, Northbrook, IL (US)

(73) Assignee: **PERSONALWEB TECHNOLOGIES, LLC**, Tyler, TX (US)

Reexamination Request:
No. 90/020,091, Sep. 23, 2015

Reexamination Certificate for:
Patent No.: **8,099,420**
Issued: **Jan. 17, 2012**
Appl. No.: **11/017,650**
Filed: **Dec. 22, 2004**

(*) Notice: This patent is subject to a terminal disclaimer.

Related U.S. Application Data

(60) Continuation of application No. 09/987,723, filed on Nov. 15, 2001, now Pat. No. 6,928,442, which is a continuation of application No. 09/283,160, filed on Apr. 1, 1999, now Pat. No. 6,415,280, which is a division of application No. 08/960,079, filed on Oct. 24, 1997, now Pat. No. 5,978,791, which is a continuation of application No. 08/425,160, filed on Apr. 11, 1995, now abandoned.

(51) **Int. Cl.**
G06F 17/30 (2006.01)

(52) **U.S. Cl.**
CPC ***G06F 17/3015*** (2013.01); ***G06F 17/3012*** (2013.01); ***G06F 17/30091*** (2013.01); *Y10S 707/914* (2013.01); *Y10S 707/915* (2013.01); *Y10S 707/916* (2013.01); *Y10S 707/922* (2013.01); *Y10S 707/959* (2013.01); *Y10S 707/961* (2013.01); *Y10S 707/966* (2013.01); *Y10S 707/99931* (2013.01); *Y10S 707/99932* (2013.01); *Y10S 707/99933* (2013.01); *Y10S 707/99942* (2013.01)

(58) **Field of Classification Search**
None
See application file for complete search history.

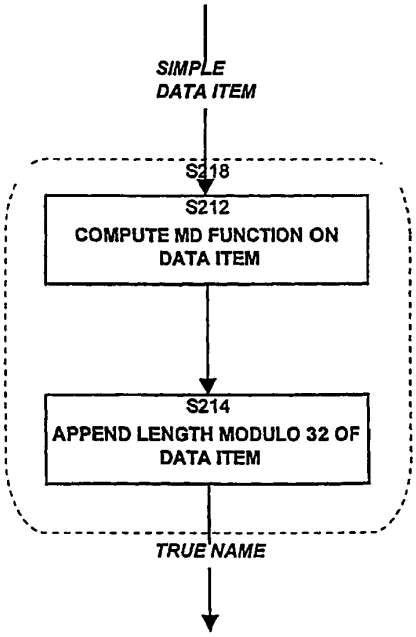
(56) **References Cited**

To view the complete listing of prior art documents cited during the proceeding for Reexamination Control Number 90/020,091, please refer to the USPTO's public Patent Application Information Retrieval (PAIR) system under the Display References tab.

Primary Examiner — Jeffrey Carlson

(57) **ABSTRACT**

Access to data items uses names based on the data in the data items; the name of a data item may be based, at least in part, on a function of some or all of the bits that comprise the data item. A data item may comprise an arbitrary sequence of bits. The function may include a hash function or a message digest function. The name of a data item may be compared to a list of names of other data items.



EX PARTE REEXAMINATION CERTIFICATE

NO AMENDMENTS HAVE BEEN MADE TO THE PATENT

AS A RESULT OF REEXAMINATION, IT HAS BEEN DETERMINED THAT:

The patentability of claim **166** is confirmed.

Claims **1-165** and **167-178** were not reexamined.

* * * * *

FORM 19. Certificate of Compliance with Type-Volume Limitations **Form 19**
July 2020

UNITED STATES COURT OF APPEALS
FOR THE FEDERAL CIRCUIT

CERTIFICATE OF COMPLIANCE WITH TYPE-VOLUME LIMITATIONS

Case Number: 2021-1858

Short Case Caption: In re: PersonalWeb Technologies, LLC

Instructions: When computing a word, line, or page count, you may exclude any items listed as exempted under Fed. R. App. P. 5(c), Fed. R. App. P. 21(d), Fed. R. App. P. 27(d)(2), Fed. R. App. P. 32(f), or Fed. Cir. R. 32(b)(2).

The foregoing filing complies with the relevant type-volume limitation of the Federal Rules of Appellate Procedure and Federal Circuit Rules because it meets one of the following:

☑ the filing has been prepared using a proportionally-spaced typeface and includes 13,799 words.

☐ the filing has been prepared using a monospaced typeface and includes _____ lines of text.

☐ the filing contains _____ pages / _____ words / _____ lines of text, which does not exceed the maximum authorized by this court's order (ECF No. _____).

Date: 08/27/2021

Signature: /s/ Michael A. Sherman

Name: Michael A. Sherman

CERTIFICATE OF SERVICE

I hereby certify that a true and correct copy of this **PERSONALWEB TECHNOLOGIES, LLC'S PRINCIPAL APPEAL BRIEF** was served on this 27th day of August, 2021 by operation of the Court's CM/ECF system on all counsel of record.

August 27, 2021

STUBBS, ALDERTON & MARKILES, LLP

/s/ *Michael A. Sherman*
Michael A. Sherman