



US 20050033775A1

(19) **United States**

(12) **Patent Application Publication**
Chen et al.

(10) **Pub. No.: US 2005/0033775 A1**

(43) **Pub. Date: Feb. 10, 2005**

(54) **FILE DELETION AND TRUNCATION USING
A ZOMBIE FILE SPACE**

Related U.S. Application Data

(63) Continuation of application No. 09/642,066, filed on
Aug. 18, 2000, now Pat. No. 6,751,635.

(75) Inventors: **Ray Chen**, Campbell, CA (US); **John
K. Edwards**, Sunnyvale, CA (US);
Kayuri Patel, Cupertino, CA (US)

Publication Classification

(51) **Int. Cl.⁷ G06F 7/00**

(52) **U.S. Cl. 707/200**

Correspondence Address:

SWERNOFSKY LAW GROUP PC

P.O. BOX 390013

MOUNTAIN VIEW, CA 94039-0013 (US)

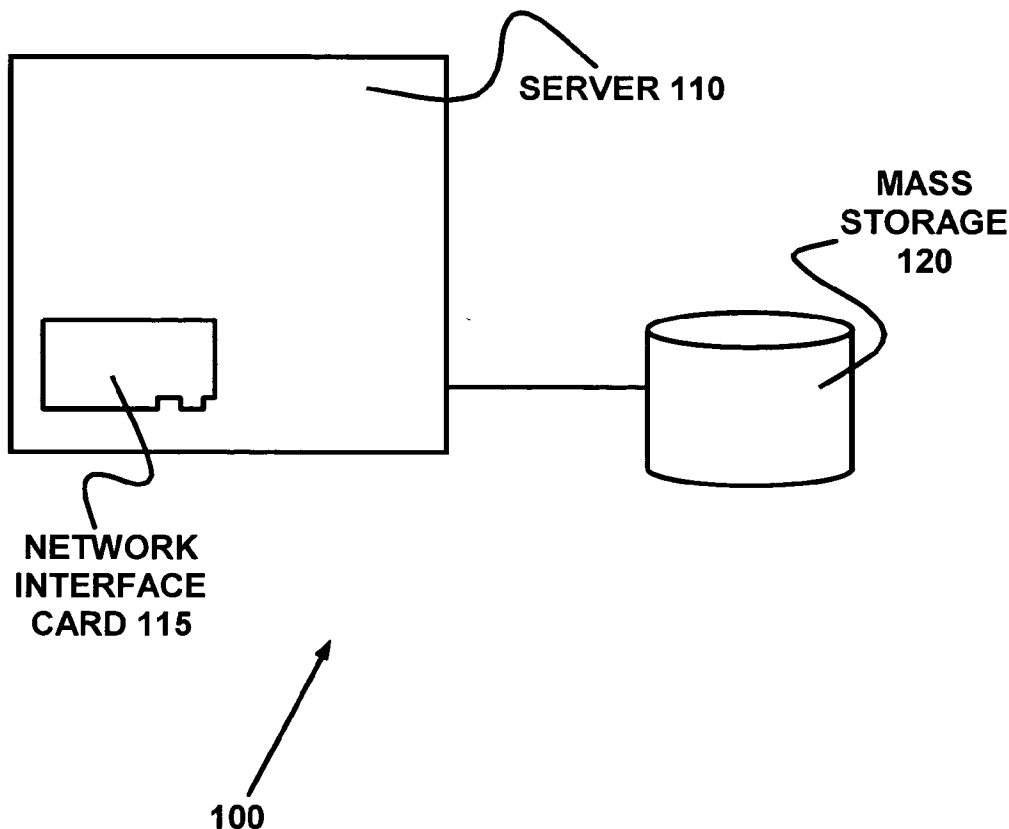
(57) **ABSTRACT**

A method and system for reliably performing extra-long operations in a reliable state-full system (such as a file system). The file system includes a separate portion of the file system reserved for files having extra-long operations in progress, including file deletion and file truncation. This separate portion of the file system is called the zombie file space; it includes a separate name space from the regular ("live") file system that is accessible to users, and is maintained as part of the file system when recording a consistency point.

(73) Assignee: **Network Appliance, Inc., a California
corporation**, Sunnyvale, CA (US)

(21) Appl. No.: **10/868,189**

(22) Filed: **Jun. 15, 2004**



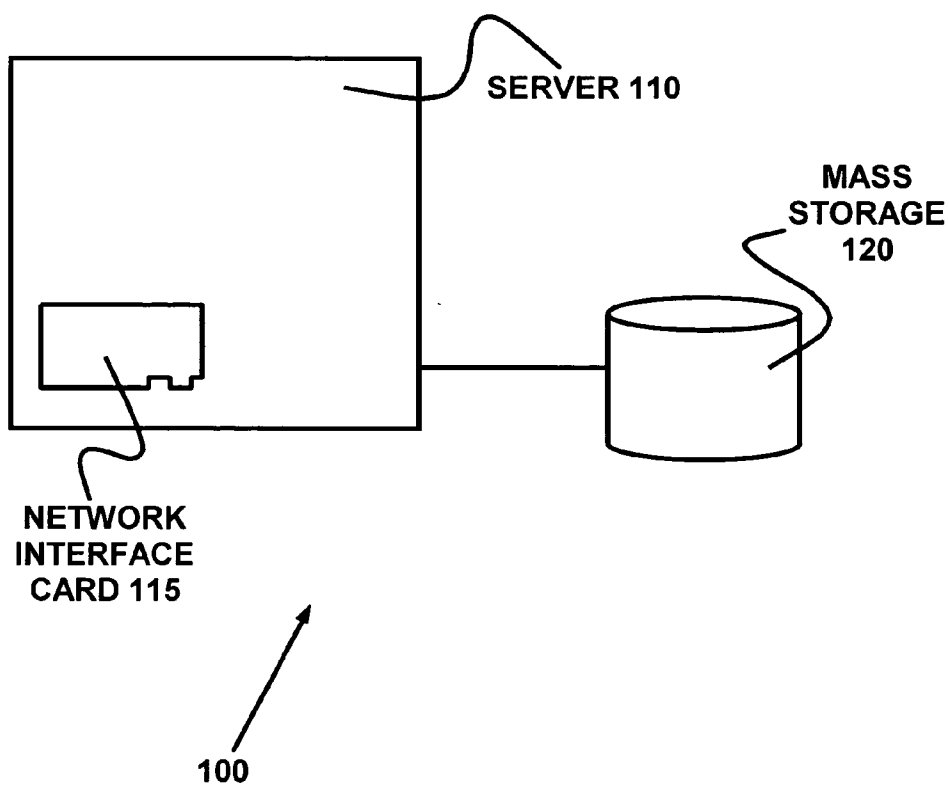


FIG. 1

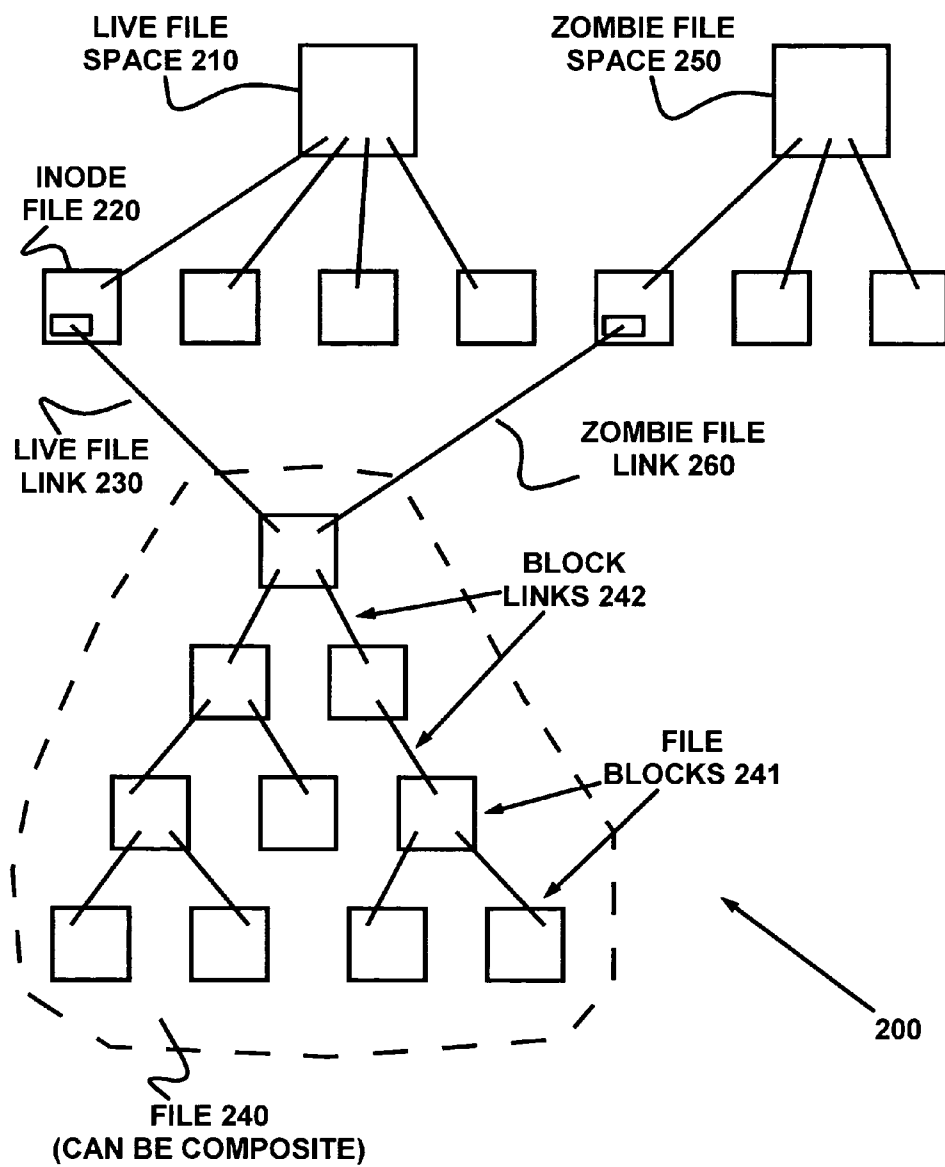


FIG. 2

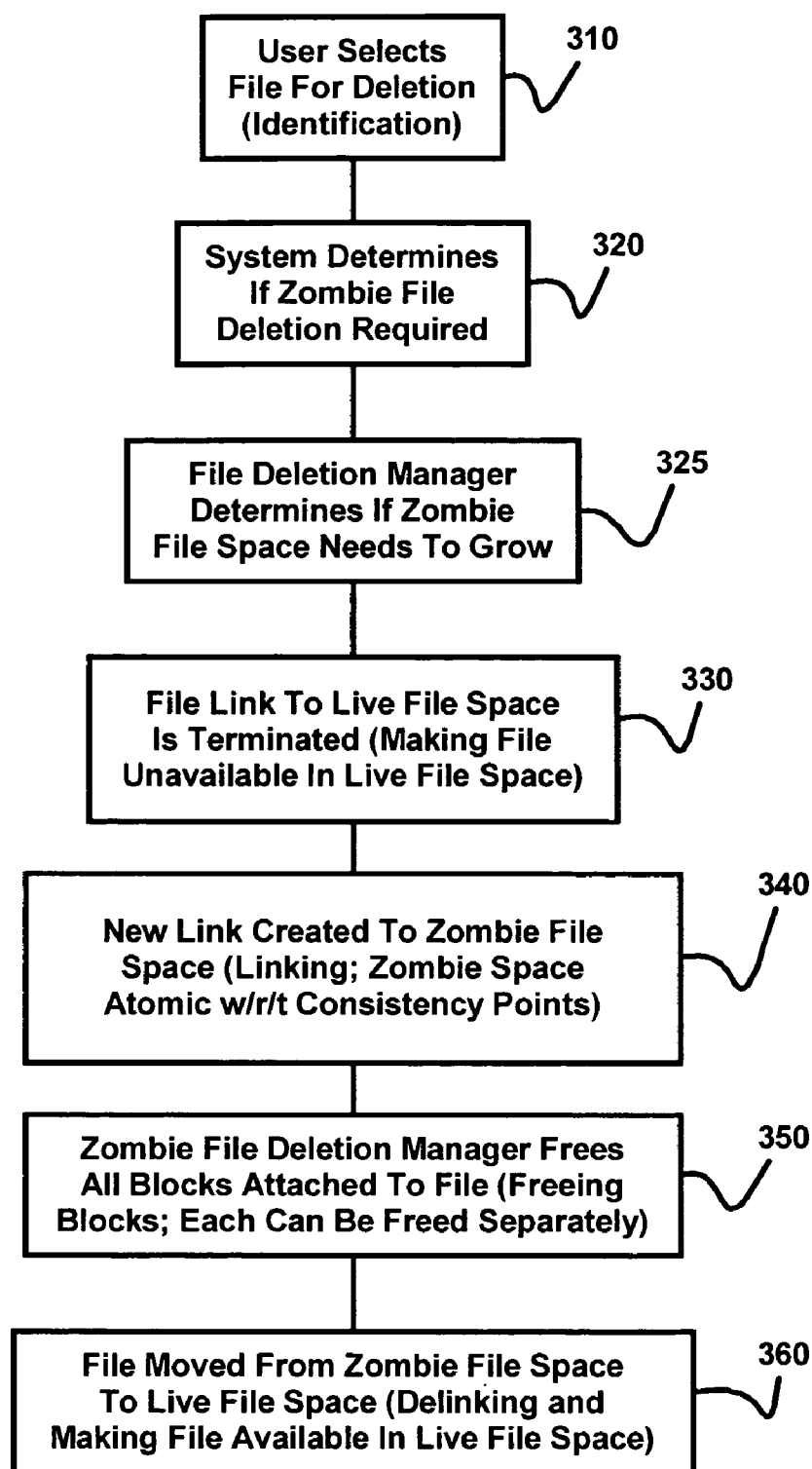


FIG. 3

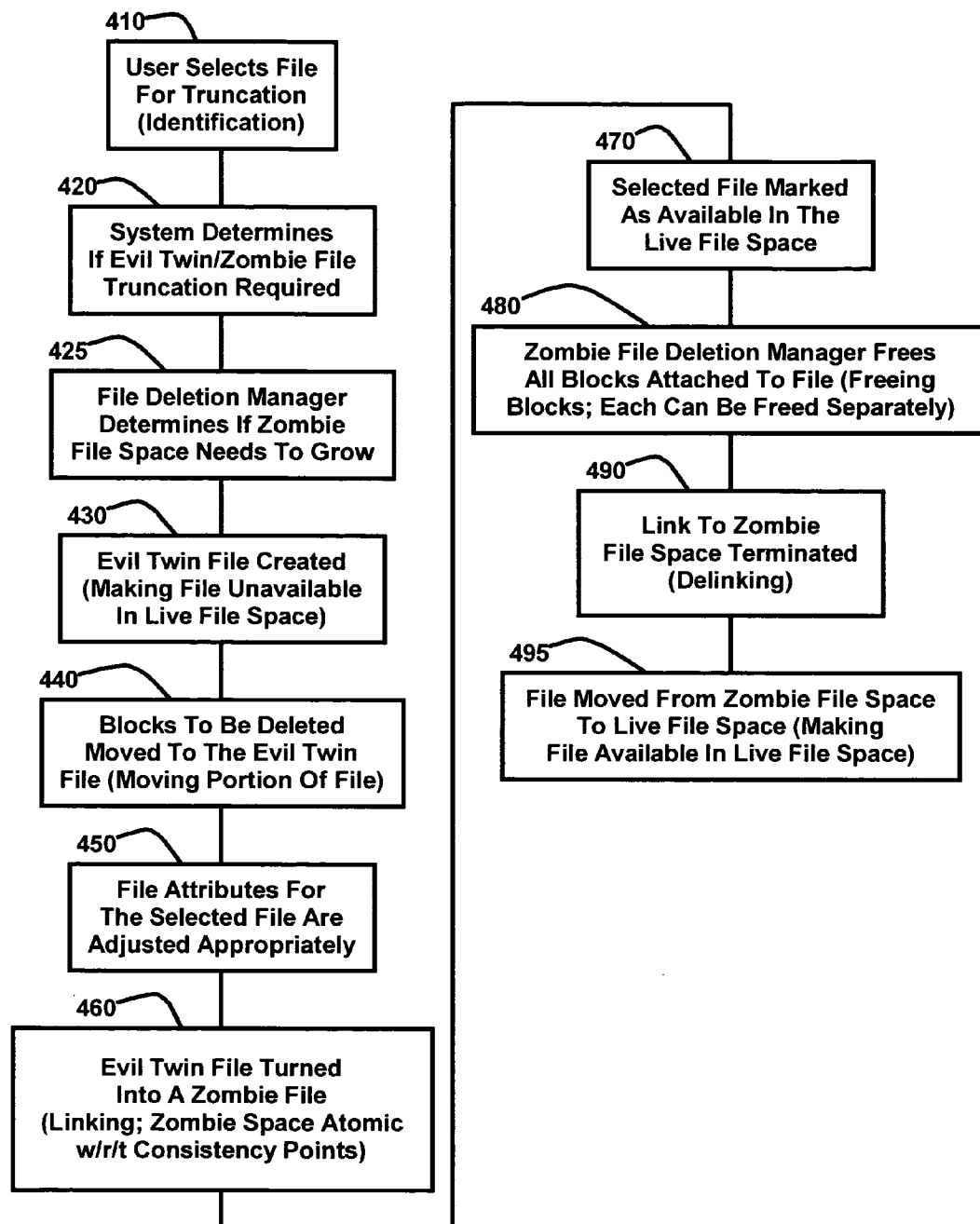


FIG. 4

FILE DELETION AND TRUNCATION USING A ZOMBIE FILE SPACE

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates to file server systems, including those file server systems in which it is desired to maintain reliable file system consistency.

[0003] 2. Related Art

[0004] In systems providing file services, such as those including file servers and similar devices, it is generally desirable for the server to provide a file system that is reliable despite the possibility of error. For example, it is desirable to provide a file system that is reliably in a consistent state, regardless of problems that might have occurred with the file server, and regardless of the nature of the file system operations requested by client devices.

[0005] One known method of providing reliability in systems that maintain state (including such state as the state of a file system or other set of data structures) is to provide for recording checkpoints at which the system is known to be in a consistent state. Such checkpoints, sometimes called "consistency points," each provide a state to which the system can retreat in the event that an error occurs. From the most recent consistency point, the system can reattempt each operation to reach a state it was in before the error.

[0006] One problem with this known method is that some operations can require substantial amounts of time in comparison with the time between consistency points. For example, in the WAFL file system (as further described in the Incorporated Disclosures), operations on very large files can require copying or modifying very large numbers of file blocks in memory or on disk, and can therefore take a substantial fraction of the time from one consistency point to another. In the WAFL file system, two such operations are deleting very large files and truncating very large files. Accordingly, it might occur that recording a consistency point cannot occur properly while one of these extra-long operations is in progress.

[0007] The fundamental requirement of a reliable file system is that the state of the file system recorded on non-volatile storage must reflect only completed file system operations. In the case of a file system like WAFL that issues checkpoints, every file system operation must be complete between two checkpoints. In the earliest versions of the WAFL file system there was no file deletion manager present, thus very large files created a problem as it was possible that such large files could not be deleted between the execution of two consistency checkpoints.

[0008] This problem was partially solved in later versions of the WAFL file system, where a file deletion manager was assigned to perform the operation of file deletion, and a consistency point manager was assigned to perform the operation of recording a consistency point. The file deletion manager would attempt to resolve the problem of extra-long file deletions by repeatedly requesting more time from the consistency point manager, thus "putting off" the consistency point manager until a last-possible moment. However, at that last-possible moment, the file deletion manager would be required to give way to the consistency point

manager, and allow the consistency point manager to record the consistency point. When this occurred, the file deletion manager would be unable to complete the file deletion operation. In that earlier version of the WAFL file system, instead of completing the file deletion operation, the file deletion manager would move the file to a "zombie file" list to complete the file deletion operation. At a later time, a zombie file manager would re-attempt the file deletion operation for those files on the zombie file list.

[0009] While this earlier method achieved the general result of performing file deletions on very large files, it has the drawbacks that it is a source of unreliability in the file system. First, the number of files that could be processed simultaneously as zombie files was fixed in the previous version.

[0010] Second, the file deletion manager and crash recovery mechanism did not communicate. The file deletion manager did not notify the crash recovery mechanism that a file was being turned into a zombie and the crash recovery mechanism was unable to create zombie files. Thus, to allow a checkpoint to be recorded, a long file would have to be turned into a zombie. If the system crashed at this point, the crash recovery mechanism might not be able to correctly recover the file system since it is unaware that a zombie file should be created and was incapable of creating zombie files should the need arise.

[0011] Third, since the file deletion manager and replay mechanism did not communicate the free space reported could be inaccurately reported. Attempts to restore state could fail, because the amount of free space could be different than that actually available.

[0012] Fourth, the earlier method is non-deterministic in the sense that it is not assured whether any particular file deletion operation will be completed before or after a selected consistency point. Moreover, the earlier method does not resolve problems associated with other extra-long file operations, such as requests to truncate very large files to much smaller length.

[0013] Accordingly, it would be advantageous to provide a technique for extra-long operations in a reliable state-full system (such as a file system), that is not subject to the drawbacks of the known art. Preferably, in such a technique, those parts of the system responsible for recording of consistency points are fully aware of the intermediate states of extra-long operations, the performance of extra-long operations is relatively deterministic, and performance of extra-long operations is atomic with regard to consistency points.

SUMMARY OF THE INVENTION

[0014] The invention provides a method and system for reliably performing extra-long operations in a reliable state-full system (such as a file system). The system records consistency points, or otherwise assures reliability, notwithstanding the continuous performance of extra-long operations and the existence of intermediate states for those extra-long operations. Moreover, performance of extra-long operations is both deterministic and atomic with regard to consistency points (or other reliability techniques used by the system).

[0015] The file system includes a separate portion of the file system reserved for files having extra-long operations in

progress, including file deletion and file truncation. This separate portion of the file system is called the zombie file space; it includes a separate name space from the regular ("live") file system that is accessible to users, and is maintained as part of the file system when recording a consistency point. The file system includes a file deletion manager that determines, before beginning any file deletion operation, whether it is necessary to first move the file being deleted to the zombie file space. The file system includes a zombie file deletion manager that performs portions of the file deletion operation on zombie files in atomic units.

[0016] The file system also includes a file truncation manager. Before beginning any file truncation operation, the file truncation manager determines whether it is necessary to create a complementary file called an "evil twin" file. The truncation manager will move all blocks to be truncated from the file being truncated to the evil twin file. Moving blocks is typically faster and less resource-intensive than deleting blocks. The "evil twin" is subsequently transformed into a zombie file. The file system includes a zombie file truncation manager that can then perform truncation of the zombie file asynchronously in atomic units. Furthermore, the number of files that can be linked to the zombie filesystem is dynamic allowing the zombie filesystem the ability to grow and shrink as required to process varying numbers of files.

[0017] An additional advantage provided by the file system is that files having attached data elements, called "composite" files, can be subject to file deletion and other extra-long operations in a natural and reliable manner. The file system moves the entire composite file to the zombie file space, deletes each attached data element individually, and thus resolves the composite file into a non-composite file. If the non-composite file is sufficiently small, the file deletion manager can delete the non-composite file without further need for the zombie file space. However, if the non-composite file is sufficiently large, the file deletion manager can delete the non-composite file using the zombie file space.

[0018] The invention provides an enabling technology for a wide variety of applications for reliable systems, so as to obtain substantial advantages and capabilities that are novel and non-obvious in view of the known art. Examples described below primarily relate to reliable file systems, but the invention is broadly applicable to many different types of systems in which reliability and extra-long operations are both present.

BRIEF DESCRIPTION OF THE DRAWINGS

[0019] **FIG. 1** shows a block diagram of a portion of a system using a zombie file space.

[0020] **FIG. 2** illustrates a file structure in a system using a zombie file space.

[0021] **FIG. 3** shows a process flow diagram for file deletion in a method for operating a system for manipulation of zombie files and evil-twin files.

[0022] **FIG. 4** shows a process flow diagram for file truncation in a method for operating a system Manipulation of Zombie Files and Evil-Twin Files.

[0023] LEXICOGRAPHY

[0024] The following terms refer to or relate to aspects of the invention as described below. The descriptions of general meanings of these terms are not intended to be limiting, only illustrative.

[0025] **live filesystem**—This term generally refers to a storage area within a file server where files are available to system users.

[0026] As noted above, these descriptions of general meanings of these terms are not intended to be limiting, only illustrative. Other and further applications of the invention, including extensions of these terms and concepts, would be clear to those of ordinary skill in the art after perusing this application. These other and further applications are part of the scope and spirit of the invention, and would be clear to those of ordinary skill in the art, without further invention or undue experimentation.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0027] In the following description, a preferred embodiment of the invention is described with regard to preferred process steps and data structures. Embodiments of the invention can be implemented using general-purpose processors or special purpose processors operating under program control, or other circuits, adapted to particular process steps and data structures described herein. Implementation of the process steps and data structures described herein would not require undue experimentation or further invention.

[0028] Related Applications

[0029] Inventions described herein can be used in conjunction with inventions described in the following documents.

[0030] U.S. patent application Ser. No. 09/642,062, Express Mail Mailing No. EL524780242US, filed Aug. 18, 2000, in the name of Rajesh Sundaram, et al attorney docket number 103.1034.01, titled "Dynamic Data Space"

[0031] U.S. patent application Ser. No. 09/642,061, Express Mail Mailing No. EL524780239US, filed Aug. 18, 2000, in the name of Blake Lewis et al attorney docket number 103.1035.01 titled "Instant Snapshot."

[0032] U.S. patent application Ser. No. 09/642,065, Express Mail Mailing No. EL524781092US, filed Aug. 18, 2000, in the name of Douglas Doucette, et al attorney docket number 103.1045.01, titled "Improved Space Allocation in a Write Anywhere File System." and

[0033] U.S. patent application Ser. No. 09/642,064, Express Mail Mailing No. EL524781075US, filed Aug. 18, 2000, in the name of Scott SCHOENTHAL, et al attorney docket number 103.1048.01, titled "persistent and reliable Delivery of Event Messages."

[0034] Each of these documents is hereby incorporated by reference as if fully set forth herein. This application claims priority of each of these documents. These documents are collectively referred to as the "Incorporated Disclosures."

[0035] System Elements

[0036] **FIG. 1** shows a block diagram of a portion of a system using a zombie file space.

[0037] A system **100** includes a file server **110** including a processor (not shown but understood by one skilled in the

art), program and data memory (not shown but understood by one skilled in the art), network interface card **115**, and a mass storage **120**

[0038] The network interface card **115** couples the file server **110** to a network. In a preferred embodiment, the network includes an Internet, intranet, extranet, virtual private network, enterprise network, or another form of communication network.

[0039] The mass storage **120** can include any device for storing relatively large amounts of information, such as magnetic disks or tapes, optical drives, or other types of mass storage.

[0040] File Structure Example

[0041] FIG. 2 illustrates a file structure in a system using a zombie file space.

[0042] A file structure **200** includes, a live file space **210**, an inode file **220**, a live file link **230**, a file **240**, a zombie file space **250**, and a zombie file link **260**.

[0043] The live file space **210** contains the root block and all associated blocks of data for live files (I.e. files that may be accessed by users).

[0044] The inode file **220** is associated with the file to be deleted and contains information about the file.

[0045] The live file link **230**, links a file to the live file space **210**.

[0046] The file **240** includes a plurality of file blocks **241**, and a plurality of block links **242**. The file blocks **241** are connected by the plurality of block links **242**. The file **240** is illustrative of a file to be deleted. The structure of the file as defined above is a hierarchical tree-like structure, however, there is no requirement in any embodiment of the invention that the invention be applied only to file structures of this type. The use of a hierarchical tree-like structure filing system is intended to be illustrative only and not limiting.

[0047] The zombie file space **250** contains the root block and all associated blocks of data for zombie files (I.e. files that are in the process of being deleted).

[0048] The zombie file link **260**, links a file to be deleted to the zombie file space **250**. A file that has been linked to the zombie file space **250** is referred to as a “zombie file” while it is so linked.

[0049] Method of Operation—File Deletion

[0050] A method **300** includes a set of flow points and a set of steps. The system **100** performs the method **300**. Although the method **300** is described serially, the steps of the method **300** can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method **300** be performed in the same order in which this description lists the steps, except where so indicated.

[0051] At a flow point **310**, a system user selects the file **240** for deletion. User interfaces for this activity vary from system to system but are well known in the art.

[0052] At a flow point **320**, the file **240** is identified by the system as a large file requiring zombie processing. In a preferred embodiment, the specific size of a file necessary to

trigger zombie processing is parameter-based, software-selectable, however, it can be any set of instructions supporting this functionality, such as instructions hard-coded on a computer chip.

[0053] At a flow point **325**, the file deletion manager determines whether the zombie file space **250** needs to be enlarged to accommodate another zombie file, and if necessary enlarges the zombie file space.

[0054] At a flow point **330**, the link connecting the file **240** to the live file space **210** is terminated. At this point the file **240** is no longer available to users connected to the file server **110**.

[0055] At a flow point **340**, the file **240** is linked to the zombie file space **250** via the zombie file link **260**. At this point, file **240** is referred to as a zombie file.

[0056] At a flow point **350**, the zombie file deletion manager starts deleting portions of the file **240** by terminating block links **242** at the outer leaves of the file tree. As file blocks **241** are deleted by the zombie deletion manager, they become available for storage of other data. This fact is reflected in the free space indicator of the mass storage **120**.

[0057] At a flow point **360**, the file **240** is moved from the zombie file space to the live file space.

[0058] Method of Operation—File Truncation

[0059] A method **400** includes a set of flow points and a set of steps. The system **100** performs the method **400**. Although the method **400** is described serially, the steps of the method **400** can be performed by separate elements in conjunction or in parallel, whether asynchronously, in a pipelined manner, or otherwise. There is no particular requirement that the method **400** be performed in the same order in which this description lists the steps, except where so indicated.

[0060] At a flow point **410**, a system user selects the file **240** for truncation. User interfaces for this activity vary from system to system but are well known in the art.

[0061] At a flow point **420**, the system identifies the amount of the file to be truncated as requiring evil twin/zombie processing. In the preferred embodiment, the specific amount of data to be truncated necessary to trigger evil twin/zombie processing is parameter-based software-selectable; however, it can be any set of instructions supporting this functionality, such as instructions hard-coded on a computer chip.

[0062] At a flow point **425**, the file deletion manager determines whether the zombie file space **250** needs to be enlarged to accommodate another zombie file, and if necessary enlarges the zombie file space.

[0063] At a flow point **430**, an evil twin file is created. At this point the file **240** is unavailable to the user.

[0064] At a flow point **440**, blocks of data to be truncated are moved from the file **240** to the evil twin file.

[0065] At a flow point **450**, file attributes for the file **240** are adjusted appropriately (E.g. The size of the file).

[0066] At a flow point **460**, the evil twin file is turned into a zombie file. It is connected to the zombie file space.

[0067] At a flow point 470, the file 240 is marked as available in the live file space. At this point the file 240 is available to all users.

[0068] At a flow point 480, the zombie deletion manager frees all blocks attached to the zombie file.

[0069] At a flow point 490, the zombie file has been deleted and the link to the zombie file space is terminated.

[0070] At a flow point 495, the file 240 is moved from the zombie filespace to the live filespace.

[0071] Generality of the Invention

[0072] The invention has general applicability to various fields of use, not necessarily related to the services described above. For example, these fields of use can include one or more of, or some combination of, the following:

[0073] The invention is applicable to all computer systems utilizing large files.

[0074] Other and further applications of the invention in its most general form, will be clear to those skilled in the art after perusal of this application, and are within the scope and spirit of the invention.

[0075] Although preferred embodiments are disclosed herein, many variations are possible which remain within the concept, scope, and spirit of the invention, and these variations would become clear to those skilled in the art after perusal of this application.

1. A method of operating a file server, comprising the steps of:

identifying a file in a first file space for truncation using zombie space inaccessible to one or more system users;
making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

truncating said file while maintaining file consistency between check-pointed intervals;

delinking said file from said zombie file space; and

making said file available to the one or more system users in said first file space.

2. The method of claim 1, wherein the step of identifying comprises:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

3. The method of claim 2, wherein the step of truncating is performed after the step of linking and before the step of delinking.

4. A method of operating a file server, comprising the steps of:

identifying a file in a first file space for deletion using zombie space inaccessible to one or more system users;

making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

deleting said file while maintaining file consistency between check-pointed intervals; and

delinking said file from said zombie file space.

5. The method of claim 4, wherein the step of identifying comprises:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

6. The method of claim 5, wherein the step of deleting is performed after the step of linking and before the step of delinking.

7. The method of claim 5, wherein said file is a composite file comprising one or more attached data elements.

8. The method of claim 7, wherein the step of deleting comprises:

deleting at least one of the one or more attached data elements after the step of linking and before the step of delinking; and

deleting said file after the step of delinking.

9. A file server, comprising:

a processor;

memory storing program code;

a network interface; and

mass storage;

wherein said processor executes the program code stored in said memory to control access to said mass storage, the program code comprising instructions that, when executed by the processor, cause the server to perform the steps of:

identifying a file in a first file space for truncation using zombie space inaccessible to one or more system users;

making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

truncating said file while maintaining file consistency between check-pointed intervals;

delinking said file from said zombie file space; and

making said file available to the one or more system users in said first file space.

10. The server of claim 9, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of identifying:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

11. The server of claim 10, wherein the server performs the step of truncating after the step of linking and before the step of delinking.

12. A file server, comprising:

a processor;

memory storing program code;

a network interface; and

mass storage;

wherein said processor executes the program code stored in said memory to control access to said mass storage, the program code comprising instructions that, when executed by the processor, cause the server to perform the steps of:

identifying a file in a first file space for deletion using zombie space inaccessible to one or more system users;

making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

deleting blocks of said file while maintaining file consistency between check-pointed intervals; and

delinking said file from said zombie file space.

13. The server of claim 12, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of identifying:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

14. The server of claim 13, wherein the server performs the step of deleting after the step of linking and before the step of delinking.

15. The method of claim 13, wherein said file is a composite file comprising one or more attached data elements.

16. The method of claim 15, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of deleting:

deleting at least one of the one or more attached data elements after the step of linking and before the step of delinking; and

deleting said file after the step of delinking.

17. A memory storing program code that, when performed by a processor of a file server, causes the server to perform the steps of:

identifying a file in a first file space for truncation using zombie space inaccessible to one or more system users;

making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

truncating said file while maintaining file consistency between check-pointed intervals;

delinking said file from said zombie file space; and

making said file available to the one or more system users in said first file space.

18. The memory of claim 17, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of identifying:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

19. The memory of claim 18, wherein the server performs the step of truncating after the step of linking and before the step of delinking.

20. A memory storing program code that, when performed by a processor of a file server, causes the server to perform the steps of:

identifying a file in a first file space for deletion using zombie space inaccessible to one or more system users;

making said file unavailable to the one or more system users in the first file space;

linking said file to the zombie file space;

deleting blocks of said file while maintaining file consistency between check-pointed intervals; and

delinking said file from said zombie file space.

21. The memory of claim 20, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of identifying:

receiving a selection of said file from a user; and

identifying said file for truncation using zombie space based on parameter-based file size comparison.

22. The memory of claim 21, wherein the server performs the step of deleting after the step of linking and before the step of delinking.

23. The memory of claim 21, wherein said file is a composite file comprising one or more attached data elements.

24. The memory of claim 23, wherein the instructions, when executed by the processor, cause the server to perform at least the following steps in the course of performing the step of deleting:

deleting at least one of the one or more attached data elements after the step of linking and before the step of delinking; and

deleting said file after the step of delinking.

* * * * *