UIUCDCS-R-72-506

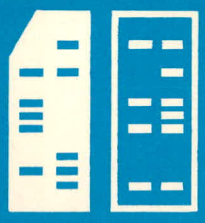
COO-1469-0200

A FAILURE TOLERANT FILING SYSTEM

by

Alfred D. Whaley

February 1972



DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN · URBANA, ILLINOIS

DISCLAIMER

UIUCDCS-R-72-506

A FAILURE TOLERANT FILING SYSTEM*

by

Alfred D. Whaley

February 1972

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF ILLINOIS
URBANA, ILLINOIS 61801

* Supported in part by the Atomic Energy Commission under grant US AEC AT(11-1)1469.

ABSTRACT

A random access tree structure filing system has been constructed that is immune to operating system or internal failures through constant maintenance of a correct data structure on disk. Emphasis is placed on the fail-safe nature of the filing system, the speed of access to random lines in a data file, and the low level of human intervention required over an extended period of time.

1. INTRODUCTION

The filing system described here was designed for storing large programs and computer output in a timesharing environment and was written for an IBM 360. The original objectives were

1. Quick random access to the lines in a file,
2. Efficient utilization of disk space,
3. No maintenance.

The last requirement is meant to apply even in real world situations such as system crashes. A system crash is any situation in which the CPU halts unexpectedly and operations must be restarted from the disk.

In addition, some features of the filing system were predetermined. A moving head disk was to be used (IBM 2314) with a small number of fixed length *blocks* permanently assigned to each track. As the blocks are made smaller, less space is wasted in partially filled blocks and more space is wasted in inter-record gaps and filing system information--pointers, etc. It was decided to use 8 blocks per track of 793 bytes (characters) each. Data files were to contain *lines*; they might be card images, printer lines, or other information. The user of the system supplies a *key* with each line--a unique code required for retrieving the line. The only other way for the user to retrieve a line is to use the commands supplied for reading the file sequentially. All lines in the file are constantly stored in order according to the collating sequence of their keys. The user of the system has no knowledge of the number of lines stored in any given block, the manner in which they are stored, or the method used by the system to locate them.

The use of the system was also partially predetermined. User files were to be separated on the basis of user name and user account number. This requirement seemed to suggest a tree structure having a catalog file of account numbers, each account number specifying a catalog of user names and each user name specifying a catalog of the user's data file names. The user could be permitted to carry on this catalog structure at his own level.

2. IMPLEMENTATION

An example tree is shown in Figure 1.

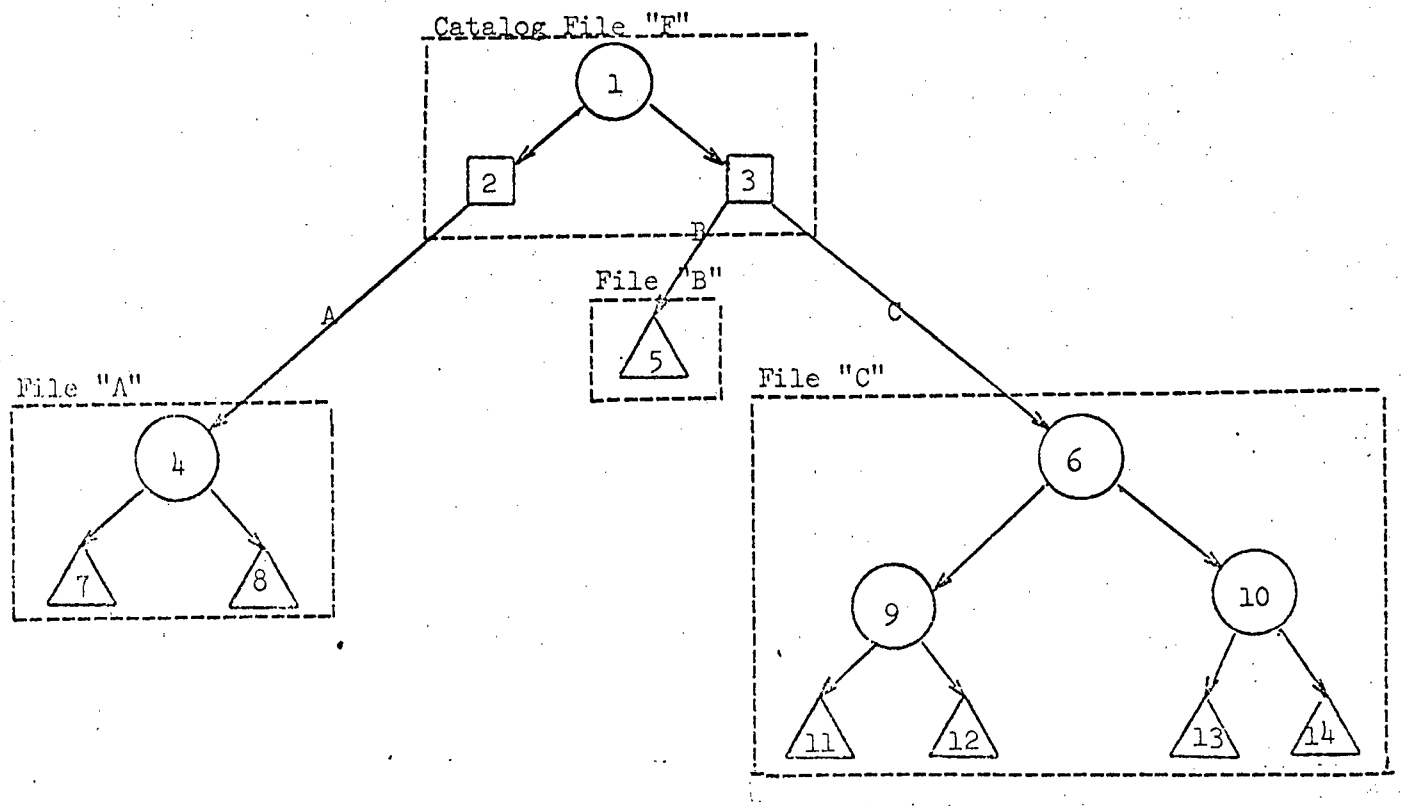


Figure 1. Typical Data Structure

Disk blocks are used for three different purposes. One kind is triangles which represent ordinary data blocks, each containing many lines (with keys) of the user's file. A data block contains 8 full card images or about 20 average cards. Circles represent directory blocks, used to keep track of a file when it is too large to be contained in one block. A directory block may contains pointers to as many as 50 other blocks. For example, blocks 4, 7 and 8 together contain one data file; blocks 6, 9, 10, 11, 12, 13 and 14 contain another; 5 still another. Squares represent catalog blocks which are the data blocks of a catalog file. Each block is self-identifying as to type (catalog, data or directory) by means of an internal code. In this example, blocks 1, 2

and 3 contain a catalog (named F) of three data files (named A, B and C). As there is more than one catalog block in the example (2 and 3), a directory block is inserted (1) which locates them. Typically, a catalog block contains catalog information for about 35 other files. Under file F are cataloged three data files. The file represented by 5 (file B) is small enough to be contained in one block and, therefore, requires no directory block. The file which starts at block 4 is somewhat larger, having two data blocks, so that it requires a directory block. The file starting at block 6 (file C) is larger still, having two levels of directories. (In practice, only one catalog block is needed to catalog files A, B and C shown in Figure 1; and only one directory block would be needed for the data blocks in file C.)

All records in both data and catalog files contain an 8 character (8 byte) key, sometimes referred to as a line number. In a catalog block a given record or line identifies a file underneath the catalog. The key in one of these records is the 8 character name of the file being identified. The remainder of the record contains the last date on which the file was accessed, a disk address of the top block of the file cataloged, and optionally, some additional information provided by the user. When a line from a catalog file is read, all but the disk address is supplied.

3. ACCESSING AND EDITING FILES

In the example, block 2 contains one record describing file A, and block 3 contains two records describing files B and C. Directory block 1 contains two records. The keys of these records have been employed to facilitate easy identification of the blocks immediately under a directory block. The keys are the same as those in the first record in the block immediately below the directory. Thus, block 1 has two records having keys A and B. The remainder of these records consists only of the disk address of the block just below.

As mentioned in our implementation of this data structure, a disk block on the 2314 was chosen to be 793 bytes (8 blocks per track), and a directory can specify over 50 blocks beneath it. Although these numbers make one realize that Figure 1 is somewhat deceptive (it appears that a directory can specify only two blocks), this figure is still potentially correct. In the data file at 4 a directory block is required, but only two entries are needed, representing the minimum configuration. The structure at 6 is hardly minimal, however, since all four data blocks could be easily specified by one directory block. One may conclude only that this file contained at one time more than the fifty odd data blocks that can be specified by one directory block, thus requiring an extra level of directory blocks to appear. After this structure was created, a large number of records were subsequently deleted. The kind of splitting that causes a second level of directory blocks to be added is analogous to what would happen if a data block such as 5 were exceeded and another (in this case the first) directory level were added, creating a structure similar to that at 4.

It should be noted that two directory levels can specify up to n^2 data or catalog blocks with n+1 directory blocks where n is the number of blocks one directory block can specify. Three or four levels are clearly adequate for most needs.

In returning to the first objective, one can see that each directory block record has the same key as the first record in the underlying block, making it easy to locate a data record by a vertical route from the top block of its file. Another equivalent data structure could have been constructed with each key the same as the last key in the underlying blocks. This technique would have been better, as it would not have required a separate mechanism for locating the last line in a file.

Efficient use of disk space is accomplished by several techniques. First, all blocks in the data structure may be anywhere on the disk, eliminating fragmentation problems. Second, the system keeps track of all discarded blocks so that they can be reused, and so that the data structure does not have to be reorganized periodically or "compressed" in some manner. Third, trailing blanks on the data lines are not stored, but are resupplied when the data is read back.

The handling of the final objective received the most attention in implementing this filing system, and will be described in the next section.

4. FAIL-SAFE DATA STRUCTURE

The third stated objective was the most difficult to implement. In order to be free of human intervention and run on a 360, the timesharing system has to be capable of co-existing with an operating system that fails repeatedly, forcing the following requirements:

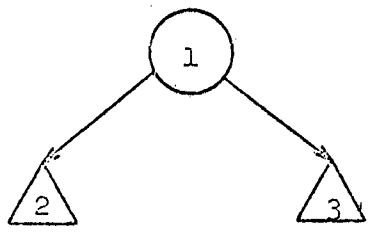
1. At all times the data structure on the disk must be an exactly correct data structure.
2. Blocks are allowed to be "lost," i.e. not be in the data structure or in the free list, but is is not legal to have a block mentioned in both places at the same time.

After dreaming up an implementation that meets the above requirements, one typically finds that the various functions which cause modification of the disk (writing and deleting of records) must be done in a rigidly determined order. Looking at the second requirement first, one sees that the free block lists on the disk must be modified to exclude the blocks about to be used before they are included in the data structure. To save disk operations, several blocks may be pre-allocated at the same time. As the number of preallocated blocks increases, however, one runs the risk of losing more blocks if the operating system should suddenly fail--the computer being restarted from disk. Fortunately, however, the number can be quite small (e.g. 10) and still keep the number of accesses to the free block lists at a small percentage of total disk accesses.

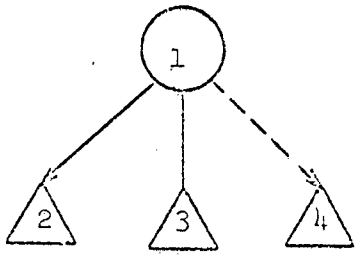
Free blocks that have never been used are contiguous and are represented by a starting disk address and a count while discarded blocks are kept on a chain. The chain represents the worst possible technique except, of course, of simply throwing away discarded blocks.

The requirement that the disk always contain a correct data structure is the difficult one since it requires a certain amount of otherwise needless activity if the file structure is to be crash-proof at every instant of time. A few simplified operations will be explained so that the application of this principle may be understood.

Imagine a file whose block structure is the following

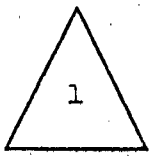


and which contains 40 lines of text. It is now desired to add some lines of text to the end of the file, but there is no room in block 3. A new block is added as follows:



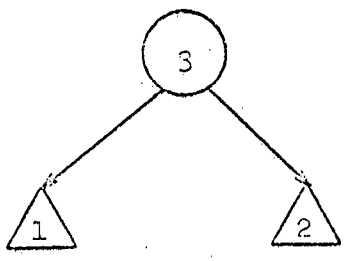
It is imperative that block 4 be written out before the directory (block 1) is rewritten to the disk with the new directory entry. Otherwise for a brief instant, the directory would point to a block containing garbage. A system crash at that time would be fatal.

The next situation involves the overflow out of a single data block.



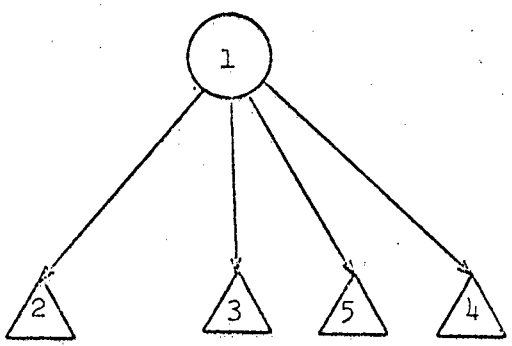
In order not to require extremely complex modification of the part of the tree which points to this file, it is necessary that the top block of the file (soon to be a new directory block) always have the same disk address. The technique utilized is to write out another copy of block 1 and the new data in block 2 at new disk addresses.

These new blocks are not in the tree structure yet, and a system crash at this time would simply cause the loss of the new data in block 2, and the extra copy of block 1, but would not cause an incorrect or unusable data structure. At this point, a directory block is written on top of the old copy of block 1.



As long as the output transfer is not interrupted--a problem to be discussed later--a crash-proof transition has been undergone.

Most operations of adding data to the file structure take place as described above. Data may be added anywhere in the file. For example, in the following diagram, block 5 contains an addition to block 3:



Deletions occur in a similar manner, but the order is reversed; directory and catalog entries are removed before underlying blocks are returned to the free block list.

Another problem arises when a data (or other) block is changed so that the directory indication of the first key in the data block is incorrect. The solution is to change the directory first when adding data lines and to change the data block first when deleting lines. If a system crash occurs between modification of the directory and the data block, the discrepancy is noticed the next time the data is accessed and the directory is changed to agree.

Discussion of some of the commands will illustrate some of the features available. Initially, a given "user" (another program) will have a pointer resting at the top of the tree, and no files opened. Let us assume that an *open* X.Y is then issued. The top file in the tree (the file indicated by the pointer) must be a catalog containing an entry for file X, or the command is rejected. File X must be a catalog with an entry for file Y, which may be either catalog or data. At this point, any of the following commands may be issued which refer to file X.Y: *read* (by key), *read next* record, *write* (by key), *append* to end of file (key is computed), *read last* record, and *close*. Special open commands are available for opening read-only files--opening files only if they already exist, opening files from the top of the tree rather than where the pointer is, etc. The pointer which may be moved by the *search* command is used to move the pointer. *Search account* moves the pointer and initiates checking individual disk space allocations.

5. CONCLUSION

Some difficulties with this filing system arise with the maximum line size of 254 characters selected and with speed of access to large quantities of data. Lack of speed is due mainly to the method of implementation. Some of these problems have been investigated in a graphics filing system used for storing pictures, with a maximum line size of 32,767 characters. Several improvements in the free block allocation scheme have been tried. Use of a bit map has been by far the most successful. A method was provided to allow the user to request many lines simultaneously so that the disk searches could be minimized. Other features have been added to keep frequently used blocks in core and to impose limits on the amount of disk space available to any user. Ways have also been found to identify blocks on the disk that were only partially written before the output operation was terminated, for example, by a power failure. A one-character counter is kept at both ends of each block, and is incremented each time the block is rewritten.

This system has been running quite successfully with the timesharing system at the University of Illinois with two 2314 packs of information. There have been two minor system crashes: one in 1968 due to a program error, and one in 1970 due to a failure in the 2314 drive. Periodic back-up tapes prevent failures from becoming serious problems.

BIBLIOGRAPHIC DATA SHEET	1. Report No. UIUCDCS-R-72-506	2.	3. Recipient's Accession No.

4. Title and Subtitle

A FAILURE TOLERANT FILING SYSTEM

5. Report Date: February 1972

6.

7. Author(s): Alfred D. Whaley

8. Performing Organization Rept. No.

9. Performing Organization Name and Address
University of Illinois
Urbana - Champaign 61801
Department of Computer Science

10. Project/Task/Work Unit No.

11. Contract/Grant No.: US AEC AT(11-1)1469

12. Sponsoring Organization Name and Address

Argonne National Laboratory
9600 South Cass Avenue
Chicago, Illinois 60600

13. Type of Report & Period Covered

14.

15. Supplementary Notes

16. Abstracts

A random access tree structure filing system has been constructed that is immune to operating system or internal failures through constant maintenance of a correct data structure on disk. Emphasis is placed on the fail-safe nature of the filing system, the speed of access to random lines in a data file, and the low level of human intervention required over an extended period of time.

17. Key Words and Document Analysis. 17a. Descriptors

Data Structure
Timesharing
Filing System
Information Retrieval
Tree Structure
Fail-Safe
Random Access
Fragmentation

17b. Identifiers/Open-Ended Terms

17c. COSATI Field/Group

18. Availability Statement	19. Security Class (This Report)	21. No. of Pages
Release unlimited	UNCLASSIFIED	16
	20. Security Class (This Page) UNCLASSIFIED	22. Price