Restart of an operating system having a permanent file structure

J. L. Smith and T. S. Holden
CSIRO, Division of Computing Research, P.O. Box 109, Canberra City, ACT, Australia

The problem of restarting an operating system after failure and recovering all files existing in the system at the time of failure is discussed. Different recovery procedures are described which depend among other things on the file address mapping scheme employed. Several models of general purpose systems are analysed to yield comparisons in I/O and recovery overhead.
(Received May 1970)

1. Introduction

This paper is concerned with a computer system recovery problem which has been termed 'warm start' (Needham and Hartley, 1969). An error or a failure has occurred in some system component which makes the contents of core store suspect or at least operator intervention is needed to restart the system. The system is one which provides permanent and temporary file storage on direct access secondary store. The administrative data necessary to control these files and storage is also maintained on the secondary store and this is the prime source of reinitialisation. The extent of the reinitialisation of the system software and its working data base will depend on the type of stoppage and the ongoing preservation action which is built into the operating system. Items which such actions might be designed to recover are:

1. Catalogued (permanent) files
2. Available secondary storage
3. Processes which were executing or awaiting execution
4. Uncatalogued (temporary) files associated with the above processes.

Obviously the recovery of any item in (3) is useless without its counterparts in (4) and vice versa. In (2) one is concerned with free storage and storage assigned to files which are not recovered. We will be considering mainly (1) and (2).

In a recent paper Fraser (1969) has set forth some sound principles for the recovery operations in a file system, covering aspects from warm start to complete reload. Particular reference is given to the file system at Cambridge University (Barron, Fraser, Hartley, Landy, and Needham, 1969) which provides 8 million words of direct access storage for sequential files and a flexible tape backup and archival system. Fraser points out that there is a large amount of redundancy in the administrative data base of a file system (for the purpose of operating efficiency) and inconsistency in this information can cause a catastrophic failure. Therefore his policy is to completely verify this data base at restart time and to carry out any deletions necessary to ensure consistency. This key problem of preventing inconsistency will be examined for various system designs.

2. File system characteristics

Many current file systems are based on ideas set forth in conjunction with the Multics system (Daley and Neumann, 1965). The administrative data base is organised into two parts, the dynamic part (Kerr, Bernstein, Detlefsen, and Johnson, 1969) being core resident in the main and pertinent only to the currently active files, and the static part (Bernstein and Hamm, 1969) residing on mass storage and containing information about all catalogued files in the system.

The static file structure may consist of one large directory file containing for each catalogued file sufficient addressing information to access it (whether in direct access or backup store), an indication of whether it is currently active and its control information (e.g. access control). The active file table (AFT) resident in core will contain the location of each open file, its length and its global control information.

Secondary storage is allocated and released dynamically, usually in fixed size blocks. Another component of the active data base is a map, at least partly core resident, showing all allocated and available storage. Logical to physical address mapping necessary to access a file may be accomplished in several basic ways:

1. *Linking* (L) The starting address of the file is contained in the directory or AFT entry. All subsequent blocks are accessed by a link address stored within each block of the file. In conjunction with this scheme it is common to avoid the use of a storage map and maintain free storage as a linked list of blocks in the same manner as each file's storage.
2. *Direct Indexing* (DI) The file entry in the directory or AFT contains the starting address of the file which must reside in physically contiguous storage. The file length defines the number of contiguous blocks allocated to the file.
3. *Indirect Indexing* (II) The file is accessed through a table containing the physical address of each block of storage allocated to the file. The required table length is determined by the file length. This table may be contained in the AFT entry or the directory entry, but it is usual that these contain a pointer to the table. II is the only address mapping scheme which allows the file addressing space to be used in the full sense of a virtual memory. Blocks can be allocated and released in the middle of the file leaving the length unchanged.
4. *Integrated Map and Index Tables* (IM) This scheme is described by Fraser (1969) and it is essentially a combination of schemes (1) and (3) above. The directory or AFT entry points to a map cell. A map entry (cell) for each allocated block of storage consists of a pointer to another cell of the map, which represents the next block allocated to the file concerned. This scheme removes the necessity for manipulating index tables separately by integrating them with the map.

Another indirect indexing scheme which has been successfully implemented (Hargraves and Stephenson, 1969) uses a range of block sizes increasing by powers of 2. Each index table entry points to a successively larger block of contiguous storage. However this will not be distinguished in this paper from the previously defined II scheme.

Ignoring the buffering and logical record operations (which may not be implemented in the main file system software) a representative set of primitive functions for file manipulation is:
catalogue and destroy file entries in the directory,
open and close a file,
read and write a file,
prune and insert records in a file,
lock and unlock a file.
Other primitives for access control are not relevant in the present context.

A characteristic of the file systems defined here is the dynamic allocation of storage for thousands of files stored on a small number of random access devices. No guarantees can be made as to which physical blocks of storage will be allocated to a file.

Therefore data organisation techniques designed for efficient processing of large files may be negated by the particular logical to physical address mapping which results in the system.

3. Computer system characteristics

The recovery problem may vary with different hardware configurations and operating system characteristics. A classification of the main hardware components is now given.

1. *Primary storage (PS) unit* This is usually a number of modules of core storage. It is immediately accessible (no latency or positioning delays) to processors and channels connected through a number of read/write ports.
2. *Processor* This is associated with one and only one PS unit to which it is directly connected via a port. It may initiate and receive notification of completion of block data transfers via channels also connected to this PS unit.
3. *Secondary storage (SS) unit* Normally this is not immediately accessible and it always has only one access port. Data transfers are usually executed in blocks for efficiency reasons.
4. *Channel* This can be connected between two ports by a processor, one port belonging to the PS with which the processor is associated. A channel may be available to more than one processor but only to one at a time. A block data transfer via a channel proceeds as follows:
   (*a*) it is initiated by a processor associated with the PS connected
   (*b*) the transfer proceeds without processor intervention
   (*c*) upon completion a notification is received by a processor associated with the PS connected.

Multiprogramming, multiprocessing (more than one processor directly connected to a PS unit) and parallel access to both the administrative data base and user files are techniques which can introduce complexities in the file system.

In the next section the recovery problem will be considered for a basic configuration including only one processor and one SS unit. Consideration of complexities arising in expanded hardware or software configurations will be deferred until Section 7.

4. Consistency and information loss

There are several premises to the successful use of the recovery procedures analysed here. At restart time the directory must be unscathed, or if sections of it have been damaged this must be recognisable when they are next read into PS (by constantly using error checking procedures) and the affected file entries deleted. Part of the preservation action will be to record on SS regularly updated versions of the storage allocation situation along with directory file updates. This information is used to reinitialise PS at restart, and a similar assumption must be made about the integrity of each section of these maps and tables. Too much destruction of these recovery data bases will necessitate a complete reload of SS from backup tapes. The administrative data base on SS should be in fixed preallocated areas not involved in the dynamic allocation scheme.

It is essential that the restarted administrative data base be consistent and that it should be as up-to-date as possible. The basic file system functions which cause the data base to become out of date are allocation and releasing storage for a file and cataloguing and deleting (uncataloguing) a file. If a failure occurs while the administrative data base is out of date there will be a loss of information on restart but some risk must be accepted in this area in order to obtain an acceptable recovery overhead. An inconsistent administrative data base can be made consistent by verification procedures which result in the deletion of files (Fraser, 1969)—the more inconsistent the data base the greater the number of intact files that will have to be discarded because of doubt about their integrity.

While the linked files and the direct indexing schemes have strong appeal because they are simpler to implement, they have serious deficiencies for large general purpose file systems. Linked files have a very high overhead in random accessing and the scheme requires threading of secondary storage in order to verify the administrative data base (Lockemann and Knutsan, 1968). Direct indexing cannot be used alone, especially when dynamic storage allocation is necessary, as contiguous storage cannot always be obtained, but the scheme can be used advantageously in conjunction with indirect indexing (Kerr *et al.*, 1969) and (Hargraves and Stephenson, 1969). In the ensuing analyses only the II and IM schemes will be considered, but the results can be trivially obtained for the other schemes.

One could design update procedures to ensure continual consistency of the administrative information on SS (perhaps with the aim of restart without a consistency check or to minimise information loss after restart). To achieve such consistency strict ordering would have to be observed in the update of the various administrative modules, and conflicts exist between the required order after allocation and that after releasing storage. For example before updating a directory entry to reflect an increase in file length one would assure that the new allocation was properly recorded on SS, but the reverse ordering would be necessary on truncation. Therefore a design may call for an independent update for each file according to urgency and consistency demands. However such schemes are generally impractical because of the overhead involved (an extreme case occurring with the use of virtual storage under II); furthermore they are still vulnerable to certain types of error which result in inconsistency (such as an obscure software bug the effect of which is nullified by a consistency check) and usually verification procedures are needed at restart in order to recover storage assigned to uncatalogued files.

Most file system designs permit inconsistencies in the administrative information on SS, but these inconsistencies should be detectable, isolated and fairly short-lived, and one accepts the prospect of a complete consistency check at each restart. The administrative updating procedure can then be organised as a global update with respect to all file storage modifications which have occurred since the previous update. In conjunction with this principle Fraser (1969) has increased reliability by creating a restart point on SS at update which is a separate copy of the administrative information; two or three of the most recent copies are retained in case of gross corruption of some restart points.

Guaranteeing consistent administrative information at restart does not ensure that corrupted files do not exist. These occurrences must be kept to a minimum and in some applications they would represent a serious security breakdown. Fraser (1969) has eliminated a prime source for the recovery of wrong information under IM by preventing reuse of deallocated storage until a global update has occurred. However if it is necessary to fall back two restart points at restart there is again scope for recovering wrong information. Apart from size considerations, this is an added reason for not including a copy of the directory file in a restart point, for those files which have gone through a reallocation of storage since a particular restart point was established could be recovered with wrong information in them by using that restart point.

The problem has an additional dimension under II because file corruption can be introduced through the index tables as well as by the storage allocation strategy. For example an implementation of II may include a file for storing index tables on SS and each directory entry would contain a pointer to a block in the index table file. Upon file deletion the associated index table block would have to be locked until an update occurred, for the same reasons that the actual released file storage would have to be locked.

The frequency of global updates of administrative information

will be determined by a trade off between the overhead involved, the cost of the potential information loss should a failure occur, and possibly the need to reuse storage or index table areas which have recently been released. At each update the option exists to create a copy of administrative information which is currently consistent with the directory contents.

5. Analysis of overhead

A measure of the file system overhead is the number of system block data transfers involved for each function. This is felt to be satisfactory without a weighting for the volume of data transferred because latency and positioning delays associated with current SS modules are likely to dominate actual data transfer times. The purpose of the I/O transfers can be classified into three areas (i) administrative update, (ii) support of user file I/O and (iii) restart.

In any implementation of the IM scheme the map would be subdivided into a number of segments each representing at least several hundred blocks of SS. Only a small number of these segments could be resident in PS but it would be essential for efficient operation that all the storage allocated to any one file be represented in one segment of the integrated map in the majority of cases. Likewise only a limited number of index table and map segments of an II implementation could be resident in PS. Because this map is more compact all the storage allocated to a file is likely to be represented on one part of the map. An analysis of each type of overhead is given below and when derivations of the analytical expressions are required they are given in Appendix 1.

The following symbols will be used in the analysis,

- k number of slots for index tables in PS
- K total numbers of index tables
- s number of slots for IM segments in PS
- S total number of IM segments
- c probability that a file reference causes a change in storage allocation
- p_i probability that a file reference results in chaining through i segments of the IM ($\sum p_i = 1$)
- m fraction of the map associated with II which can be resident in PS
- a number of active files
- f number of file entries per directory page
- F total number of files

Administrative update

The aim of this function is to bring the working copies of all administrative information on SS up to date with the status in PS. It may be necessary to withhold file operations during the update. The various steps under IM and II are given with the estimated number of transfers in parenthesis.

IM (i) update the directory file from modified AFT entries ($\leq 2a$),
(ii) write out all modified resident IM segments, unlocking storage ($\leq s$),
(iii) read-alter-rewrite remaining IM segments, unlocking storage ($2S/s$),
(iv) create restart copy of IM (S/s),

II (i) update the directory file from modified AFT entries ($\leq 2a$),
(ii) write out all modified index tables ($\leq k$),
(iii) write out resident section of map, unlocking storage (1),
(iv) unlock and initialise deallocated index table storage ($\leq 2K/k$),
(v) read-alter-rewrite remainder of map, unlocking storage ($2/m - 2$),
(vi) create restart copy of index table file (K/k).

This overhead could be considerably greater under II because of items (iv) and (vi). The index table file will always be sparsely occupied and so a large file has to be copied to create a restart point. It may even be necessary to temporarily obtain additional buffer space in PS in order to copy it efficiently. If the number of index tables to be initialised is small then item (iv) under II is not so significant.

Support of user file I/O

We will consider two models of the pattern in which files are referenced. In model 1 it will be assumed that in any file reference cycle there is only one primitive issued which results in I/O, a change in storage allocation to the file or both; then the file is unreferenced for a period of time such that under II there is no likelihood of its index table being preserved in PS or under IM the probability that the relevant section of the integrated map is resident in PS reaches a stationary value. In model 2 it will be assumed that there is a fixed number of active files in the system and there is equal probability that any one of them will be referenced next. It will also be assumed that each is referenced a sufficient number of times so that the effect of initial reference (e.g. directory operations) can be ignored. In this model the replacement strategy for index tables or map sections will be to displace the oldest resident in PS.

Model 1

Since there is only one reference to each file the long run average overhead will be approximately the same as if only one PS slot were used for swapping index tables or IM sections. However the average overhead is dependent on the total fraction of each map resident in PS. The expressions given below are for overhead per file reference.

IM

Define c_1 to be the probability that a displaced section of the IM has been altered.

Expected number of directory transfers $= 1 + c$

Expected number of map section transfers

$$= \sum_i p_i \sum_{j=1}^{i} \binom{i}{j} \left(\frac{s}{S}\right)^{i-j} \left(1 - \frac{s}{S}\right)^{j} j(1 + c_1) \qquad (1)$$

$$= \left(1 - \frac{s}{S}\right)(1 + c_1)\bar{p}$$

where

$$\bar{p} = \sum_i p_i\, i$$

II

Expected number of directory transfers $= 1 + c$

Expected number of index table transfers

$$= 1 + c \qquad (2)$$

Expected number of map transfers

$$= 2c(1 - m) \qquad (3)$$

Model 2

Define c_2 and c_3 to be the respective probabilities that a displaced index table or IM section has been modified.

IM

Expected number of integrated map transfers

$$= \sum_i p_i \sum_{j=1}^{i} \binom{i}{j} \left(\frac{s}{S}\right)^{i-j} \left(1 - \frac{s}{S}\right)^{j} j(1 + c_3)$$

$$= \left(1 - \frac{s}{S}\right)(1 + c_3)\bar{p} \qquad (4)$$

II

Expected number of index table transfers

$$= \left(1 - \frac{k}{a}\right)(1 + c_2) \qquad (5)$$

Expected number of map section transfers

$$= 2c(1 - m) \quad (6)$$

Restart

During the normal operation of the file system, segments of the working administrative data base will be displaced from PS to SS. This working copy of the administrative information on SS will be the most up to date from which to attempt a restart, but also the most likely to have been seriously corrupted. Nevertheless the same procedures can be employed no matter which copy of the administrative information is being used for restart. The aim is to regenerate a consistent administrative data base and one hopes to perform this with the minimum of file deletions. The reasons for deletion will be (i) violation of standard error checks on reading a segment of information, (ii) conflict because the same storage appears to have been assigned to more than one file, (iii) conflict on the file length determined from redundant information. Different implementation strategies may vary on when one of the above conditions makes it necessary to delete a file.

The time taken to restart may be considerably reduced by using the additional PS available at this time. Thus values of the parameters k, s and m could all be increased for the restart operation. It will be assumed that the number of inconsistencies discovered do not affect the overall restart time. Assuming that all information can be read successfully from SS, typical restart procedures are given below.

IM (i) read the directory, page by page, performing the operations (ii) and (iii) below for each file entry,
(ii) thread the integrated map from the entry point indicated in the directory, generating an additional map (for the purpose of detecting conflicts) with cells containing the identity of the file assigned the corresponding storage block,
(iii) if a storage conflict or length conflict is detected enter the file identities in a table,
(iv) delete all files entered in the above table and release the storage assigned according to the integrated map.

By a similar derivation to (1) it is seen that the expected number of map segments swapped in threading one file's storage is $(1 - s/S)n$, where n is the number of segment boundaries crossed ($n \geq 1$). Likewise the number of generated map segments swapped is $2(1 - s/S)n$.* This suggests an intolerable overhead if each file entry is processed independently, unless $s/S \sim 1$. However, if $n = 1$ in most instances, an alternative is to thread the storage map for all files of a directory page which are connected with resident map segments before swapping any map segments. Then the total number of transfers would be approximately:

$$F/f[1 + 3(S/s - 1)] \quad (7)$$

II (i) read the directory, page by page, performing the operations (ii) and (iii) below for each file entry.
(ii) read the index table, generating one map with each cell containing the identity of the file assigned the corresponding storage block and another map with each cell containing the identity of the file assigned the corresponding index table file block,
(iii) if a storage or length conflict is detected enter the file identities in a table,
(iv) delete all files entered in the table and transform the two generated maps into bit maps, releasing the storage involved in conflicts.

The major overhead is involved in accessing each index table and generating the storage allocation check map. No threading is necessary with index tables but each index table has to be retrieved independently. An efficient strategy would be to fill all k index table slots with the index tables belonging to files in a directory page, then swap the generated map segments only once for the processing of all k index tables. The total number of transfers is then approximately:

$$F/f + F/k[k + 2(S/s - 1)] \quad (8)$$

*It is assumed that the cell size of the generated allocation map (for IM and II) is the same as the cell size for the integrated map.

6. Expanded systems

A number of SS units may be connected to a PS unit, but in practice this should pose no complication to the recovery problem. On the one hand a complete copy of the administrative data base could be stored in one SS unit describing the state of every file and every block of SS in the system. There seems to be no point in allowing this copy of the administrative data base to cross SS boundaries. On the other hand a separate administrative data base could be maintained on each SS unit accounting for its own storage allocation with the additional restriction that no file could cross SS boundaries. In the latter case the recovery procedure would have to be exercised separately for each SS unit.

The file system process may be capable of parallel accessing of the administrative data base and thus capable of parallel processing of file primitives. This presents a synchronisation problem dependent on the logical structure of the data base and it has been given consideration elsewhere (Shoshani and Bernstein, 1969) and (Diijkstra, 1968). In a single processor configuration these problems should not occur in the storage allocation or recovery procedures. However in a multiprocessor configuration they can and the synchronisation problems are fairly obvious. The test and set lock instruction has been evolved as an effective synchronisation aid (Lampson, 1968).

Another type of parallelism in computer architecture has been described (Allen and Pearcey, 1969) and (Jones and Purcell, 1969), in which there is a network of processor-PS combinations some sharing access to SS units and each dedicated to particular system tasks. A simple example is where one processor-PS combination is devoted to file system management in order that another more powerful processor-PS combination may be devoted to more suitable computational tasks. The recovery problem here reduces to that already defined provided that the latter processor deals only with logical file manipulation, accepting all physical file parameters from the file system processor. The problem is considerably complicated if more than one such processor is involved in the file system management and recovery. Then an implicit restriction is that any file can be active only with one processor at a time. It is also apparent that a further channel control is required in which one processor can reserve a channel connection to a particular SS until it has carried out several transfers; this would allow coherent updating and access control.

The recovery and restart of executing processes in a general purpose environment is a considerable problem. In a recent paper (Crook, Smithies, and Raeburn, 1969) a thorough treatment has been given for magnetic tape oriented systems and the design goals can be applied to any such recovery system. Within the framework of file system recovery, a rerun point in a process may be established by creating and cataloguing a file containing the components necessary to restart the process (the segments of the process, its register settings and the equivalent of a descriptor segment (Bensoussen, Clingen, and Daley, 1969) with all file references accompanied by their catalogue name). It is implied that all scratch files known to the process at the rerun point must be catalogued.

7. Example

In this section the overheads in two systems having single processors are considered for the II and IM address mapping schemes.

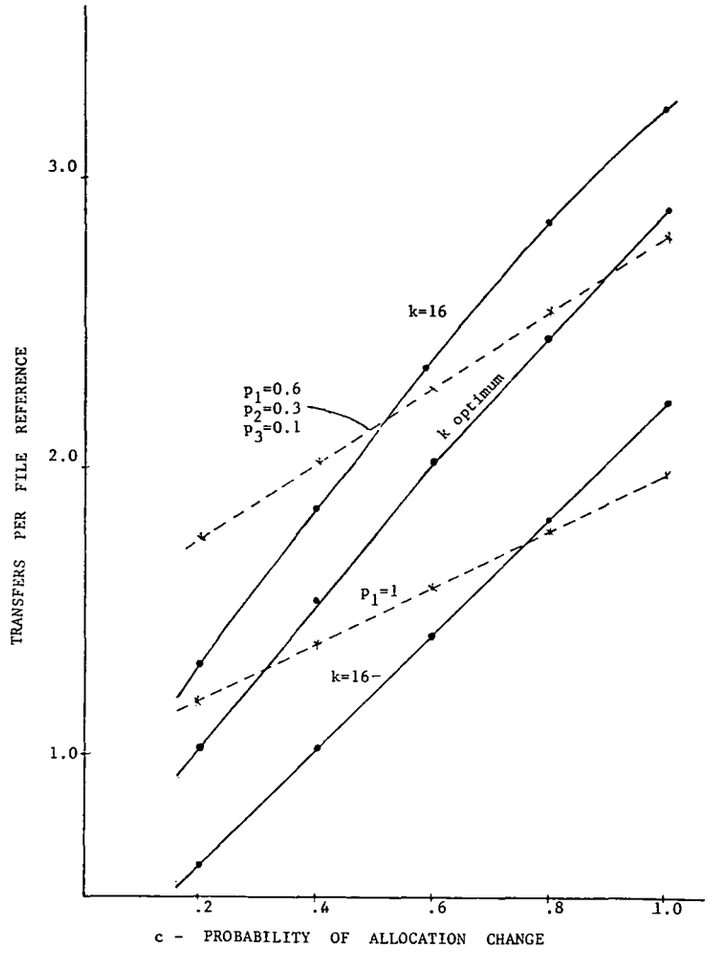
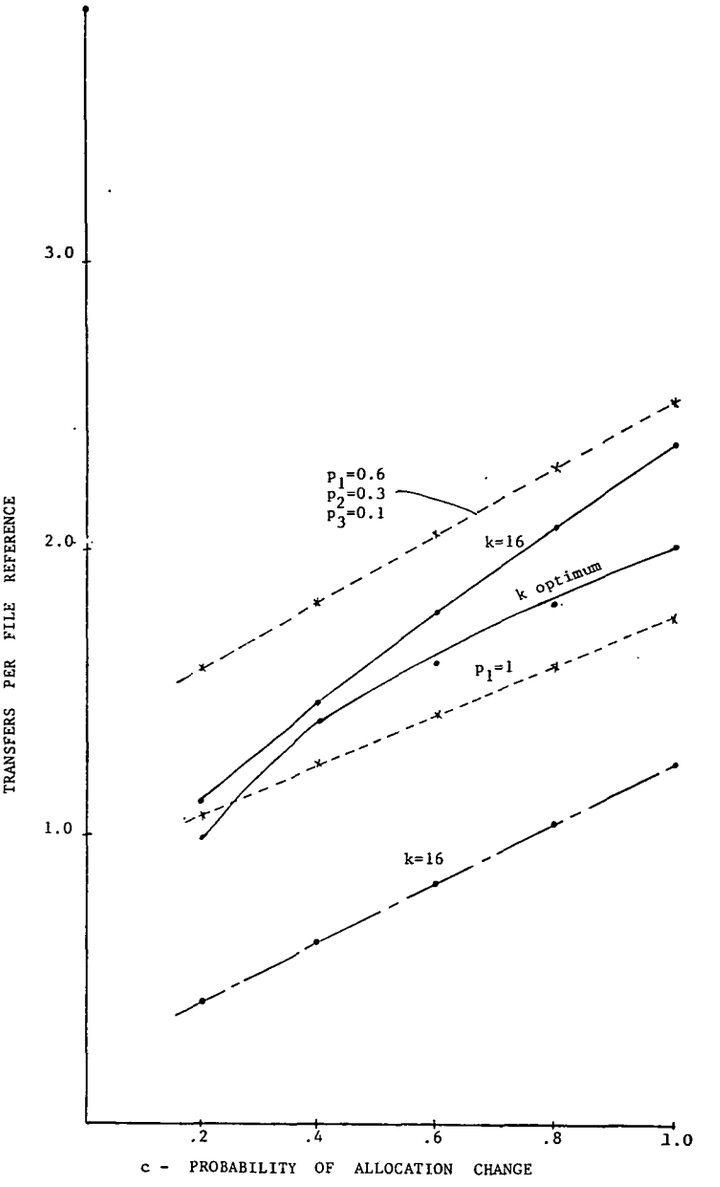


Fig. 1. File I/O overhead. (*a*) Model 1 configuration A



(*b*) Model 1 configuration B

If b is the number of blocks of SS in the system then the size of the map associated with II is $2b$ bits, b bits being necessary for the locking of released storage until update. The size of an integrated map including locking bits would be $b(2 + \log_2 b)$ bits.

It will be assumed that each index table can hold 64 entries so that its size is $64 \log_2 b$ bits, and it will also be assumed that one segment of the integrated map contains 512 entries. **Table 1** gives parameter values for two configurations. The amount of PS chosen for administrative tables was an arbitrary decision not necessarily optimal.

In **Fig. 1**(*a*), (*b*), (*c*), and (*d*) the overhead for user file I/O is plotted in terms of the expected number of transfers between PS and SS per file reference for configurations A and B under the file accessing patterns of models 1 and 2. Two curves are given for II, these being for the optimal value of k, and for $k = 16$ which roughly divides the PS area between map and index tables. Two curves are also given for IM these being for no file overflow between map sections ($p_1 = 1$) and for a case of considerable file overflow ($p_1 = 0{\cdot}6$, $p_2 = 0{\cdot}3$, $p_3 = 0{\cdot}1$, $c^* = 0{\cdot}9c$) (see Appendix). A further curve is plotted for a combined II and DI system where 75% of files are directly indexed.

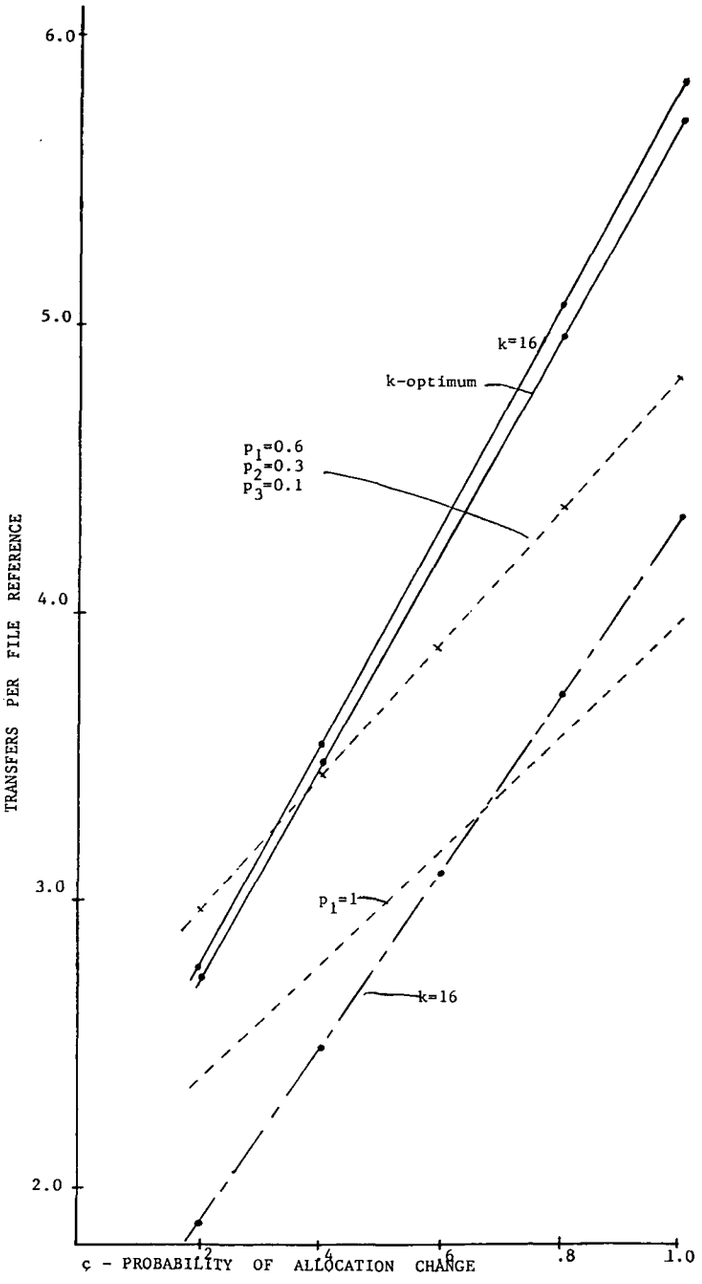
Provided that there is no file overflow between integrated map sections ($p_1 = 1$) the I/O overhead is significantly lower with IM than with II for either pattern of file accessing, with one exception. When the likelihood of a change in storage allocation on file access is small ($c < 0{\cdot}2$) II shows to advantage because references to the map are proportionately lower. If the likelihood of file overflow between integrated map sections is high, II becomes preferable for more typical values of c. This situation can be brought about by a combination of large files and storage fragmentation.

It is a simple matter to modify the models to reflect a fraction of files addressed in the DI mode. A good approximation is to reduce the number of index table swaps by this fraction (see Figs). The overhead would be further lowered by the effect of few index tables competing for the slots in PS.

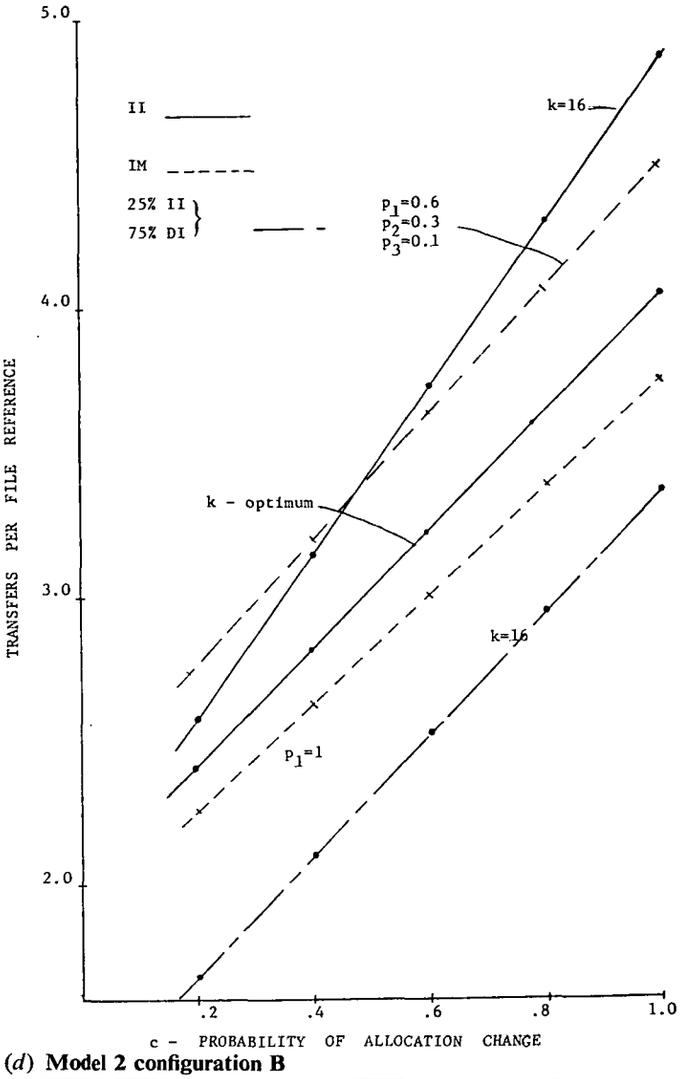
The question arises as to how complete a comparison is yielded by the models. In reality some file accessing patterns will be in bursts so that replacement strategies should take account of recent file references. In this case most improvement will be gained by reducing index table swaps.

Table 1

	CONFIGURATION A	CONFIGURATION B
b	2^{14}	2^{17}
PS for administration	31×2^{10} bits	38×2^{10} bits
m	$\frac{512 - 14k}{512}, k < 36$	$\frac{608 - 17k}{4096}, k < 35$
s	4	4
S	32	304
a	64	64



(*c*) **Model 2 configuration A**



(*d*) **Model 2 configuration B**

When one considers the recovery overhead, the total number of files, the size of the directory pages and the size of the AFT enter into calculations. Typically one would be dealing with from 2,000 to 20,000 files with say 200 active files. From the estimates for administrative update given in Section 5 this overhead will be dominated by the components $3S/s$ for IM and $3K/k$ for II. For the respective configurations under IM this represents 24 and 228 transfers, and under II between 180 and 1,800 transfers. These values determine the frequency at which one can afford to update the administrative information, and so it would be imperative to reduce the overhead under II by expanding the buffer area for index tables during the update phase or by combining it with DI.

Equations (7) and (8) show that the number of transfers for restart is directly proportional to F the number of files under II, and to F/f the number of directory pages under IM. Typically this means at least a factor of 10 difference in the restart time between II and IM.

By introducing directly indexed files (DI) the size of the index table file is proportionately reduced and so is the administrative update overhead. Restart time is also directly proportional to F/f for the DI files, and so the recovery overhead becomes comparable with IM, if most files are DI.

Conclusion

The IM scheme appeals strongly because its comparatively simple overall logic and its obvious superiority to linked files and direct indexing. It is likely to provide most efficient operation when restricted to small files on medium sized systems. Large files could cause the IM scheme to degenerate and a separate index table is much more efficient for manipulating a large file. Indexing offers two advantages when manipulating files, first the random access mode does not differ in overhead from sequential access, and second the file can be traversed in either direction with the same overhead. However the II scheme is likely to be impractical by itself particularly because of the recovery overhead, but when combined with DI it constitutes an efficient and versatile addressing scheme.

Obviously when files grow very large (such as in data base management systems) their administration is outside the scope of a general purpose file system. For large data base files physical continuity of storage must be strictly controllable and many applications call for painstaking reliability measures with many levels of recovery and restart.

Acknowledgement

The authors are indebted to the referee for his helpful criticism.

Appendix 1

The derivations of probabilities c_1, c_2 and c_3 are given. Let us first consider c_2 which is associated with the process of index

tables competing for k slots ($k < a$) in PS. If we assume that the process is in statistical equilibrium the probability of any index table being resident in PS is k/a. When an index table is displaced from PS it will have been resident during the displacement of $k - 1$ other index tables under the strategy assumed. If we denote by p_r the probability that during its residence there were r references to resident index tables we have (Feller, 1957):

$$p_r = \binom{k + r - 1}{r}\left(1 - \frac{k}{a}\right)^k \left(\frac{k}{a}\right)^r$$

Then denoting by q_n the probability that there were n references to the particular index table being displaced we have

$$q_n = \sum_{m=n}^{\infty} p_m \binom{m}{n}\left(\frac{1}{k}\right)^n \left(\frac{k-1}{k}\right)^{m-n} \quad \text{for } k \neq 1$$

$$q_n = p_n \quad \text{for } k = 1$$

Thus the probability that the displaced index table has been modified is c_2 where

$$1 - c_2 = \sum_{n=0}^{\infty} q_n (1 - c)^{n+1}$$

In order to similarly derive c_1 and c_3 we need to know the probability c^* that a reference to an integrated map section causes it to be modified. In practice this probability is strongly dependent on the primitive function; for example on storage deallocation all integrated map sections which contain entries for a given file will be modified. An average value for c^* is bounded above by c and below by

$$c^* < \sum_i p_i \, c/i$$

and this lower bound approaches c as $p_1 \to 1$.

Assuming c^* is a constant probability it follows in the same manner as above that

$$1 - c_3 = \sum_{n=0}^{\infty} q_n (1 - c^*)^{n+1}$$

where s and S replace k and a in the expressions for q_n and p_r. Thus we have

$$q_n = \sum_{m=n}^{\infty} p_m \binom{m}{n}\left(\frac{1}{s}\right)^m \left(\frac{s-1}{s}\right)^{m-n} \quad \text{for } s \neq 1$$

$$q_n = p_n \quad \text{for } s = 1$$

$$p_r = \binom{s + r - 1}{r}\left(1 - \frac{s}{S}\right)^s \left(\frac{s}{S}\right)^r$$

In the case where only one of s slots is used for swapping integrated map sections we have

$$1 - c_1 = \sum_{n=0}^{\infty} q_n (1 - c^*)^{n+1}$$

where q_n is as given above and

$$p_r = \left(1 - \frac{s}{S}\right)\left(\frac{s}{S}\right)^r$$

The derivation of equations (1) to (6) is typified by the following derivation of equation (3). With probability p_i a file reference results in chaining through i sections of the map (it is assumed that any stage in the chain the next section can be any one of the other $S - 1$ sections with equal probability). The probability that a required section of the map is already resident in PS is $\frac{s}{S}$. The probability that j of the sequence of i sections will be non-resident when required is

$$\binom{i}{j}\left(\frac{s}{S}\right)^{i-j}\left(1 - \frac{s}{S}\right)^j$$

If j sections are non-resident these will have to be transferred displacing j other sections. The displaced sections will have to be transferred to SS if modified. Thus the expected number of transfers is $j(1 + c_1)$. Summing over i and j yields

$$\sum_i p_i \sum_{j=1}^{i} \binom{i}{j}\left(\frac{s}{S}\right)^{i-j}\left(1 - \frac{s}{S}\right)^j j(1 + c_1)$$

$$= (1 + c_1) \sum_i p_i \, i \left(1 - \frac{s}{S}\right) \sum_{j=1}^{i} \binom{i-1}{j-1}\left(\frac{s}{S}\right)^{i-j}\left(1 - \frac{s}{S}\right)^{j-1}$$

$$= (1 + c_1)\left(1 - \frac{s}{S}\right)\bar{p}$$

where

$$\bar{p} = \sum_i p_i \, i$$

References

ALLEN, M. W., and PEARCEY, T. (1969). Developments in Machine Architecture, *Proc. Fourth Australian Computer Conf.*, Vol. 1, Adelaide, 1969, pp. 227-230.

BARRON, D. W., FRASER, A. G., HARTLEY, D. F., LANDY, B., and NEEDHAM, R. M. (1969). File Handling at Cambridge University, *AFIPS Conf. Proc.*, Vol. 30 (*SJCC* 1967), pp. 163-167.

BENSOUSSEN, A., CLINGEN, C. T., and DALEY, R. C. (1969). The Multics Virtual Memory, ACM Second Symposium on Operating System Principles, Princeton University, October 1969, pp. 30-42. Sponsored ACMSIGOPS.

BERNSTEIN, A. J., and HAMM, J. B. (1969). The design and implementation of a directory hierarchy for a general purpose operating system, Internal Report, 1969, General Electric Research and Development Centre, Schenectady, N.Y. (*to be published*).

CROOK, B. H., SMITHIES, A. P., and RAEBURN, J. H. (1969). Program rerun facilities in magnetic tape systems. *Proc. Fourth Australian Computer Conf.*, Vol. 1, Adelaide, 1969, pp. 159-165.

DALEY, R. C., and NEUMANN, P. G. (1965). A general-purpose file system for secondary storage, *AFIPS Conf. Proc.*, Vol. 27 (*FJCC* 1965), pp. 213-229.

DIJKSTRA, E. W. (1968). The Structure of 'THE' multiprogramming system. *CACM*, Vol. 11, No. 5, pp. 341-346.

FELLER, W. (1957). *An introduction to probability theory and its applications*, Wiley: New York.

FRASER, A. G. (1969). Integrity of a Mass Storage Filing System, *The Computer Journal*, Vol. 12, No. 1, pp. 1-5.

HARGRAVES, R. F., Jr., and STEPHENSON, A. (1969). Design Considerations for an Educational Time-Sharing System, *AFIPS Conf. Proc.*, Vol. 34 (*SJCC* 1969), pp. 657-664.

JONES, P. D., and PURCELL, C. J. (1969). Economics and Resource Parallelism in Large Scale Computing Systems, *Proc. Fourth Australian Computer Conf.*, Vol. 1, Adelaide, 1969, pp. 241-244.

KERR, R. H., BERNSTEIN, A. J., DETLEFSEN, G. D., and JOHNSTON, J. B. (1969). Overview of the R & DC operating system, Internal Report, 1969, General Electric Research and Development Centre, Schenectady, N.Y. (*to be published*).

LAMPSON, B. W. (1968). A Scheduling Philosophy for Multiprocessing Systems, *CACM*, Vol. 11, No. 5, pp. 347-360.

LOCKEMANN, P. C., and KNUTSAN, W. D. (1968). Recovery of Disk Contents After System Failure, *CACM*, Vol. 11, No. 8, p. 542.

NEEDHAM, R. M., and HARTLEY, D. F. (1969). Theory and Practice in Operating System Design, ACM Second Symposium on Operating System Principles, Princeton University, October 1969, pp. 8-12. Sponsored ACMSIGOPS.

SHOSHANI, A., and BERNSTEIN, A. J. (1969). Synchronisation of a Parallel-Accessed Data Base, *CACM*, Vol. 12, No. 11, pp. 604-607.