2204161

METHOD OF RAPIDLY OPENING DISK FILES IDENTIFIED BY PATH NAMES

Description

This invention relates to the field of data processing and, more particularly, to a method of rapidly opening disk files that are identified by path names.

The invention was designed as an improvement over a well known, commercially available system of the type in which hardware, such as an IBM Personal Computer, is operated under a disk operating system (DOS) and in which files are stored on fixed disks using tree structured directories and path names. Information is stored on a disk according to a predetermined pattern of cylinders and sectors, each sector containing a predetermined number of bytes. In order to access a desired sector, a head must first be moved to the cylinder containing the desired sector, the disk rotated past the head until the desired sector is reached and then the sector is read and the contents placed in a buffer. In looking at the total amount of time required to access data on a disk, the principal delay occurs during physical movement of a head. Where an application requires a large amount of physical I/O activity, it is desirable to reduce as much as possible the degree of head movement.

Files are stored on the disk in a cluster or clusters of sectors, each cluster having a predetermined number of sectors. Each cluster has a unique different starting address. The locations of files on a disk is kept track of by means of a file allocation table (FAT) which itself is stored on the disk. Each position of the FAT is associated with a different cluster and contains an entry indicating there are no other clusters associated with a file or pointing to the next cluster of the file. Small files are contained in a single cluster. Long files are contained in clusters that are chained together.

Files are located through the use of tree structured directories. Each disk contains a root directory, many sub-directories and a multiplicity of files. A given file may be at the end of a path passing through the root directory and several sub-directories. each director contains entries for additional directories and files. A specific file may be identified by specifying the drive, path and filename. For example, C:/DIR1/DIR2/FILE 1 identifies a filename FILE1 that is listed in directory DIR2, which is a sub-directory of DIR1 and listed therein, DIR1 being a sub-directory of the root directory, and listed therein, of drive C.

When a file is opened, it is necessary to access the drive and search through all of the directories specified in the path to locate the directory containing the entry of the filename. In such latter directory, all of the entries and filenames are searched until the desired one is located. If a file has not been previously opened, there will be no entry and therefore entries will have to be made before the file can be used. If the file has been previously opened, then the entry in the directory in which the filename is listed, contains an entry that is an index into the FAT that corresponds to the cluster where the file begins. During such open procedure, physical I/O activity must occur to access the root directory, each sub-directory and to search through a long list of filenames. In some applications, the same files are opened many times during the course of running a given program and because each opening involves a large amount of physical I/O activity, a relatively large amount of time can be lost. The invention is designed to improve upon the prior art method of opening by providing a method that rapidly opens files and reduces the amount of physical I/O activity associated with the opening process.

A method of operating a data processing system having a main memory, a cache in the main memory, and a disk in which is stored a tree structure of directories and files, each file being identified by a path containing a chain of directories including one directory having an

entry therein that contains information indicating where the file is stored on the disk, characterised in that the method comprises the steps of:

searching, in response to an initial request to open a first file, through the chain of directories stored on the disk to read therefrom the information indicating where the file is located on the disk;

storing in the cache a plurality of chained entries including one entry for each of the directories in the path to the first file, and one entry containing the information dictating where the file is stored on the disk,

searching, in response to a subsequent request to open the first file, through the chain entries to locate the information indicating where the file is stored on the disk,

searching, in response to a further request to open a second file identified by a path which includes one or more directories the same as one or more of the directories in the path of the first file, through the entries in the cache corresponding to the common directories and thereafter reading from the disk such further information as is necessary to form a complete path to the second file.

An embodiment of the invention will now be described, by way of example with reference to the accompanying drawings wherein;

Fig 1 is a schematic diagram generally illustrating the relationship of the invention to the environment or prior art in which the method is carried out,

Fig 2 is a flow chart of general steps that occur response to a user's request for a file;.

Fig 3 is a flow chart for the selection from the main procedures of various details procedures;

Fig 4 is a flow chart of the look-up procedure;

Fig 5 is a flow chart of the insert procedure;

Fig 6 is a flow chart of the delete request procedure;

Fig 7 is a flow chart of the delete procedure;

Fig 8 is a flow chart of the update procedure; and

Fig. 9 is a flow chart of how DOS can invoke certain procedures of the invention.

Referring now to the drawings, and first to Fig. 1, a data processing system 10, such as an IBM Personal Computer AT, includes a processor 12 connected over a bus 14 to a main memory 16 and disk controller 18. The controller in turn connected to a disk drive 20 which contains a disk 22 having a conventional storage media thereon into which bits of information are recorded. During operation of system 10, DOS procedures 24 are loaded or resident in main memory 16 and are primarily used to control access to information on disk 22. DOS is associated with system buffers 26 into which or through which information is transferred between disk 22 and main memory 16 in conventional fashion.

Also loaded into and resident in main memory 16 are a set of procedures named FASTOPEN 28 that are associated with operation of a FASTOPEN cache 30, the details of the procedures being described hereinafter. If there is more than one drive in the system, there will be a number of caches 30 one for each drive. The disk itself 22 is formatted so as to contain a file allocation table (FAT) 32, and root directory 34. Also illustrated in Fig. 1 is an exemplary file structure in which file 40 named FILE1 is located in sub-directory 38 named DIR2, which itself is a sub-directory of sub-directory 36 named DIR1 which is a sub-directory of root directory 34. Without FASTOPEN procedures 28 installed, the system operates in conventional well known fashion. FASTOPEN 28 is installed by initialising various variables described below and thereafter terminating but staying resident within main memory 16 to be available for use as needed.

As indicated previously, there is one cache 30 for each disk drive in the system. Each cache contains a block of information known as the drive cache header. Such header contains various fields whose meaning is set forth in Table 1.

Field	Meaning
1	Header of LRU chain of this drive
2	Offset to last entry of LRU chain
3	Pointer to first child in entry chain
4	Pointer to next drive cache header
5	Drive ID

TABLE 1 - DRIVE CACHE HEADER

Also located in each cache 30 will be a number of directory and file entries each containing the fields of information listed in Table 2. These entries form a chained data structure, that is more fully described below. The size of cache 30 will be determined by the desired number of entries which number can be chosen through a system default value or by the user selecting the number of entries. Normally the number selected by the user should be larger than the deepest nesting of path entries in the drive.

Field	Meaning
1	Pointer to next LRU entry
2	Pointer to next MRU entry
3	Pointer to child
4	Pointer to sibling
5	Pointer to preceding node
6	Directory/file information

TABLE 2 - ENTRY

As will be appreciated, the entries are maintained in a chain in such a manner that when cache 30 is full of entries, and a new entry needs to be added, the new entry will replace the least recently used (LRU) entry in the cache. Thus fields 1 and 2 are used to chain the entries in accordance with the LRU, MRU (most recently used) concept.

An entry is added for each file that has been opened and for each sub-directory included in the path to such file. Two files or directories or combinations thereof in the same sub-directory are considered to be at equal levels and are called "siblings". Any entry that is a sub-directory of another or a file is called a "child". For search purposes, the entries in the cache are arranged in a tree structure through the pointer fields 3-5. Field 6 contains that information that is associated with the normal directory entry contained on the disk media except that this information is inserted into field 6 of the name entry and therefore is resident in main memory. Such information can therefore be rapidly accessed when it is necessary to open a file for which there is a corresponding entry in cache 30. The information for a file is filename and extension, attributes, time of last write, date of last write, first cluster in file, and file size.

Referring now to Fig. 2, when the application or user program wants to open a file, the known DOS call interrupt 21Hex, for opening a file, is used. Such call goes to DOS 24 and FASTOPEN, as previously indicated, is installed so as to intercept such interrupt and act on it. The first decision of course is to determine whether or not FASTOPEN has been installed and this is done in step 52. If it has not, then the procedure advances in accordance with the prior art step 54 of accessing the disk to locate each sub-directory and file in the normal manner after which step 56 returns back to the user or application. If FASTOPEN is installed, then step 58 looks up the request in a manner to be described in more detail hereafter. If there was an error that may have occurred during the process, step 60 then branches to 54 and the process proceeds in accordance with the prior art process. If there was no error, step 62 then determines whether the path was partially found or conversely, if it was wholly found. If it was not partially found, i.e., the whole path and filename were found, then the same information as is provided by step 54 is then returned by step 56 back to the user. If the information was only partially found, then step 64, starting from the point of what information has been found, then proceeds to access the disk to locate the remaining portions of the path and filename.

Each portion is inserted in step 66 and step 68 causes the process to be repeated from step 64 until the end of the path or filename has been reached, at which time the appropriate information is returned to the user.

Referring to Fig. 3, the central part of the procedure is known as "main" and it is used to decide which of four sub-procedures are to be used. The sub-procedures being known as "look-up", "insert", "delete", and "update", respectively. These different procedures might for example be invoked by inserting a different numeral corresponding thereto in one of the processor registers. In any event, step 70 decides if it is a look-up operation, step 72, 74 and 76 similarly decide whether it is an insert, delete or update operation respectively. If none of these operations are invoked, then an error signal is returned. These various sub-procedures will now be described relative to their separate figures.

Referring now to Fig. 4, in accordance with the look-up procedures, step 78 first locates the cache 30 associated with the requested drive. Step 80 then sets the LRU chain from the pre-LRU stack. This stack is a logical stack used to keep the tree structured subdirectory while maintaining LRU chain. Next, step 82 forms the start of a loop in which the current node is a pointer and it is initially set to point to the drive header and thereafter is then set to point to the current node entry. Current-sibling is set to zero and the directory or filename is looked at. The end of path or filename is determined by encountering the ASCII value zero and step 84 then decides if it has been reached. If it has not, which would be the usual instance when a search first begins, then step 86, using the pointer to the child from the current entry, would then find a child entry from the current mode. Step 88 looks to see if there are anymore children. That is, is the entry the last child in the sequence. This fact would be indicated by a minus one in the child field. The desired name is then compared with the child filename. If such comparison as a result of step 90 is okay then, step 92 creates the pre-LRU stack, saves the current node address in the

stack and branches back to step 82. The process continues until the filename is found at which the yes or positive determination from step 84 branches to step 94 and the record or information that is found is returned to DOS. If step 90 results in a negative determination, then step 98 looks at the pointer to the sibling field and by a loop going back through step 96, 90 and 98 would continue until no more siblings are left at which time step 94 returns to DOS. When returning to DOS, FASTOPEN will set the pointer up to the directory or filename found in the path/filename string provided by DOS. DOS, in turn, will look at the pointer to determine whether the whole path has been found, or if only a partial path has been found or if it found nothing. When there are more siblings, step 96 sets the current node pointer to the current sibling.

Referring to Fig. 5, the purpose of the insert procedure is to insert directory or file information as an entry in the cache, based on the current node and current sibling information. Step 102 makes a name record entry from the input and step 104 gets the next available entry. Until the cache is filled, a new entry would simply be put into whatever location or entry is free. Once the cache is filled, then the least recently used entry based upon the LRU list, would then be the next available entry to be replaced by the newer one. Step 106 clears the pre-LRU stack to the extent anything is there. If the current sibling pointer is equal to zero, then the new entry is installed by step 110 as a child under the current node and step 112 records such node in the pre-LRU stack. If a current sibling in step 108 is not equal to zero, then step 114 installs the new entry as a sibling of the current node.

Referring now to Figs. 6 and 7, when an application or user requests that a particular file be deleted, a procedure not only deletes the file in the disk in usual fashion but also deletes any corresponding entries in the cache. Thus, in response to the user request in step 116, the process passes through DOS 24 and step 118 decides whether FASTOPEN has been installed. If not, step 122 accesses the disk and deletes the file after which step 124 returns to the user. If FASTOPEN

is installed, then the delete request 120 is made. In response to such request, step 126 looks up the cache to determine whether or not there is a corresponding entry. If no entry if found the procedure is simply exited. If the entry is found, it is removed from the tree structured LRU chain by step 132 and that particular entry is put on the top of the LRU chain as being available for use by a new entry. A file entry is first deleted and then any parent directories that do not contain any child or sibling pointers.

Referring now to Figs. 8 and 9, there are certain functions built into the FASTOPEN procedures that can be invoked directly by DOS for accomplishing certain functions ancilliary to functions that DOS provides. The DOS requests are ancilliary to the functions of closing a file, writing zero bytes in a file or making a node and steps 142, 144 and 146 respectively determine whether such functions are requested and then branch to the update steps of 134-140. The positive results from steps 136-140 respectively update a directory entry, update a cluster number associated with a file and delete an entry.

In the foregoing discussion, mention was made of the pre-LRU stack and this will now be explained relative to the LRU stack, both of which involves the use of the LRU and MRU pointers in the entries. As the process steps through a path, it goes from the root directory and through any sub-directories until it reaches the filename and a pre-LRU stack is made in accordance with the order in which the various nodes are accessed. Thus, in an example of a file accessed or specified by C:/DIR1/DIR2/FILE1, the order of entries is the root directory, DIR1, DIR2 and FILE1. This ordering creates a problem to the extent that the root directory would be the least recently used entry and should such entry be deleted for replacement, then access to the remaining entries would be cut off. This problem is avoided by using the pre-LRU stack and re-ordering the entries, once the path is set up, to create an LRU stack in which the filename would appear to be the least recently used entry and therefore on replacement, it would be the filename and not any of the parent directories that would be initially replaced.

CLAIMS



1. A method of operating a data processing system having a main memory, a cache in the main memory, and a disk in which is stored a tree structure of directories and files, each file being identified by a path containing a chain of directories including one directory having an entry therein that contains information indicating where the file is stored on the disk, characterised in that the method comprises the steps of:

searching, in response to an initial request to open a first file, through the chain of directories stored on the disk to read therefrom the information indicating where the file is located on the disk;

storing in the cache a plurality of chained entries including one entry for each of the directories in the path to the first file, and one entry containing the information indicating where the file is stored on the disk,

searching, in response to a subsequent request to open the first file, through the chained entries to locate the information indicating where the file is stored on the disk,

searching, in response to a further request to open a second file identified by a path which includes one or more directories the same as one or more of the directories in the path of the first file, through the entries in the cache corresponding to the common directories and thereafter reading from the disk such further information as is necessary to form a complete path to the second file.

* C L M S *

2. A method as claimed in claim 1, wherein the system comprises a plurality of disk drives and a plurality of caches corresponding to the disk drives, comprises the step of:

including in each cache a drive header identifying a different one the drives; and in which,

the searching in response to said subsequent request to open the first file includes the preliminary step of accessing at least one of the drive headers to locate the cache containing the chained entries.

3. A method as claimed in claim 1, wherein the cache is of a size for containing a predetermined number of entries, including the steps of:

in response to a plurality of requests for opening a plurality of different files, filling the cache with entries corresponding to the different files;

storing in the cache a history of each file indicating which files is least recently used; and

in response to a request to open an additional file, deleting entries corresponding to the least recently used file and adding entries corresponding to the additional file.

4. A method as claimed in any preceding claim including the step of storing pointers in the entries to create a chain of entries ordered from a most recently used file to a least recently used file.

5. A method as claimed in claim 4 comprising the step of:

creating one or more new entries from the further information and adding the new entries to the cache to form therein a complete path of entries for the second file.