PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 : G06F 12/00	A1	(11) International Publication Number: WO 94/14119
		(43) International Publication Date: 23 June 1994 (23.06.94)

(21) International Application Number: PCT/US93/11797

(22) International Filing Date: 6 December 1993 (06.12.93)

(30) Priority Data:
07/987,755 7 December 1992 (07.12.92) US

(71) Applicant: RAXCO, INCORPORATED [US/US]; 2440 Research Boulevard, Suite 200, Rockville, MD 20850 (US).

(72) Inventor: DAVY, William, R.; 5522 Loch More Court, Dublin, OH 43017 (US).

(74) Agents: GATTO, James, G. et al.; Baker & Botts, L.L.P., 555 13th Street, N.W., Suite 500 East, Washington, DC 20004 (US).

(81) Designated States: AU, CA, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.
Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.

(54) Title: APPARATUS AND METHOD FOR MOVING OPEN FILES

(57) Abstract

A method for moving open files on a computer system is disclosed. According to one aspect of the invention, an open file may be accessed by a user while being moved. To ensure accuracy, if data is to be written to an open file while it is being moved, the data is written to both the old and new locations.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	**GB**	United Kingdom	**MR**	Mauritania
AU	Australia	**GE**	Georgia	**MW**	Malawi
BB	Barbados	**GN**	Guinea	**NE**	Niger
BE	Belgium	**GR**	Greece	**NL**	Netherlands
BF	Burkina Faso	**HU**	Hungary	**NO**	Norway
BG	Bulgaria	**IE**	Ireland	**NZ**	New Zealand
BJ	Benin	**IT**	Italy	**PL**	Poland
BR	Brazil	**JP**	Japan	**PT**	Portugal
BY	Belarus	**KE**	Kenya	**RO**	Romania
CA	Canada	**KG**	Kyrgystan	**RU**	Russian Federation
CF	Central African Republic	**KP**	Democratic People's Republic of Korea	**SD**	Sudan
CG	Congo			**SE**	Sweden
CH	Switzerland	**KR**	Republic of Korea	**SI**	Slovenia
CI	Côte d'Ivoire	**KZ**	Kazakhstan	**SK**	Slovakia
CM	Cameroon	**LI**	Liechtenstein	**SN**	Senegal
CN	China	**LK**	Sri Lanka	**TD**	Chad
CS	Czechoslovakia	**LU**	Luxembourg	**TG**	Togo
CZ	Czech Republic	**LV**	Latvia	**TJ**	Tajikistan
DE	Germany	**MC**	Monaco	**TT**	Trinidad and Tobago
DK	Denmark	**MD**	Republic of Moldova	**UA**	Ukraine
ES	Spain	**MG**	Madagascar	**US**	United States of America
FI	Finland	**ML**	Mali	**UZ**	Uzbekistan
FR	France	**MN**	Mongolia	**VN**	Viet Nam
GA	Gabon				

APPARATUS AND METHOD FOR MOVING OPEN FILES

Field of the Invention

The invention relates to an apparatus and method for moving open files in a computer system.

Background of the Invention

It is well known in the computer field that for performance and other reasons, it is desirable to defragment (i.e., consolidate the segments of a file into one logically contiguous location on a disk) and/or optimize the position of files at a location on a disk other than their current location. Typically, defragmenting and positioning have been performed on files not currently in use.

Commercial defragmenters and disk optimizers (which both defragment and/or optimize file position on a disk) have been available for a number of years. Specifically, defragmenters and disk optimizers for use in the VAX/VMS marketplace are available. While the discussion herein is primarily directed to the VAX/VMS, application of this method to other systems will be readily apparent to one of ordinary skill in the art. However, none of these products can move files that are concurrently being read and written (i.e., "open" files). These commercial defragmenters and disk optimizers have a number of key features which are necessary to make them generally useful. Among these necessary and currently available functions are the following.

First, the software must run in a VAXcluster. VAXcluster is the name of the software environment created by DEC which allows multiple VAX systems to be linked together in such a way that any or all of the systems can share the disks on any or all of the other systems just as though those disks were attached to the local systems.

Second, the operation must be completely transparent to any and all user applications. That is, all user programs must run exactly the same and produce the exact same results, regardless of whether or not files are being defragmented or moved. Currently available software accomplishes this feat in part by not moving files that are currently being accessed by other users. If another user were to try to access the file

being moved, that user would either be stalled until the file move was completed or else the file move would be aborted, leaving the old version of the file for the user to access.

Third, the move file operation must be "atomic." That is, a file can never be left in an intermediate state. For example, it is possible that a system can crash at any time (for example, due to a power failure, hardware failure, etc.). Regardless of the nature of the failure, the file must be left either in its original state or else in its completely copied state.

The reference to "locks" herein is intended to refer to the standard Distributed Lock Manager locks described in the VAX/VMS documentation set. These are logical locks on arbitrary "resources" whose names can be up to 31 characters. The lock manager is a standard part of the VMS operating system and are maintained cluster-wide by VMS through standard VMS system calls. A working knowledge of the Distributed Lock Manager is assumed.

One prior software package is called Perfect Disk ("PD"), which operates as follows. When a process in the VMS file system tries to open, close, extend, or delete a file, the XQP (the file system processing code) takes out a "protected write" mode (PW) lock on the file that is called the "file serialization" lock. Its name is F11B$s + the file identification number. This lock will be referred to herein as the F11B$s lock or the file serialization lock. By taking out this lock, the system can check the status of the file (opened, closed, etc.) and be guaranteed that no other user will change the status while it is doing so. When the status check or state change is completed, the XQP gives up the lock so that other users may access the file.

When PD determines that it would like to move a particular file, it starts by taking out a "file serialization" lock in "protected read" (PR) mode with a "blocking AST" (the blocking AST causes a notification if another users tries to take out an incompatible lock). While it holds the F11B$s lock in PR mode, no other users in the cluster can change the state of its access. In particular, if no other user has the file open, then no other user can access the file while the lock is held.

After PD acquires the lock, it checks locally to determine if another user has the file open locally. This is done by searching the file control blocks (FCBs) maintained

in main memory by the XQP for all open files. If it is not open on the local node, then PD takes out a "file access arbitration" lock (referred herein as the F11B$a lock) in null (NL) mode. If a file is open on any node in a VAXcluster, then there exists such a lock on that node. PD can then do a $GETLKI (get lock information) system call and determine how many such locks exist in the cluster. If there is more than one (PD's lock), then another user has the file open and PD will not attempt to move the file. PD then drops the F11B$a lock since it has no further use for it at that time. Assuming the process is to continue, PD then allocates space on the disk at the target location for the defragmented/optimized version of the file. It reads the file data from the old location and writes it to the new location. A verification pass can be performed if desired to guarantee that the data was correctly copied. Up to this point, if the system crashes for some reason, the old file exists as always and there is no problem. The space allocated for the new version of the file will be deallocated when the disk bitmap is rebuilt, a normal operation at start-up.

As is well known, a file on a disk contains not only the data portion of the file, but also a file header containing "metadata." This file header contains data about the file including its name, size, creation, last backup, expiration, and modification dates, and mapping pointers that describe where the data portion of the file exists on the disk. The file header typically exists in block(s), and if it exists in more than one block, PD only moves the portion mapped by one file header blocking at a time. PD reads the old header, rewrites the file mapping pointers in memory, and then queues the rewrite of the header to disk. Either this rewrite succeeds or it fails. If it succeeds, then the file exists at its new location. If it fails, it exists at its old location. PD then deallocates the space where the old version of the file existed and drops the F11B$s lock so other users can then access the file. Note that any user that tried to access the file while PD was copying it was naturally put into a wait state by the lock manager (the process would be waiting to get its F11B$s lock in PW mode). When PD drops the F11B$s lock, the process may resume.

The foregoing method is useful for moving files (or segments) that are not open. However, various problems arise when trying to move open files. As a result, the above

scheme is inadequate to move "open files" (i.e., files that are being accessed for read or write by other users). While it has been previously recognized that it would be desirable to perform these functions while users are using the system and perhaps even the very file(s) to be defragmented or positioned, no solution to the various problems associated with such a capability has been provided. For example, in trying to move open files, one or more of the following problems may arise, among others.

A user that has the file open (anywhere in the cluster) has two data structures in memory that describe the state of the file and its location. The first is the file control blocking (FCB) mentioned before. It may have information that indicates the logical blocking number on the disk of the first blocking of the file (if the file is contiguous). It also has a "window control block" (WCB) that indicates where at least a portion of the file exists on the disk. If PD moves the file without causing these structures to be updated, then the reads and writes depending upon these structures will read and write where the file previously existed. This is undesirable.

For example, consider the case where a user is writing to the file while it is being copied. The writes must be coordinated with the copy of the file. For example, if a portion of the file has been read from the old location and written to the new location, then the writes must be made over the new portion of the file. If the write is to a portion of the file that has not yet been copied, then it must be made to the old portion of the file so that when PD copies over the new portion of the file, updated data will be written. If a user extends a file (that is, allocates more space to the file, and perhaps writes new data to it), PD must make sure that the new segment(s) of the file exists somehow in the new version of the file. If a user write to the file's new location should fail to properly write the data due to some I/O error (perhaps a bad spot on the surface of the disk) that would not have occurred writing to the file in its old location, then PD must be notified that the new copy of the file is bad so that it will not complete the copy operation. Various other concerns and problems also exist when trying to open files.

DEC and third-party developers have written products for the highly competitive defragmenter market since at least 1985 but none of these products has moved open files. Potential developers would be highly motivated to provide such a

capability because of the great marketing and technical advantages of being able to work on all of the files on a disk instead of just a portion of them. The failure of others to provide a workable solution evidences the long-felt but unfulfilled need to move open files.

Summary of the Invention

It is an object of the invention to overcome these and other drawbacks of the prior art. More specifically, it is one object of the invention to provide a method for moving open files.

In order to solve the foregoing and other problems, a cooperating "server" process is provided on each node in the VAXcluster. In this description, the file moving process will be called PD, and the server process will be called PD_SERVER.

Brief Description of the Drawings

Figure 1 is a flow chart illustrating a portion of the initial operation of PD and PD_SERVER.

Figure 2 and 2a show the flow of control for a portion of PD and PD_SERVER.

Figure 3 is a flow chart illustrating a portion of the synchronization of PD and a user file I/O.

Detailed Description of the Preferred Embodiments

Before attempting to copy a file, the PD_SERVER process is started on every node in the cluster on which the file to be copied is open. In this particular implementation, PD automatically tries to start the server on every node in the cluster before it tries to copy its first file so that it does not suffer the overhead at each file copy.

There are a number of ways to start processes on both local and remote nodes as will be readily apparent to one skilled in the art. In this implementation, PD copies the server process code to the disk being optimized and then uses "SYSMAN," a VMS utility, to start it from a spawned process.

As shown in Figure 1, for example, before starting the PD_SERVER processes, PD takes out a lock with the name PD_SERVER_diskname (where diskname is the name of the disk to be optimized) in PW mode (101). It holds this lock as long as it is

working on the disk. When it gives up the lock, it is a signal to the PD_SERVER processes that PD is no longer interested in the disk and that the PD_SERVER process should exit. If the node upon which PD is running should crash, then the other nodes are automatically notified because this lock will go away when the node does. Before starting the PD_SERVER processes, PD also takes out a lock with the name PD_FID_diskname in PW mode (102). This lock has several uses as discussed below. For example, this lock is used to signal to the PD_SERVER processes that a new file is being copied, which file is being worked upon, and completion status (success or failure) of the copy operation. The PD_SERVER can tell which disk it is being run from and therefore, it knows which disk it is serving. It also makes checks to see that it is the only copy of PD_SERVER which is serving that disk. If there is already another such process, it just exits. Once the PD_Server processes are started (103, 104), The PD_SERVER process queues a lock with the name PD_SERVER_diskname in PR mode (105). This lock is incompatible with the PD_SERVER_diskname lock held by the PD process. If the lock is ever granted, it is a signal that the PD process is exiting or that the PD process's node has crashed. If PD_SERVER is currently assisting with the copying of a file, it also serves as a signal that the copy will not be completed and the final update should not be made. The PD_SERVER process then takes out a lock with the name PD_SERVER_diskname in concurrent write (CW) mode (106). (This lock mode is compatible with other CW mode locks on the resource but not with PW locks. The utility of this will become apparent from the discussion below). The PD_SERVER process then queues for a PD_FID_diskname lock in PR mode (107). The PR mode is compatible with similar locks queued for by the PD_SERVER processes but not with the PW lock held by PD. PD_SERVER waits for the lock to be granted (108).

Meanwhile, PD identifies a file to be copied and starts a file copy operation at A (Figure 2) similar to the one described for moving closed files. It takes out the F11B$s lock on the file in EX mode to keep other users from opening, closing, extending, deleting, or truncating the file while it holds the lock. It checks for other users accessing the file anywhere in the cluster. If none are accessing it, then the

normal closed file copy procedure may be used. However, if the file is open anywhere on the system, PD checks that PD_SERVER is holding the PD_FID_diskname lock on every node that has the file open. It is not necessary the PD_SERVER be running on nodes that do not have the file open (202). If all such nodes have not successfully started and maintained the PD_SERVER processes, then the file will not be copied. PD solves the problem(s) of other users opening the file during the copy by not relinquishing the F11B$s lock during the copy. It also avoids the problem(s) of other users closing, extending, truncating, or deleting the file because none of these operations can occur while PD is holding the F11B$s lock.

PD then takes out a lock on PD_ERROR_diskname lock in PR mode with blocking AST (203). If a PD_SERVER process queues for a PD_ERROR_diskname lock in CW mode, PD will get such a blocking AST. This is a mechanism that is used for the PD_SERVER processes to notify PD that an error has occurred "remotely" which should cause PD to abort the copy operation without updating the header to point to the new data.

PD next determines which part of the file it will move (it doesn't necessarily have to move the entire file) to what location on the disk and allocates that space (204). When all of this setup is completed, PD notifies the PD_SERVER processes by writing the file ID (a unique identifier), the starting virtual blocking number (VBN) of the file to be copied, the number of blocks to be copied, and the starting logical blocking that the new portion of the file will occupy into the "value block" of the PD_FID_diskname lock and converts the PD_FID_diskname lock from the PW to PR mode (205).

When PD lowers the PD_FID_diskname lock to PR mode, any waiting requests for the PD_FID_diskname locks in PR mode in all of the PD_SERVER processes are granted (206). The PD_SERVER processes then read the lock's value blocking to learn the file ID, starting VBN, number of blocks to be copied, and target LBN. At this point, the PD_SERVER processes are ready to cooperate with PD in the file copy. However, the PD_SERVER process must stall the PD process from starting the file copy until certain housekeeping functions are taken care of.

It must be guaranteed that user processes will read and particularly write to the right location on the disk while PD is copying a file. While PD is copying a file and before it updates the header, all of the user processes can read the file from its original position. The old data is there the entire time, and furthermore, VMS naturally makes the processes read the right data under all conditions. So reads are not a particular problem while PD is copying the data to the new location.

Writes to the file are, however, a particular problem. It is not sufficient to write any given I/O to just one of the old or new locations of the file. While it might seem that some scheme would allow the data to be written to just one location or the other, depending upon PD's progress at copying the data, this is not adequate since, for example, PD may fail to complete the copy. (For example, its node might have a power failure, etc.) Therefore, all writes to the file during the PD copy phase are preferably written over the old version of the file. Furthermore, if PD has already copied that area of the file to its new location, the write must also occur at the new location. The solution therefore, is to make all write I/Os write both to locations. Methods for "shadowing" disk I/Os (i.e., making them write in two or more locations), in general, are known to those skilled in the art, but have not been used for this particular purpose. PD_SERVER can identify and shadow just exactly those I/O since it knows which blocks in which files must be shadowed.

However, there remain at least three other problems related to the file copy phase that must be recognized and addressed. First, when PD wants to start copying a file, PD_SERVER acquires the PD_FID_diskname lock as described above and is ready to go. It knows to shadow all future write I/Os to the file until the copy phase is done. However, this alone is inadequate. It is possible that one or more write I/Os to the file were queued before PD wanted to copy the file, but that for one reason or another, have not yet completed. (Though unlikely, in VAXclusters, it is possible that disk I/Os may take minutes or even hours to complete.) Therefore, PD must wait to start its copy until all outstanding write I/Os to the file have completed.

Another problem is that even though shadowing the write I/Os to both the old and new copies of the file is occurring, the system still must synchronize the write I/Os

with the PD copy process for at least the following reason. Suppose that during the copy phase, PD reads some blocking of data from the old location of the file, but before it can write it out to the new location, a user process writes that blocking of data. Even though the user write is shadowed to the new location, if it is written before PD can write its version of the data to the new location (and that is quite possible), user data will be lost in the new portion of the file. If the file copy is completed by PD, the user data will be lost. Therefore, writes to the disk should preferably be synchronized to prohibit this possibility.

Finally, when PD is done copying the file, it must point the file header mapping pointers to the new location of the file, rewrite the file header, and then deallocate the space where the file previously but no longer exists. This space can then be allocated to another file. If PD_SERVER was shadowing a write I/O to the file and the I/O was somehow delayed (a distinct possibility), when it completed later on, it might incorrectly write old data over another file. The following solutions overcome these potential problems.

In order for any user process on any node to be able to write to a file being copied, the PD_SERVER process must hold the PD_BLOCK_diskname lock in CW mode. In order for PD to do a read and then write of any data in the file for its copy operation, it must hold the PD_BLOCK_diskname in PW mode. While any PD_SERVER holds this lock in CW mode, PD cannot hold the lock in PW mode. So PD is preferably programmed so that it will not do the copy unless it holds the PD_BLOCK_diskname lock in PW mode and the PD_SERVER processes are preferably programmed to stall all write I/Os (303) until it holds the PD_BLOCK_diskname lock in CW mode. (Acquiring and releasing the locks is standard VMS lock work for those skilled in the art.) This solves the problem of users writing data during PD copy read-then-write copy operations. As shown in Fig. 3, for example, this may be implemented as follows.

To write to a file being copied, PD_SERVER determines whether it is currently holding PD_BLOCK_diskname lock in CW mode (301). If not, it holds the I/O and queues and waits for PD_BLOCK_diskname lock in CW mode (302). When this occurs,

it sends the I/O request to the driver to write to the old location (303). When complete, it requeues the driver to write to the new location (304). Upon completion of the second queuing (305), it determines whether there is a blocking AST on PD_BLOCK_diskname lock (306). If yes, it drops the PD_BLOCK_diskname lock and waits for a next event (307). If not, it is done and waits for the next I/O or PD done with file or blocking AST signal (308).

Meanwhile, PD determines whether it holds PD_BLOCK_diskname lock (309). If not, it queues and waits for PD_BLOCK_diskname lock in PW mode (310). If yes, it reads the next segment of data from the file (311), writes data to the new file location (312) and determines whether there is a blocking AST on PD_BLOCK_diskname lock (313). If yes, PD drops the PD_BLOCK_diskname lock (314) and control returns to 310. If not, control passes to 311.

To make sure that there are no latent I/Os at the start or finish which would cause other problems (for example, as described above), the PD_SERVER process stalls operations as follows. As described above, at the start of a file copy operation, PD_SERVER acquires the PD_FID_diskname lock which tells it which file is to be moved (206), but it is already holding the PD_BLOCK_diskname lock in CW mode (106) so that PD cannot actually start the data transfer. PD_SERVER then watches all of the new diskname I/Os that are queued to the diskname device from the local node. It passes all of the I/O requests onto the driver except for the I/Os to the file being copied (208), which it holds in a temporary queue. It then compares the number of requests in its temporary queue with the VMS device driver field (UCB$W_QLEN) which contains the number of outstanding I/Os on the device. When the two number are equal, then there are no previously queued I/Os to the file still outstanding. This allows the diskname to service all other I/Os and still stall the key ones as long as necessary. When all of the outstanding I/Os for the file have been "collected" in the temporary queue, synchronization has been accomplished and they are just requeued to the driver in the standard VMS way. If PD is just starting to process a new file, then when they are reprocessed by the driver, PD_SERVER will intercept them and shadow the writes properly. If PD is finishing a file, PD_SERVER will have done its cleanup

and will cause only the second half of the shadowing to occur. That is, the part in which the new location of the file is written.

Referring to the flow of a file copy, once PD_SERVER has performed the above described stall operations, it waits for the outstanding write I/Os to complete (208). PD_SERVER then drops the PD_BLOCK_diskname CW lock (209). When all of the servers have done so, PD's request for the PD_BLOCK_diskname PW lock will be granted (207) and PD can copy file data (211). PD will hold the lock through each individual sequence of reading and write each portion of the file. (A large file may require many reads and writes.) At the end of each individual read/write, PD will check to see if any PD_SERVER has requested the PD_BLOCK_diskname. If so, PD will drop the lock, which will allow other users to write the file as described, and then will requeue for it. When the other write(s) have completed, the PD_SERVER process(es) will drop the PD_BLOCK_diskname lock(s) and PD will reacquire the PD_BLOCK_diskname PW lock and then can continue its read/write operations.

From the PD_SERVER viewpoint, PD_SERVER is watching all of the diskname I/O requests. If it encounters a write I/O to the file being copied, rather than sending the request on to the driver, it stalls the request until the PD_SERVER_diskname lock is acquired in CW mode. When it gets the lock, it then sends the request to the driver to write to the original location. When the original request is completed, it then shadows the I/O to the new location. Only then does PD_SERVER drop the PD_BLOCK_diskname lock so that PD can start another read/write operation. If PD_SERVER detects an error in either of the writes, then PD_SERVER takes out a lock with the name PD_ERROR_diskname in CW mode (220). This is incompatible with PD's PR lock on the resource, so a blocking AST is generated in PD which informs PD that there is an error somewhere and that PD should not finish moving the file and control passes to 225.

Otherwise, PD finishes copying the data portion of the file. Up to this point, there is no problem if any of the systems in the cluster should crash. If the system running PD crashes, then its PD_SERVER_diskname PW lock is released, all of the remaining PD_SERVER process acquire their locks and know to exit without finishing

a file update. The original file and data structures are just as before the start of the copy. If any of the remote nodes running PD_SERVER crash, there is no problem because they will not be doing any I/O to the file and will not be able to do so until the copy is completely finished.

Then, PD (while holding the PD_BLOCK_diskname lock in PW mode) rewrites the file header pointing to the new location of the data (221). Once again, if any of the remote nodes crash, there is no problem because the new file will exist properly when the system is rebooted. If the node running PD crashes, the signalling scheme still works. The PD_SERVER processes will acquire the PD_SERVER_diskname lock and know that there was a failure. PD_SERVER will mark the file's FCB and WCBs stale so that the file system will reread the file header (PD may or may not have updated it.) Any pre-existing write I/Os will be shadowed to both the old and new locations. Since the PD process had not yet deallocated the old file space, it is safe to write to both places. New I/Os will write to the proper place.

After writing the header (221), PD drops the PD_BLOCK_diskname lock so that outstanding I/Os are allowed to proceed by PD_SERVER. PD queues to convert its PD_FID_diskname to PW mode if the copy was a success (222) (or to EX mode if the header was not updated (226)). Meanwhile, the PD_SERVER processes, which are holding the lock in PR mode, receive a blocking AST which informs them the PD is done copying the file (210). Then PD_SERVER checks the lock mode of the PD request to determine success or failure, but it matters little. In either case, PD_SERVER marks the FCB and WCBs "stale" (228) so that the file system must reread the file header so that new I/O requests will read/write the file in its new location. PD_SERVER does its "stall" operation so that outstanding I/Os (possibly shadowed) can complete. Upon completion of all outstanding I/Os, PD_SERVER drops the PD_FID_diskname lock to signify to PD that it has done all its cleanup (229). PD_SERVER requeues for the PD_FID_diskname lock (230) so that it will be ready for the next file to be copied by PD (231).

When all the PD_SERVERs have dropped their PD_FID_diskname locks (229), PD acquires the lock in PW mode (232) and knows that the PD_SERVER cleanup is

complete. At this point, PD deallocates the diskname space where the old file no longer resides, and finally, PD drops the F11B$s lock (233) so that other users may open, close, extend, truncate, or delete the file in the normal manner. This completes the file copy operation and PD finds the next file to process (234).

The foregoing is a description of the preferred embodiments of the present invention. However, various modifications within the scope of the invention will be readily apparent to those skilled in the art. The invention is only limited by the claims appended hereto.

CLAIMS

1. A method for moving an open file in a computer system, wherein said open file is currently being accessed by at least one user, said method comprising the steps of:

identifying an open file which is located at a first portion of a disk, where at least a portion of said file is to be moved to a second portion of said disk;

moving at least a portion of said open file to said second portion of said disk.

2. The method of claim 1 wherein said user may write to said open file while said open file is being moved and further comprising the step of:

writing data to said open file while at least a portion of said open file is in the process of being moved to said second portion of said disk, wherein said step of writing data includes the steps of writing said data to said first and second portions of said disk.

3. The method of claim 1 further comprising the step of acquiring and maintaining a lock on said open file while said open file is being moved to said second portion of said disk wherein said lock prevents said user from extending or truncating said open file while said lock is maintained on said open file.

4. The method of claim 2 wherein once said step of writing data to said first and second portions of said disk begins, no portion of the open file is copied until said data is written to both said first and second portions of said disk.

5. In a computer system comprising means for defragmenting files by moving at least a portion of said files and optimizing disks by moving file positions on said disk, a method for moving at least a portion of an open file while said file is being accessed by a user, said method comprising the steps of:

determining a portion of said file to be moved from a first location to a second location; and

copying said portion of said file to said second location while enabling said file to continue to be accessed by said user.

6. The method of claim 5 wherein when a user request to write data to said file during said step of copying is made, said data is written to said file by writing said data to both said first and second locations.

7. The method of claim 5 wherein accesses by a user comprises reading and writing data.

FIG. 1

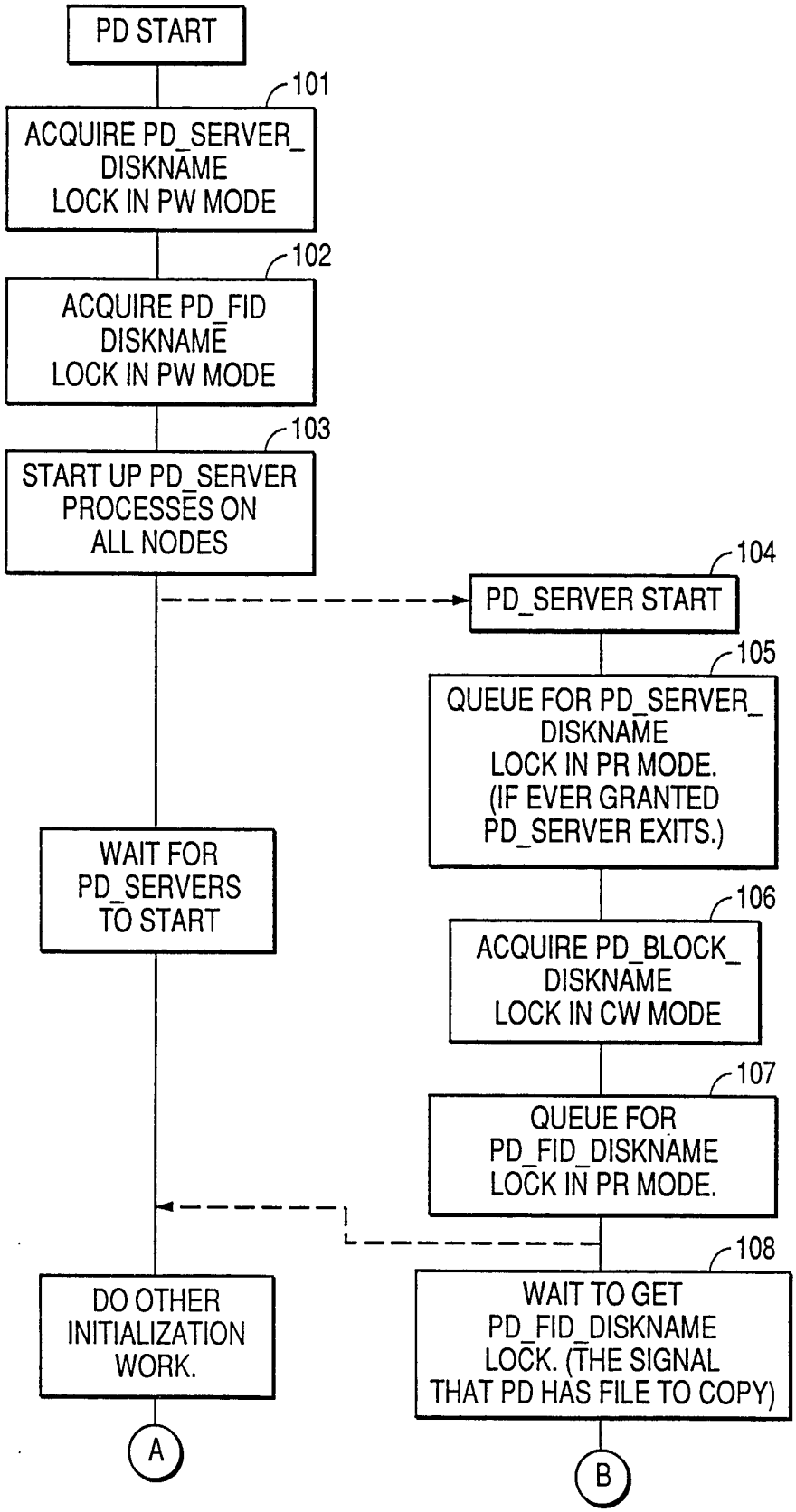
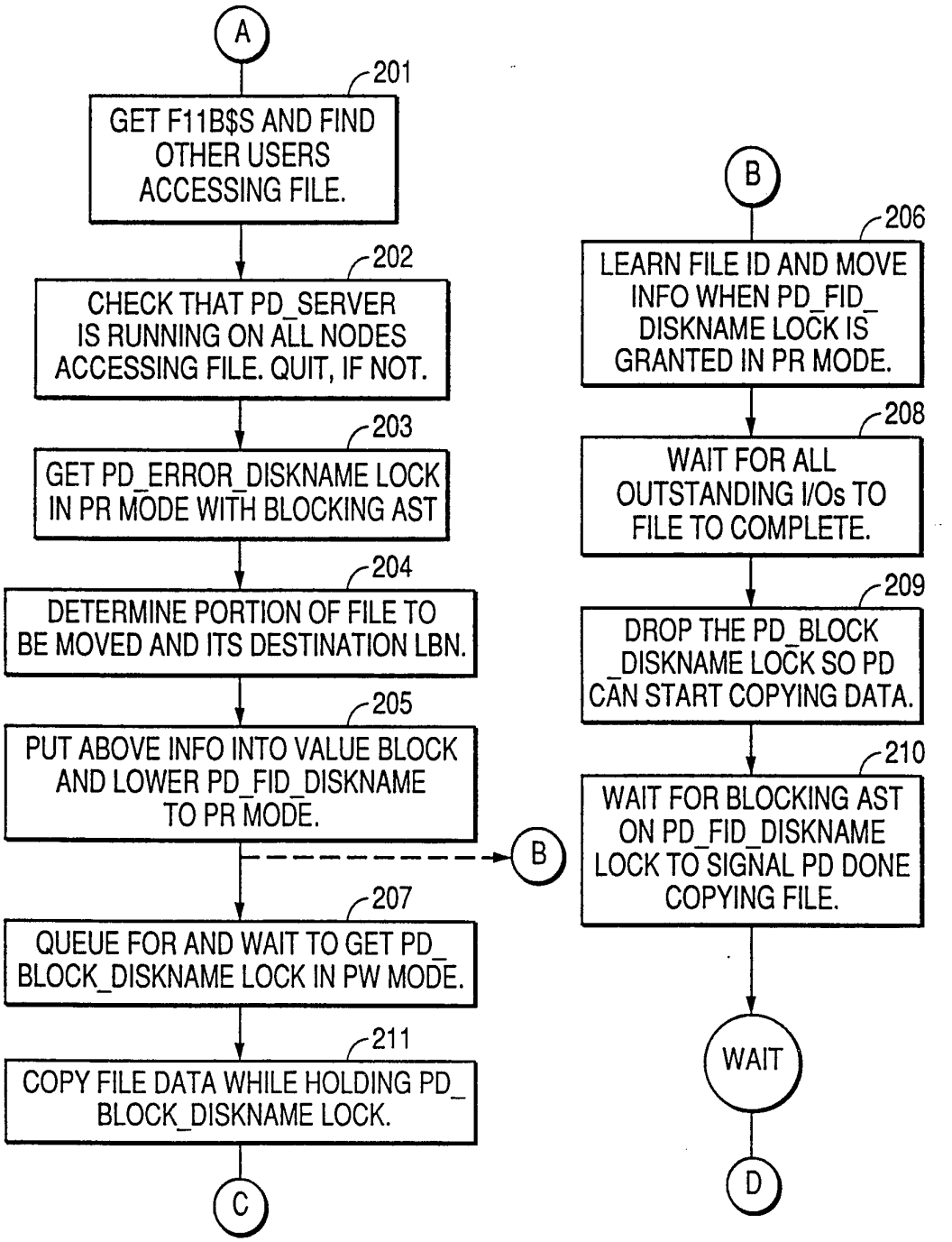

PD START
101
ACQUIRE PD_SERVER_ DISKNAME LOCK IN PW MODE
102
ACQUIRE PD_FID DISKNAME LOCK IN PW MODE
103
START UP PD_SERVER PROCESSES ON ALL NODES
104
PD_SERVER START
105
QUEUE FOR PD_SERVER_ DISKNAME LOCK IN PR MODE. (IF EVER GRANTED PD_SERVER EXITS.)
WAIT FOR PD_SERVERS TO START
106
ACQUIRE PD_BLOCK_ DISKNAME LOCK IN CW MODE
107
QUEUE FOR PD_FID_DISKNAME LOCK IN PR MODE.
108
WAIT TO GET PD_FID_DISKNAME LOCK. (THE SIGNAL THAT PD HAS FILE TO COPY)
DO OTHER INITIALIZATION WORK.
A
B

FIG. 2


A
201
GET F11B$S AND FIND OTHER USERS ACCESSING FILE.
202
CHECK THAT PD_SERVER IS RUNNING ON ALL NODES ACCESSING FILE. QUIT, IF NOT.
203
GET PD_ERROR_DISKNAME LOCK IN PR MODE WITH BLOCKING AST
204
DETERMINE PORTION OF FILE TO BE MOVED AND ITS DESTINATION LBN.
205
PUT ABOVE INFO INTO VALUE BLOCK AND LOWER PD_FID_DISKNAME TO PR MODE.
B
207
QUEUE FOR AND WAIT TO GET PD_BLOCK_DISKNAME LOCK IN PW MODE.
211
COPY FILE DATA WHILE HOLDING PD_BLOCK_DISKNAME LOCK.
C
B
206
LEARN FILE ID AND MOVE INFO WHEN PD_FID_DISKNAME LOCK IS GRANTED IN PR MODE.
208
WAIT FOR ALL OUTSTANDING I/Os TO FILE TO COMPLETE.
209
DROP THE PD_BLOCK_DISKNAME LOCK SO PD CAN START COPYING DATA.
210
WAIT FOR BLOCKING AST ON PD_FID_DISKNAME LOCK TO SIGNAL PD DONE COPYING FILE.
WAIT
D

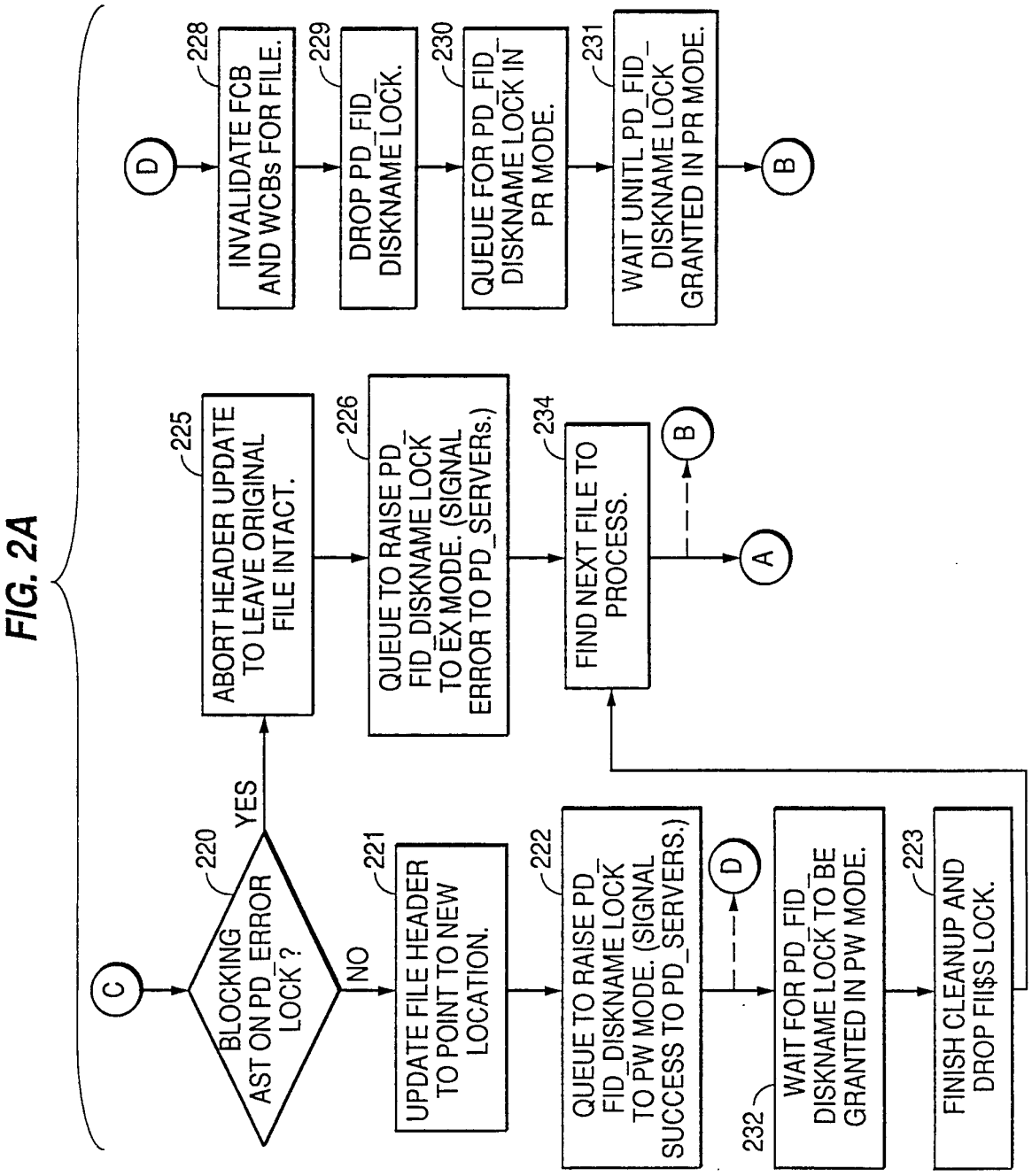

C
BLOCKING AST ON PD_ERROR LOCK ?
220
YES
NO
ABORT HEADER UPDATE TO LEAVE ORIGINAL FILE INTACT.
225
UPDATE FILE HEADER TO POINT TO NEW LOCATION.
221
QUEUE TO RAISE PD_FID_DISKNAME LOCK TO EX MODE. (SIGNAL ERROR TO PD_SERVERs.)
226
QUEUE TO RAISE PD_FID_DISKNAME LOCK TO PW MODE. (SIGNAL SUCCESS TO PD_SERVERS.)
222
FIND NEXT FILE TO PROCESS.
234
B
A
D
WAIT FOR PD_FID_DISKNAME LOCK TO BE GRANTED IN PW MODE.
232
FINISH CLEANUP AND DROP FI$S LOCK.
223
D
INVALIDATE FCB AND WCBs FOR FILE.
228
DROP PD_FID_DISKNAME LOCK.
229
QUEUE FOR PD_FID_DISKNAME LOCK IN PR MODE.
230
WAIT UNITL PD_FID_DISKNAME LOCK GRANTED IN PR MODE.
231
B

FIG. 2A

FIG. 3

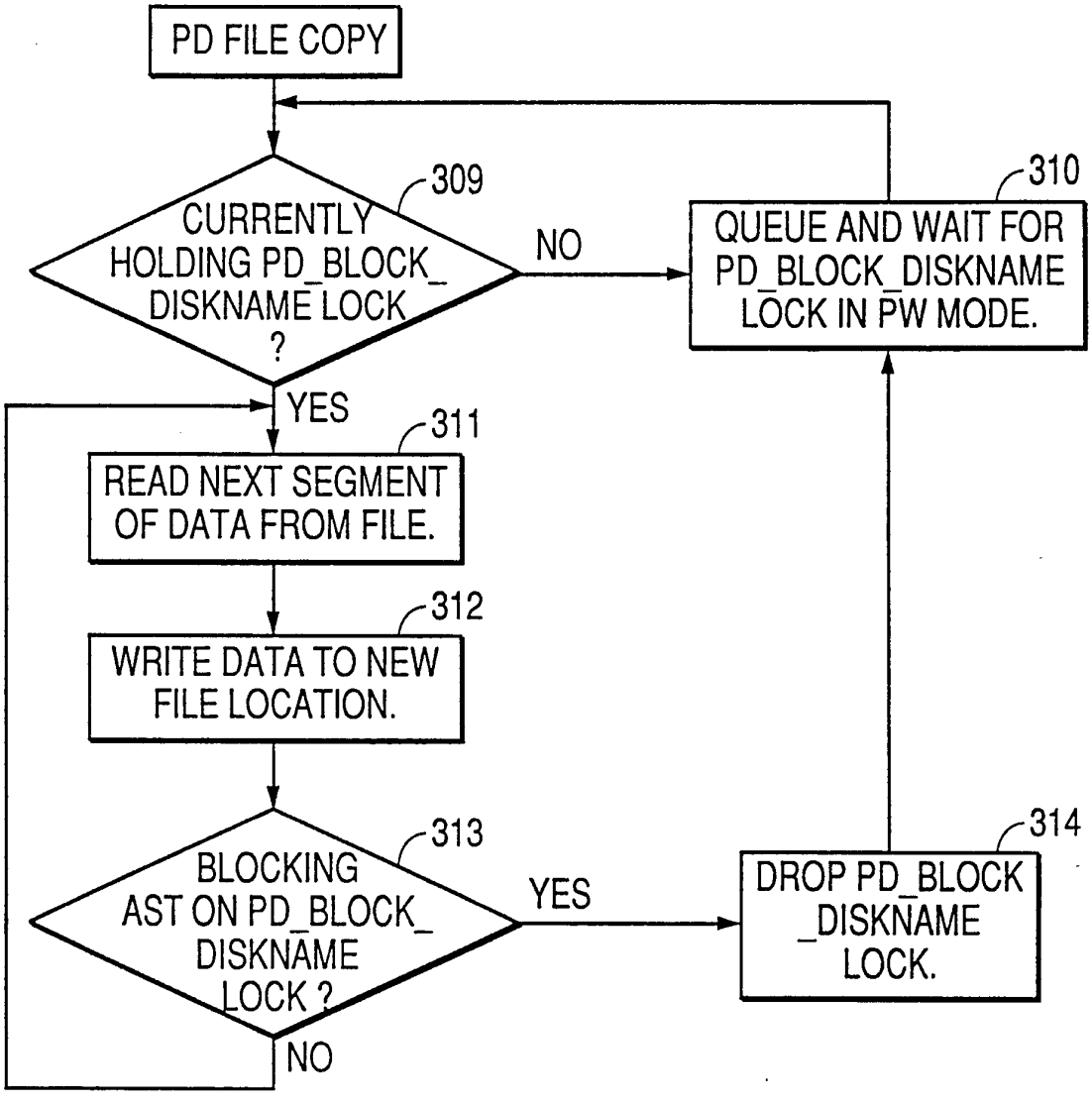
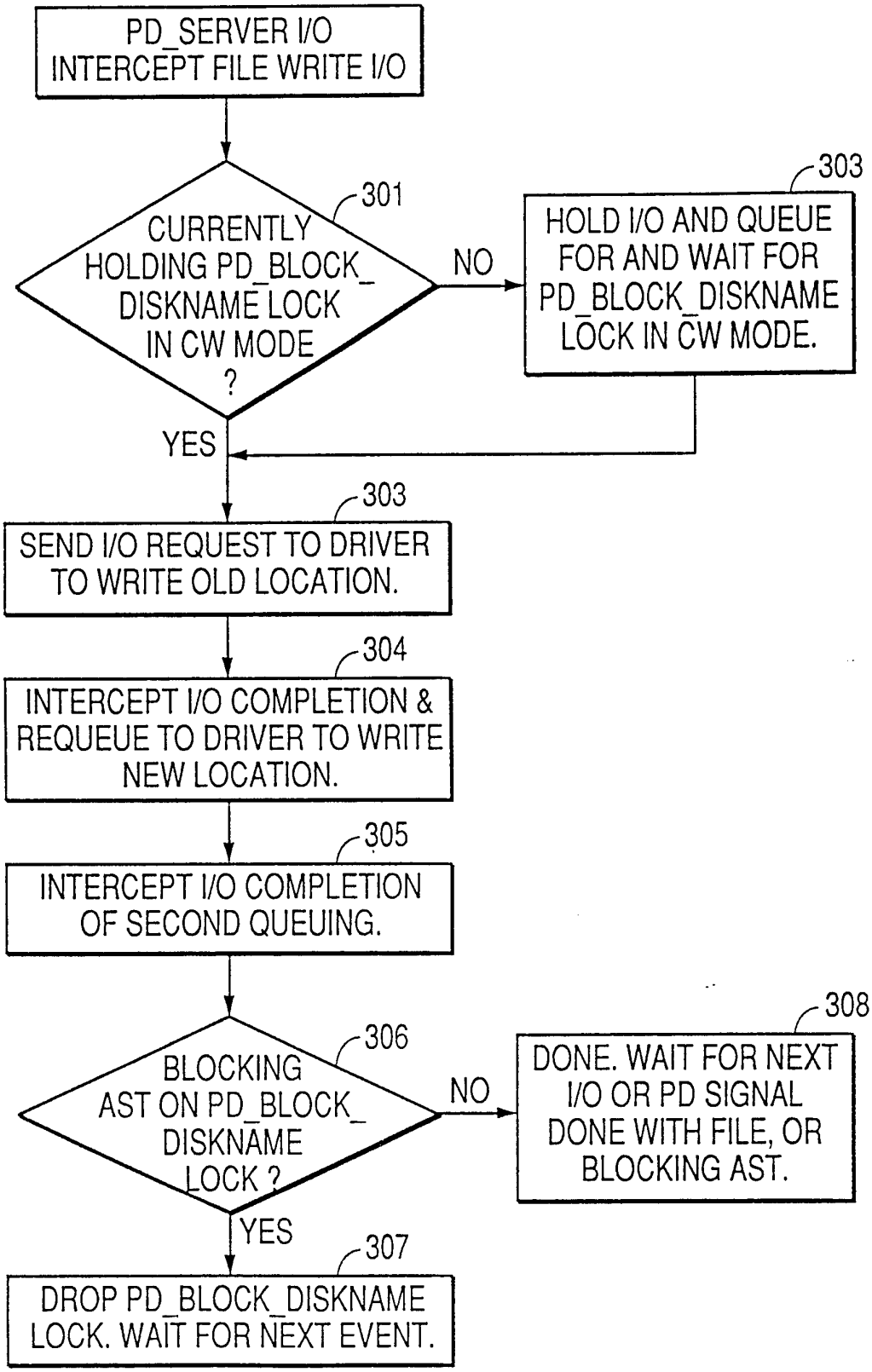

PD FILE COPY
309
CURRENTLY HOLDING PD_BLOCK_ DISKNAME LOCK ?
NO
310
QUEUE AND WAIT FOR PD_BLOCK_DISKNAME LOCK IN PW MODE.
YES
311
READ NEXT SEGMENT OF DATA FROM FILE.
312
WRITE DATA TO NEW FILE LOCATION.
313
BLOCKING AST ON PD_BLOCK_ DISKNAME LOCK ?
YES
314
DROP PD_BLOCK _DISKNAME LOCK.
NO

FIG. 3A


PD_SERVER I/O INTERCEPT FILE WRITE I/O
301
CURRENTLY HOLDING PD_BLOCK_ DISKNAME LOCK IN CW MODE ?
NO
303
HOLD I/O AND QUEUE FOR AND WAIT FOR PD_BLOCK_DISKNAME LOCK IN CW MODE.
YES
303
SEND I/O REQUEST TO DRIVER TO WRITE OLD LOCATION.
304
INTERCEPT I/O COMPLETION & REQUEUE TO DRIVER TO WRITE NEW LOCATION.
305
INTERCEPT I/O COMPLETION OF SECOND QUEUING.
306
BLOCKING AST ON PD_BLOCK_ DISKNAME LOCK ?
NO
308
DONE. WAIT FOR NEXT I/O OR PD SIGNAL DONE WITH FILE, OR BLOCKING AST.
YES
307
DROP PD_BLOCK_DISKNAME LOCK. WAIT FOR NEXT EVENT.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US93/11797

A. CLASSIFICATION OF SUBJECT MATTER

IPC(5) :G06F 12/00
US CL :395/600

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 395/600; 364/DIG.1, DIG.2

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A,E	US, A, 5,276,867 (Kenley et al) 04 January 1994, whole document.	1-7
A,P	US, A, 5,237,682 (Bendert et al) 17 August 1993, whole document.	1-7
Y,P	US, A, 5,212,786 (Sathi) 18 May 1993, abstract, Figures 16- 17, col. 1, line 27 - col. 2, line 4, col. 7, line 43 - col. 8, line 42.	1,5,7
Y,P	US, A, 5,175,852 (Johnson et al) 29 December 1992, whole document.	3

[X] Further documents are listed in the continuation of Box C. [ ] See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be part of particular relevance

"E" earlier document published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
16 February 1994	MAY 09 1994
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer B. Hardaway MARIA N. VON BUHR for Telephone No. (703) 305-9600

Form PCT/ISA/210 (second sheet)(July 1992)*

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US93/11797

C (Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X -- Y	US, A, 5,163,148 (Walls) 10 November 1992, abstract, Figures 4a-c, col. 2, lines 10-27, col. 2, line 62 - col. 3, line 22, col. 4, lines 43-68, col. 5, line 10 - col. 9, line 13.	1,2,4-7 ---------- 3
A	US, A, 5,021,946 (Korty) 04 June 1991, whole document.	1-7
Y	US, A, 5,008,853 (Bly et al) 16 April 1991, abstract, col. 30, line 31 - col. 32, line 14, col. 36, lines 5-64.	3

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US93/11797

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

APS - search terms: discs/disks, compaction, (de)fragmentation, read, write, update, editing, locking, moving, copying, open/active files, (dis)contiguous, optimizing