A COMPARISON OF FILE ORGANIZATION TECHNIQUES

THESIS

Presented to the Graduate Council of the
North Texas State University in Partial
Fulfillment of the Requirements

For the Degree of

MASTER OF SCIENCE

By

Roy Lee Rogers, B. S.
Denton, Texas
August, 1977

IAC

Rogers, Roy Lee, A Comparison of File Organization Techniques, Master of Science (Computer Science), August, 1977, 99 pp., 5 tables, 15 illustrations, bibliography, 23 titles.

This thesis compares the file organization techniques that are implemented on two different types of computer systems, the large-scale and the small-scale. File organizations from representative computers in each class are examined in detail: the IBM System/370 (OS/370) and the Harris 1600 Distributed Processing System with the Extended Communications Operating System (ECOS).

In order to establish the basic framework for comparison, an introduction to file organizations is presented. Additionally, the functional requirements for file organizations are described by their characteristics and user demands. Concluding remarks compare file organization techniques and discuss likely future developments of file systems.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER I

INTRODUCTION

File organization techniques have received attention periodically over the years from persons interested in information retrieval and from persons interested in file-oriented computer applications in business, engineering, scientific, and government work. This thesis presents by means of survey and contrasts some implications for file organization techniques used on large scale and small scale computer systems. To accomplish this it examines and compares the file structures used on representative systems from these two classes: the IBM System/370 (OS/370) and the Harris 1600 Distributed Processing System with the Extended Communications Operating System (ECOS) respectively.

The following is a brief outline of this thesis. Chapter II classifies and introduces various techniques of file organizations, with a survey of some interesting file organizations. During research of file organization techniques, it was observed that a great wealth of useful and different material has been published, but that no single work provides an adequate survey of all file organizations. For this reason, it was decided to include a large portion of this material and to present it in a manner so that other researchers with limited knowledge of

file organization techniques would have these fundamental developments in a single source. Therefore, some of the introductory material of this chapter will be familiar to advanced students of file organizations and file structures. Six general file organization techniques are surveyed: the sequential, the random, the list, the tree file structures, the indexed sequential, and the attributed string.

In Chapter III the functional requirements for file organizations are examined via the characteristics of file organizations, and the demands users place upon file organizations. To that end, a statement of the users' common demands are graphed into four categories. The common user demands considered are naturalness, ease of access, ease of maintenance, and the extend of software and hardware support.

Chapter IV examines the file organizations of a large scale computer system and a small scale computer system. The file organization techniques are contrasted from a critical viewpoint regarding variations in file organization, access methods, and data management facilities. The chapter is not exhaustive but does provide important terminology for use in the comparison of the file organization techniques. Chapter V presents a comparison of file organizations for the two computer systems.

The final chapter summarizes the comparative study of file organization techniques which exist in large scale computer systems and small scale computer systems. Among the

contents of this chapter is information concerning the future developments of file systems.

Two appendixes are included. The first presents some basic key words and phrases and a brief description of new phrases used in Chapter IV. Because terminology is expanding somewhat faster than it is being standardized, certain definitions offered in this thesis and appendix are not absolute and some file organization specialists may use different terminology. The second appendix exemplifies the file system performance for the small scale computer system.

CHAPTER II

SURVEY OF FILE ORGANIZATIONS

This chapter classifies and introduces various techniques of file organizations. The six file organization techniques surveyed in this chapter are the sequential, the random, the list, the tree file structures, the indexed sequential, and the attributed string.

In practice, many of these exist not only in pure forms but also in hybrid and modified forms. Some are used in a combined manner. That is, at one level a file may be organized by using one technique and at another level may be organized by using a different technique (4). Therefore, the techniques observed in actual use may not be pure forms. For clarity of introduction, however, pure forms will be considered here.

The six file organizations selected here are obviously not the only forms of data structure possible for use as file organization techniques. For example, the file organizations for simple list, simple ring, and the binary tree have been omitted. In the first two cases, more complex forms of these file organizations are surveyed because of their greater popularity as file organization techniques. For the same reason, a general form of tree file structures has been included instead of a binary tree.

Table I is a sample file that is used to illustrate the discussion of the first four file organizations. The file

TABLE I

EXAMPLE FILE--I

Record Number	Automobile			Owner			
	License	Make	Model Year	Name	Street Address	City	State
1	JBG-018	Chevrolet	1967	Brian K. Morris	559 Oak Lane	Austin	Tx.
2	JDC-062	Plymouth	1973	Erica A. Rogers	8003 Richmond	Dallas	Tx.
3	KIA-548	Mercury	1972	LeRoy C. Berry	1627 Dumont Dr.	Carthage	Tx.
4	FED-607	Chevette	1976	Nettie Jones	5815 Shady Crest Tr.	Dallas	Tx.
5	FED-608	Triumph	1969	Eleanor L. Jones	5815 Shady Crest Tr.	Dallas	Tx.

contains data on automobiles and their owners, as might be maintained by a state department of motor vehicles. Figure 1 shows a record layout that might be used for each record of this file. Each of the other file organizations will be illustrated separately as they are discussed.

Sequential Organization

Sequential organization was, historically, the first to be developed and is the best known. In a sequential file, records are arranged in position according to some common attribute of the records (10, p. 117). Sometimes records in a sequential file are stored in order of their arrival in the file. Figure 2 shows part of the example file organized as a sequential file, where the file is ordered by license number.

The principal advantage offered by sequential organization is rapid access to successive records. That is, if the nth record has just been accessed, then the (n+1)th record can be accessed very quickly. This is always true if the sequential file is resident on a single-access auxiliary storage device, such as a tape drive. However, if the file is stored on a multi-access device, such as a disk drive, then, in a multiprogramming environment, head motion caused by other users may reduce this advantage.

A sequential file is searched by scanning the entire file until the desired record is found. For a large file, this is a lengthy process. For this reason, transactions for a

LAST NAME	FIRST NAME, INITIAL
STREET ADDRESS	
CITY	STATE
AUTO MAKE	MODEL YEAR
LICENSE NUMBER	

Figure 1. Record Format for Example File--1

Morris	Brian K.	5559 Oak Lane	Austin	Tx.	Chevrolet	

1967	JBG-018	Rogers	Erica A.	8003 Richmond	Dallas	

Tx.	Plymouth	1973	JDC-062	Berry	Leroy C.	

1627 Dumont Dr.	Carthage	Tx.	Mercury	1972	KIA-548	

Figure 2. Sequential Organization of Example File--1

sequential file are usually accumulated, sorted so that they are in the same order as the file, and then presented all at once for processing, so that only one pass through the file is required. Using such batching techniques, it is possible to process sequential files at a very low cost per transaction. Obviously, a sequential file is not well suited to an online application, where the transaction batch size is small.

When a file is to be processed on the basis of more than one key, efficiencies that are achieved by ordering the file on a single key are impossible unless a duplicate copy of the file, ordered on the second key is maintained. The alternate copy can then be used to determine the primary keys of the record to be processed, and standard sequential processing techniques can be used. This technique adds many steps to any process accessing the file, and may greatly reduce efficiency.

In summary, sequential organization permits rapid access to successive records and is the most economical implementation of a file that will have large batches of transactions processed against it according to one key. However, processing and retrieving records out of sequence are very slow, and small volume updates are inefficient, since any update requires recopying of the entire file. Other major strengths of the sequential organization are its economy of storage, and its speed for access by attributes if the cost of ordering (that is, sorting) be ignored, and if the attribute by which the user desires access is reflected in the key used for ordering.

Random Organization

The records in a randomly organized file are arranged according to some established relationship between the key of the record and the location of the record on direct-access storage; records are stored and retrieved through the use of this relationship (10, p. 118). There are three types of random organization: direct address, dictionary lookup, and calculation.

Direct Address

When the address of the record is known, this address can be used directly for storage and retrieval. This presumes that some address bookkeeping is performed outside the computer system, or that each record contains some field that is used directly as the key. Figure 3 shows a direct organization of the example file. In this case the last three digits of the vehicle license number must be known to access a record.

The direct address method of organization is based upon some predictable relationship between the key to the record and its location within the file. In its simplest form, the key would be the address of the cylinder and track on a DASD where the record is placed. With this system, the location of the record is known as soon as the key is known, so that the record can be read directly.

A desirable characteristic of the direct address method of organization is that records should be distributed as evenly

Morris	Brian K.
5559 Oak Lane	
Austin	Tx.
Chevrolet	1967
JBG-018	

018

Rogers	Erica A.
8003 Richmond	
Dallas	Tx.
Plymouth	1973
JDC-062	

062

Berry	Leroy C.
1627 Dumont Dr.	
Carthage	Tx.
Mercury	1972
KIA-548	

548

Jones	Nettie
5815 Shady Crest Tr.	
Dallas	Tx.
Chevette	1976
FED-607	

607

Jones	Eleanor L.
5815 Shady Crest Tr.	
Dallas	Tx.
Triumph	1969
FED-608	

608

Figure 3. Direct Address Organization of Example File--1

as possible throughout the file. If an attempt is made to write too many records to a file that has already been allocated, there will not be room to add the record and it must be placed in some other location. Retrieving records which are placed anywhere other than their originally calculated location makes programming more complex and increases access time. The chief advantage of this type of random organization is speed of access.

Dictionary Lookup

A dictionary is a table of two-element entries, each of which specifies a key-to-address transformation (10, pp. 119-120). When a record is added to the file, an entry is added to the dictionary; when a record is deleted, a dictionary entry is removed. Retrievals are performed by looking up the desired key in the dictionary, and then using the address obtained from the dictionary to access the record. Figure 4 shows a dictionary for the example file, which could be used for access based on owner's surname.

Since each reference to a record in the file requires a search of the dictionary, the search strategy used has great influence on the effectiveness of the file design. The two search strategies that are commonly employed are the binary search and sequential scan.

If the dictionary is not maintained in any collating sequence, a sequential scan is the only method that can be used to obtain an address. If the dictionary has n entries,

Berry	*3
Jones, N.	*4
Jones, E.	*5
Morris	*1
Rogers	*2

*Record Number

Figure 4. Dictionary for Example File--1

then, on the average, (n + 1)/2 accesses will be necessary to obtain an address.

On the other hand, if the dictionary is maintained in collating sequence of the keys, a binary search is possible. The binary search begins by first testing the key at the location that is a power of 2 nearest to the middle of the dictionary. A comparison indicates whether it is the desired key, and, if not, in which half of the file the key is located. This operation is then repeated, eliminating half the remaining dictionary at each step, until the desired key is located, or its absence is established.

Calculation

In the calculation method, a key is converted into an address by some computational manipulation of the key (10, pp. 120-121). Since the set of addresses produced is much smaller than the set of possible keys, the addresses generated from distinct keys are not always distinct.

Two processes, compression and hashing, are discussed here; only hashing, however, is an address calculation method. Compression is included because an important performance measure of both compression and hashing algorithms is the extent to which they map different keys into the same transformed key.

Compression, called "abbreviation" by Bourne and Ford (2) is the transformation of the key so that it requires as little

storage as possible, and yet retains as much as possible the discrimination and uniqueness of the original key (2). Compression is normally applied to keys that are words or names in natural language, so as to remove characters that add the least to the information content of the keys.

Hashing is a transformation on keys that produce a uniform spread of addresses across the available file addresses (9). Thus, hashing is used to transform a key (that may have been compressed) to an address. A popular hashing algorithm is to split the key into two parts, multiply the two halves of the key, and use the middle bits of the product as the hash address.

Compression techniques are especially useful in cases where the keys may contain errors, such as may occur in systems performing retrievals based on names, particularly if the name has been transmitted verbally. Specialized compression techniques based on phonetics have been developed that map various spellings of the same word into the compressed key (5). A name-based retrieval system is an example where both compression and hashing might be used together.

List Organization

This section describes list structures, which include linear lists, inverted files, and rings. For each list structure, methods for inserting and updating are outlined.

In dealing with list structures, it is desirable to have a compact notation for specifying algorithms for manipulating them. A convenient notation for this purpose has been introducted by Knuth (7). Every record consists of a number of fields, some of which are pointers; therefore, all operations on lists of records can be expressed in terms of operations on fields of records. Let the notation also include link variables whose values are pointers.

A pointer to a record is the address of that record, expressed in a way that permits the direct location of the record. Thus, a pointer can be an actual disk address, or it can be an address relative to the disk address of the first record of the file, or some other quantity. By the use of pointers to imply linking relationship between records, it is possible to completely divorce the physical and logical arrangement of a file. In fact, through the use of pointers, it is even possible to represent recursive data structures, which have no representation without pointers.

The fundamental component of a list is a record, as defined above, where one or more of the fields may be pointers. Then a list can be defined as a finite sequence of one or more records or lists.

Lists that are not recursive and do not have loops or intersections can be represented without the use of pointers, if physical ordering is used to represent the linking relationship between records. This type of allocation is called

sequential allocation; in contrast, the use of pointers to join related records is called linked allocation. Linked allocation schemes are easier to update, but require more storage for pointers.

In the figures showing list file organizations with pointers, pointers are represented by arrows from the record containing the pointer. The end of a list is indicated by a pointer of some special value, often zero; in the figures this end-of-list indicator is represented by the symbol used to represent "ground" in circuit diagrams, after Knuth (7):

Linear Lists

A linear list is a set of records whose only structure is the relative linear positions of the records (7). There are special names for linear lists in which insertions and deletions are made at one end of the list (7):

A stack is a linear list for which all insertions and deletions are made at the same end, called the top.

A queue is a linear list for which insertions are made at one end, called the back, and deletions are made at the other end, called the front.

A deque (contraction of "double-ended queue") is a linear list for which insertions and deletions can be made at either end, called the left and right of the deque.

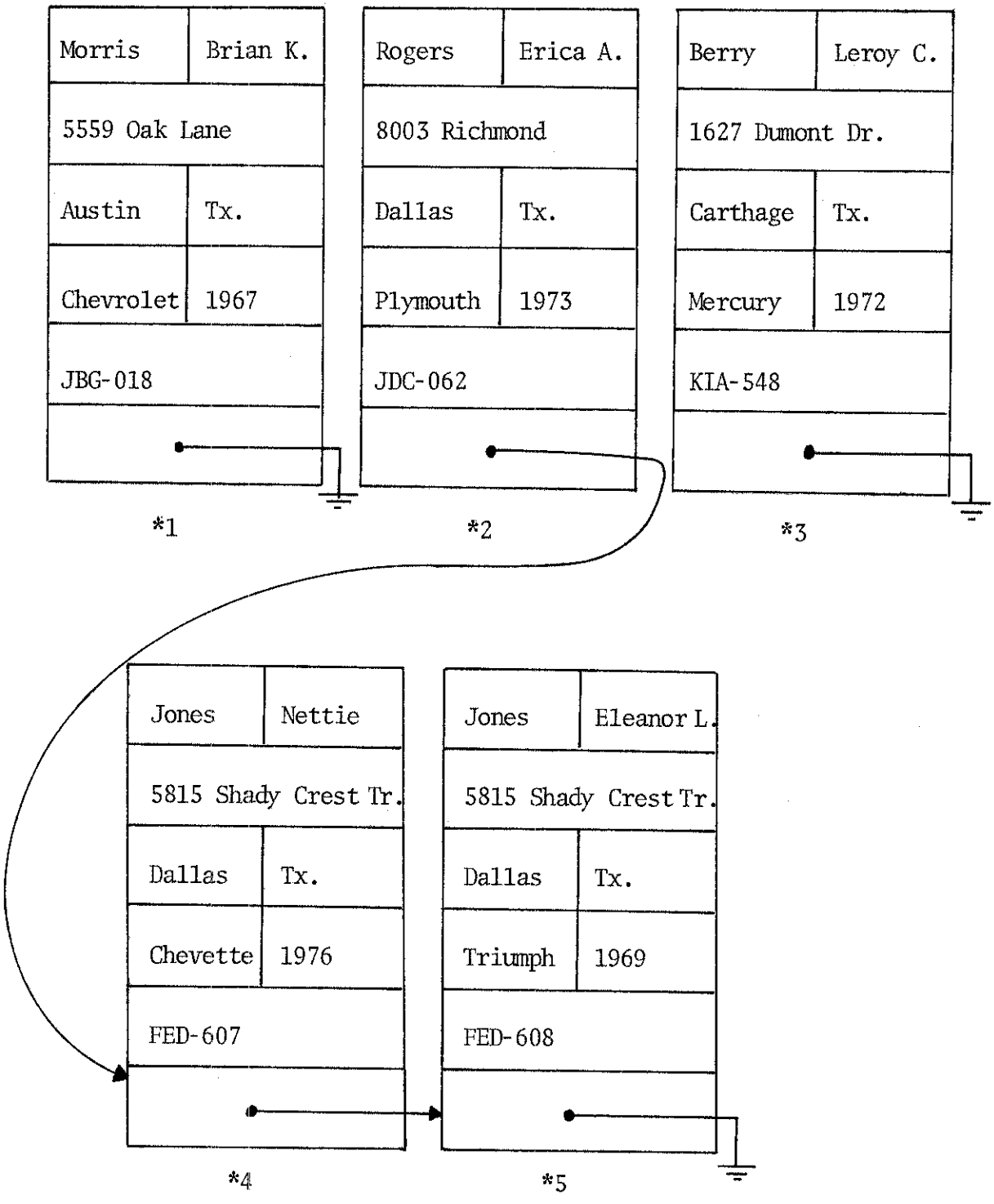
These three structures are encountered frequently. They are sometimes also called queueing disciplines; a stack is a

LIFO (last in first out) queue, a queue is a FIFO (first in first out) queue, and a deque is a queue that can be used in either way. These names reflect the primary use of these structures--the construction of various types of task queues. They also occasionally are useful as intermediate files in applications that require complex retrieval processes on large files. For example, a stack might be used to accumulate the results of a search of a file on a primary key for later qualification by other processes.

A linear list may be implemented using either sequential or linked allocation, but implementation is not limited to these restricted cases. In general, additions and deletions may be made at any point in a list, and a record may be on several lists at once. The great ease with which additions and deletions can be made in a linked list is one of the chief advantages of linked allocation. Figure 5 shows a linear list organization for the example file, where a separate list has been used for each distinct city of residence in the file. The first spare field has been used as the list pointer.

Inverted Files

This second method of implementing file structures for generic and multiple keyed records is called the inverted file organization (8). By this method, another file is inserted as an intermediary between the index and the data file. This intermediary file contains records that are simply lists of



*Record number

Figure 5. List Organization of Example File--1

pointers to the data file records. Every key in the index points to one of these lists, called an inverted list (8, p. 64), and the list in turn points to all of the records in the data file that contain the given generic key.

An inverted file is composed of a number of inverted lists. Each inverted list is associated with one particular value of some key field of a record and contains pointers to all records in the file that contain that value of the key field. Inverted lists are normally produced for all fields of the record, permitting the file to be accessed on the basis of any field (10, p. 123), Figure 6 shows an inverted file structure for the example file. Since the file is inverted on the basis of all fields, any one can be used to access a record. Note that the longest inverted list is the one for "Tx.", which appears in all records in the example file.

An inverted file permits very rapid access to records based on any key. However, updating an inverted file structure is difficult, because all the appropriate inverted lists must be updated. For this reason, an inverted file structure is most useful for retrieval if the update volume is relatively low or if updates can be batched.

A variation of inverted file structure that includes features of lists is multilist structure. A multilist consists of a sequential index that gives, for each key, the location of the start of a list that links together all records characterized by that key value. A multilist can be regarded as

Austin	Berry	Brian K.	Carthage	Chevette	Chevrolet	Dallas	Eleanor L.
*1	3	1	3	4	1	2, 4, 5	5

Erica A.	FED-607	FED-608	JBG-018	JDC-062	Jones	KIA-548
2	4	5	1	2	4, 5	3

Leroy C.	Mercury	Morris	Nettie	Plymouth	Rogers	Triumph
3	3	1	4	2	2	5

Tx.	1627 Dumont Dr.	1967	1969	1972	1973	1976	5559 Oak Lane
1,2,3,4,5	3	1	5	3	2	4	1

5815 Shady Crest Tr.	8003 Richmond
4, 5	2

*Record number

Figure 6. Inverted Organization of Example File--1

an inverted list structure in which all entries after the first in each inverted list have been represented by lists in the file rather than by entries in the inverted list.

Figure 7 shows a multilist organization of the example file. Note that one link field is needed for each of the key fields in the original record, since exactly one list will pass through every record for each field in the record.

If the lists are divided into pages, if pointers in the index refer to records by page and record number within page, rather than record number within the file, and if each list is restricted in length to one page, the structure is called a cellular multilist (10, p. 126). In a cellular multilist, then, each inverted list is represented by a number of sublists, where a sublist is a linked list within a page. The index points to the first record of each sublist.

A multilist is easier to update than an inverted file because it avoids the necessity for complete reorganization of the sequentially allocated inverted lists, but retrievals are slower than with an inverted file because the lists must be traversed to perform a retrieval. A cellular multilist organization lies midway between the inverted file and multilist, both in updating difficulty and in retrieval speed, because it represents the inverted lists by a structure that is partially linked (the file) and partially sequentially allocated (the index).

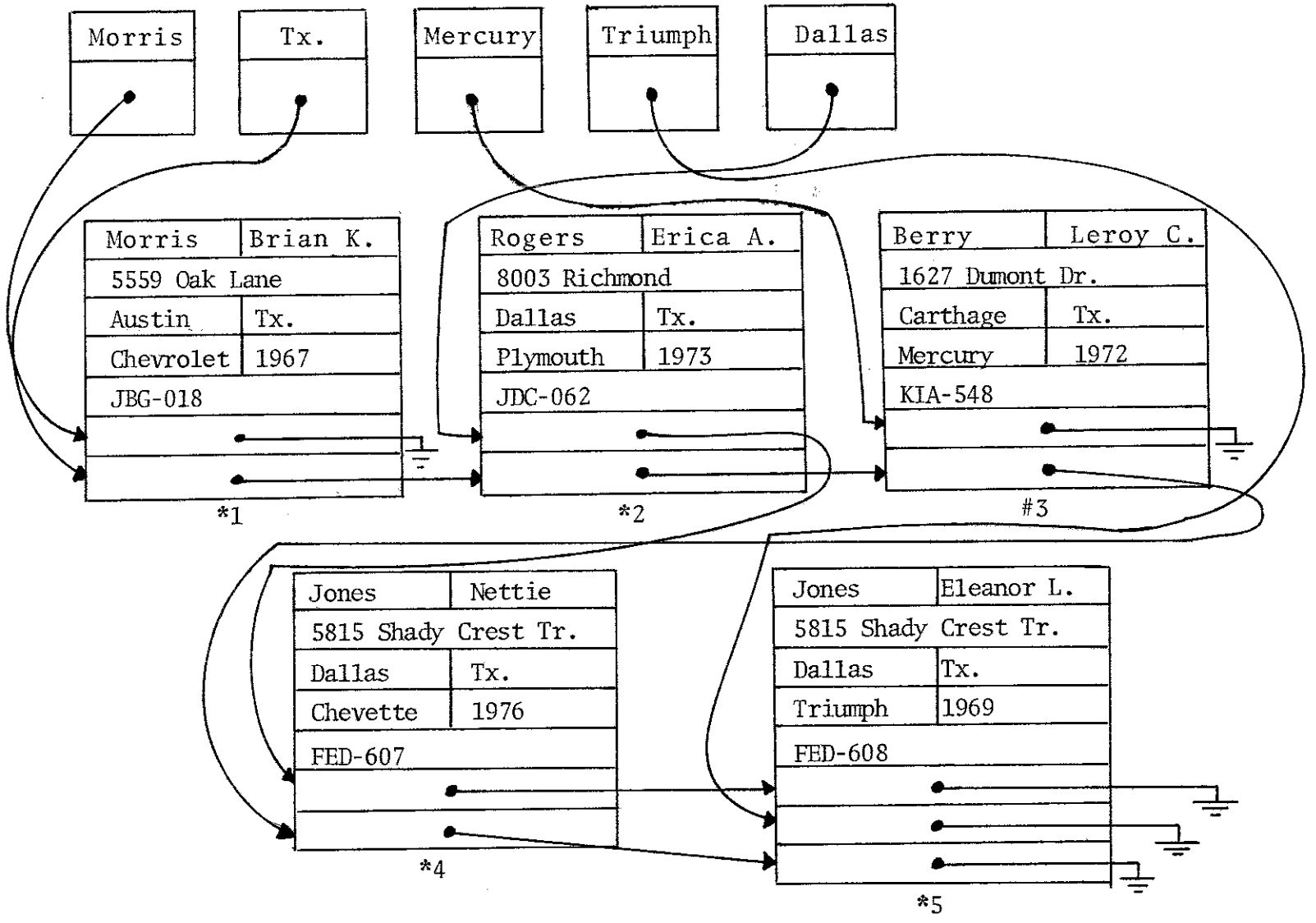


Figure 7. Multilist Organization of Example File--1

Rings

A ring is simply a linear list that closes upon itself (10, p. 126). A ring is very easy to search--a search starting at any element of the ring can search the entire ring. This facilitates entry into a ring from other rings. The danger that an unsuccessful search may continue endlessly around the ring is solved very simply if the search program saves the record number of the first record it searches; the record number of each record that is searched is then compared to this stored number, and the search terminates unsuccessfully when it returns to its starting point.

One important use of rings is to show classifications of data. A ring is used to represent each class; all records in that class are linked to its ring. If a tag field is included in each record along with each pointer, then a class can be searched for the records it contains, or a record can be searched for the classes which contain it with equal ease. The classification scheme can be hierarchic; if it is, a hierarchic search of the file can be performed very easily starting at the highest level ring.

Figure 8 shows a ring organization of the example file. This example shows only a city ring; other rings could be incorporated if the file was to be entered using other keys. The organizations of Figures 7 and 8 are very similar; in fact, the only two differences are the circularity of the ring

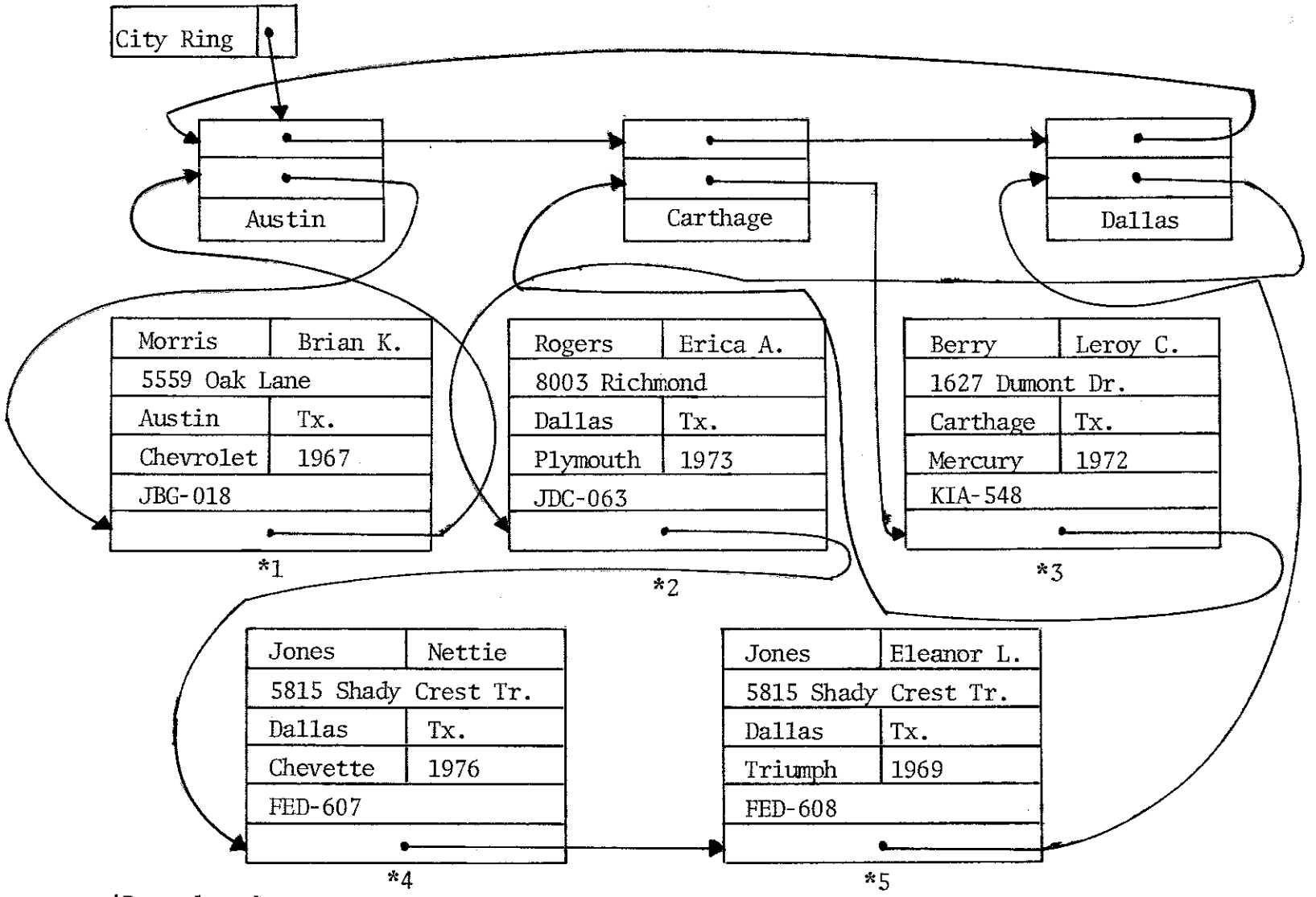


Figure 8. Ring organization of Example File--1

(rather than having some end) and the ring versus sequential organization of the index information. If a file is to be accessed using a variety of types of data, using a ring structure to classify the index information reduces the average access time.

Updating a ring structure is slightly easier than with other list structures because all the pointers to be modified during an update can always be found by traversing the ring. Because a ring has no end and no beginning, and because, given some record number x, the record on each side of x can always be located by moving forward through the ring (10, p. 150).

There are two chief disadvantages associated with the use of ring structures: the overhead introduced by the pointers (which is essentially identical to the overhead associated with any list organization), and the number of operations required to locate a record. With an inverted file, for example, if a record is to be located on the basis of three keys, the three appropriate inverted lists can be used in combination to locate the desired record very quickly; with a ring structure, one of the three rings would have to be searched sequentially. The reason that only one ring structure is searched sequentially is, once the appropriate ring is located that contains records with all three keys, a sequential search through the ring would be necessary to locate the desired record. Thus, if the rings are very long, the search time to locate a record can be long.

Tree File Structures

A binary search is used for a sequentially allocated random-access file that is stored in order of the collating sequence of its keys. This arrangement reduces search time at the expense of update time. For a file that is updated more often than it is searched, linked allocation can be used to minimize update time at the expense of search time. For a file that is updated and searched with similar frequency, however, neither of these approaches is very practical, and some sort of compromise must be struck. A tree structure is such a compromise, combining the speed of a binary search with the updating ease of linked allocation (10, pp. 151-153).

A precise definition of a tree structure can be expressed easily with the use of elementary graph theory. The following is a modification of definitions used by Sussenguth (11):

A graph is a set of nodes and branches joining pairs of nodes.

A path is a sequence of branches such that the terminal node of each branch coincides with the initial node of each succeeding branch.

A graph is said to be strongly connected if there exists a path between any two nodes of the graph.

A circuit is a path in which the initial and final node are identical.

A tree is a strongly connected graph with no circuits.

The root of a tree is a node with no branches entering it. The root of the tree is said to be at the first level; a node which lies at the end of a path containing j branches from the node is said to be at the $(j + 1)$th level.

A leaf is a node with no branches leaving it.

The filial set of a node is the set of nodes which lie at the end of a path of length one from the node.

The set of nodes reachable from a node by moving toward the leaves are said to be governed by the node.

From the above definitions, it follows that a tree will always have one and only one root; a tree of n nodes will always have exactly $(n - 1)$ branches; and $(n - 1)$ is the smallest number of branches required for the tree to be strongly connected.

Tree-organized files are used most often as indexes to some other file. Such an arrangement permits a record in the tree file to consist of only keys, pointers to other records in the tree, and addresses in the file. This approach is particularly useful if the records in the file are variable in length. If the file consists of short, fixed-length records, then the entire record can be placed within the tree structure.

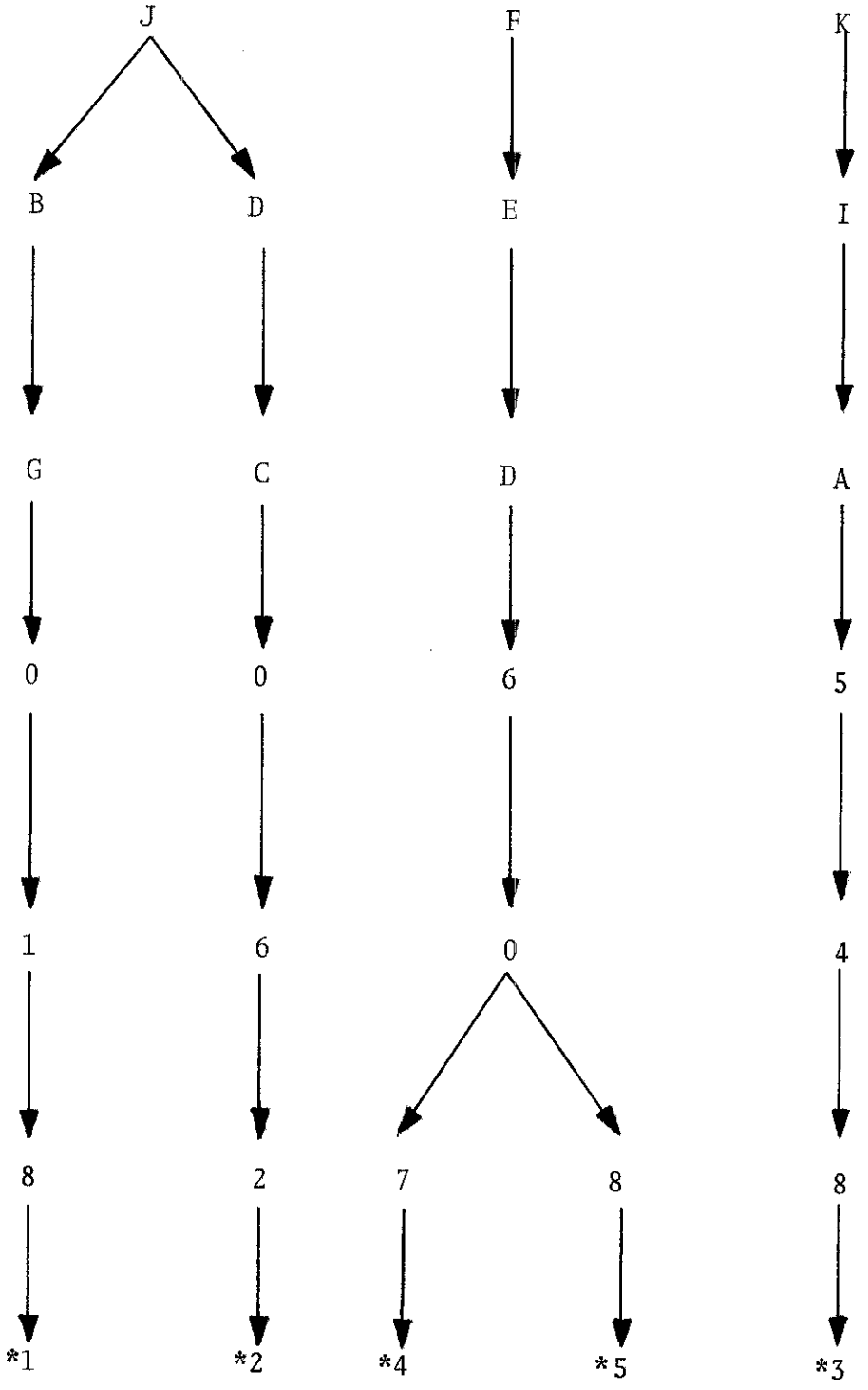
In this discussion, it is assumed that the tree structures under consideration are being used for key-to-address transformation that can be represented in a computer--a series of mixed letters and digits, binary integers, or some other quantity. Since decoding is an application of tree searching, which

is the fundamental operation performed in any tree-structured file, this discussion is also applicable to any tree-structured file.

Many names have been used for various tree structures; no standard terminology exists. For this discussion, trees are classified into three types: (1) symbol trees, (2) immediate decoding trees, and (3) directory trees. These three categories provide a reasonable basis for a survey of tree organizations. However, the following discussion by no means exhausts all the possible combinations of node contents and linkage arrangements that can be made.

Symbol Trees

In the construction of a symbol tree, the leaves of the tree are placed in one-to-one correspondence to the addresses in the file that is indexed by the tree. The non-leaf nodes of the tree are used only for searching; each leaf node of the tree contains a file address. Each key is broken into a number of symbols, and one node is used for each symbol. The tree will have one node on the first level for each distinct first symbol of a key; one node on the second level for each distinct first symbol of a key; one node on the second level for each distinct second symbol; and so on (11). The filial set for each symbol will have one node for each distinct following symbol in the key set. Figure 9 shows a symbol tree for the example file for searching by license number.



* Record number

Figure 9. Example File--1 Symbol Trees

Immediate Decoding Trees

With a symbol tree, decoding is never complete until a leaf is reached. There is another type of tree that has appeared in the literature (1), called in this discussion an immediate decoding tree, in which an entire key is stored at each node, and decoding can be completed without reaching a leaf. In this structure, one node is used for each distinct key value. The tree is searched until an exact match is found, and then the file address in that node is taken as the result of decoding. An immediate decoding tree for keys which take the integral values from one to fifteen is shown in Figure 10. This structure is particularly suitable for relatively short keys that can be represented in one or two machine words. For long keys, particularly if they share many initial polygrams, a symbol tree makes more efficient use of storage.

When an immediate decoding tree is searched, the search key is compared with the key at the root. If the two keys are equal, the file address stored at the root is the decoded address. If the search key is less than the key at the root, the left branch is taken and the previous step is repeated. If the search key is greater than the key at the root, the right branch is taken and the previous step is repeated. If a leaf is reached without an equal comparison, the search terminates unsuccessfully.

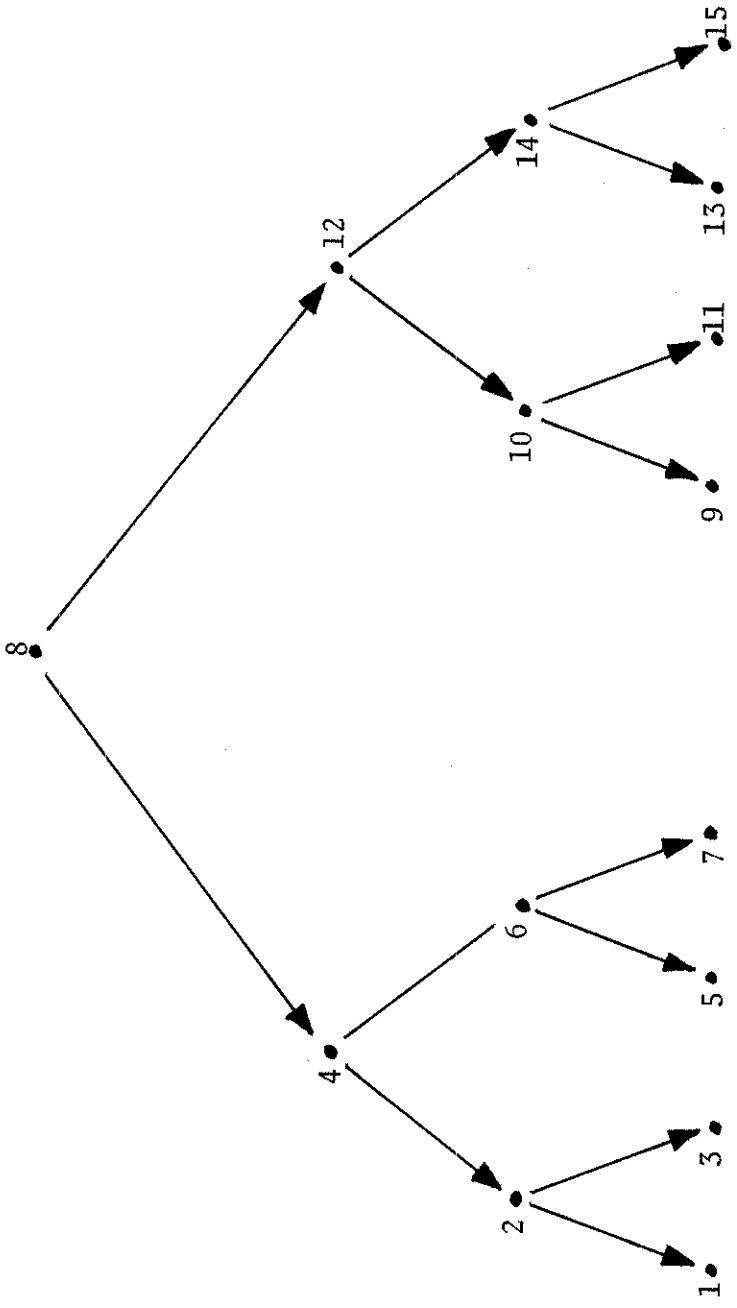


Figure 10. Immediate Decoding Tree

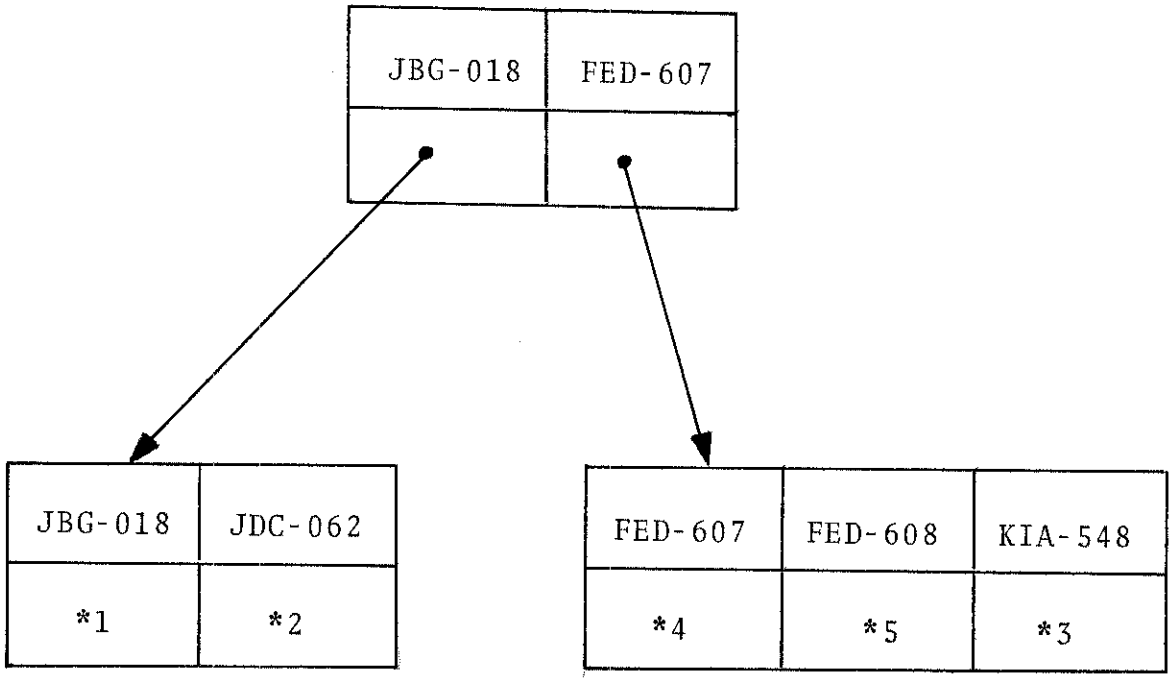
Directory Trees

A directory tree lies in the middle ground between a symbol tree and an immediate decoding tree. Each node of a directory tree contains several entire keys, but no file addresses. During the search, if the search key is greater than the jth key at a node but less than the (j + key), then the branch corresponding to the jth node is tested next. When a leaf is reached, that leaf will contain the desired file address, if the search key is in the file. Figure 11 shows a directory tree for searching the example file by license number.

A directory tree is most useful if the file keys are all the same length, or can be compressed to the same length. In that situation, the keys stored at each node can be searched using a binary search, greatly speeding the search process.

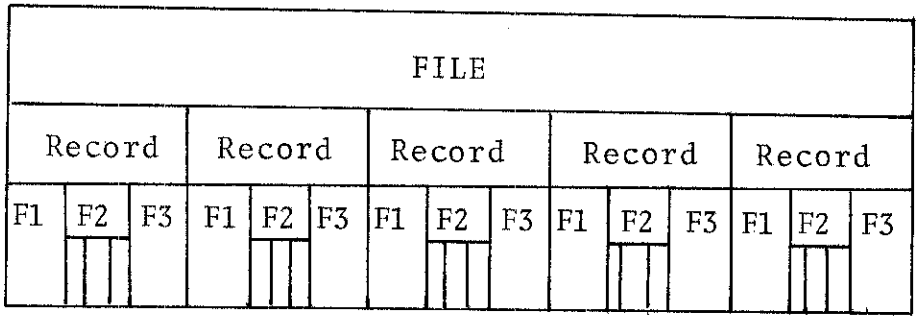
Indexed Sequential Organization

The indexed sequential file organization follows the same basic pattern of sequential organization previously described. For convenience, however, in permitting access by attribute when the attribute is the part of the key used for ordering the file, the indexed sequential adds an index to the keys. As indicated in Figure 12, this adds to the data structure a separate new portion showing an address for each of the cited keys. The index when supported by the hardware reduces the long sequential access time "to pass the file" and thus permits



*Record number

Figure 11. Example File--1 Directory Tree



Example File--2

(A file consisting of five file components [records] each with three fields, one of which has three subfields).

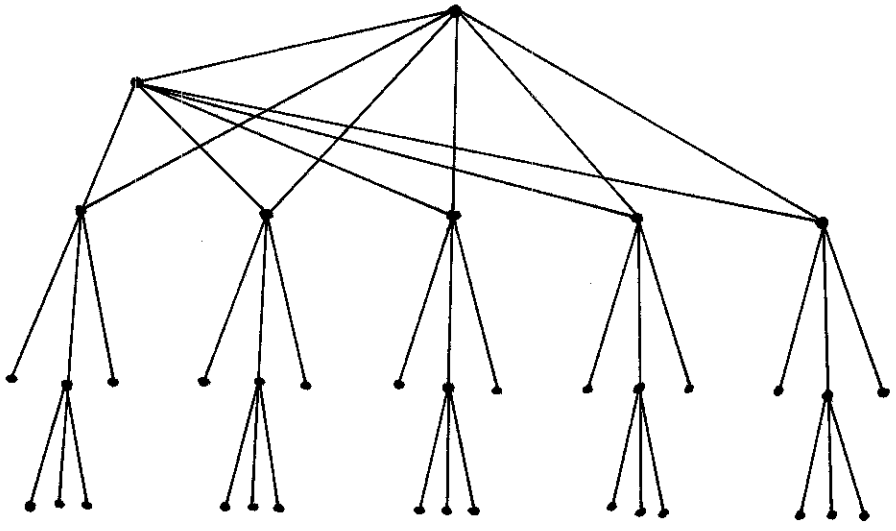


Figure 12. Indexed Sequential Structure of Example File--2

access to any file component given the key, in about the same length of time ("permits random access") (4, p. 276).

Indexed sequential organization, however, by reference to indexes associated with the file, makes it also possible to quickly locate individual records for non-sequential processing. Moreover, a separate area of the file is set aside for additions; this obviates a rewrite of the entire file, a process that would usually be necessary when adding records to a sequential file. Although the added records are not physically in key sequence, the indexes are referred to in order to retrieve the added records in key sequence, thus making rapid sequential processing possible.

The principal use of this file organization is for the random access of records with nonserial keys (8, p. 56). There must be a single key designated within each record for file positioning, but the value of this key need not necessarily be unique; if it is not, then the access to the file will be to the first record in the series containing this key, and the other records containing the same key will appear immediately afterward. The key values themselves do not have to be serial since the access mechanism to the records does not use the key itself as a locator, but rather relies upon an index or a look-up in order to point to a particular block containing the indicated record key; hence, the key may be alphanumeric and of varying lengths.

Attributed String Organization

The attributed string file orgnization consists of a string of properties with both elements of the organization explicitly stated. The elements of the organization can be termed as the attributes, and the string properties. Here the usual concept of a record may be maintained as the file components, or the properties themselves may be the file components or any grouping of them. This technique of file organization might be more properly called the "property string" since it consists of a sequence of properties (3). However, since the major difference between this and the common sequential file organization is the presence of the attribute in explicit form, the term attributed string is for historical contrast.

An example of an attributed string file organization is in the intelligence area (4, pp. 279-280). The items of data gathered by an intelligence operation and maintained in an intelligence file typically show very little uniformity of conceptual content. The attributed string, therefore, appears as an attractive way of showing such varieties of conceptual content since it explicitly designates both attributes and values, in contrast to the sequential or indexed sequential file organizations.

The prime cost of the attributed string is the storage space taken by the attributes. For this reason, the attributed string organization finds application mostly where the

sequential file organization would result in much storage space occupied by filler symbols. Attributed strings may be ordered but commonly are unordered. A diagram illustrating the character of an attributed string organized file is shown in Figure 13.

Access by attribute, by property, and by value are all equally convenient, but typically slow. If the file be ordered on an attribute, property, or value, then the file becomes essentially an indexed sequential organized file with the characteristics noted previously.

In summary, the attributed string organization, because of its use of storage space, serves as an alternative to the sequential or indexed sequential file organizations for the components of a file where the variability in the conceptual content is such that the space occupied by the attributes is less than the space that would be occupied by fillers in the sequential or indexed sequential file organizations. It provides further the most equal ease of access by attribute, by property, and by value.

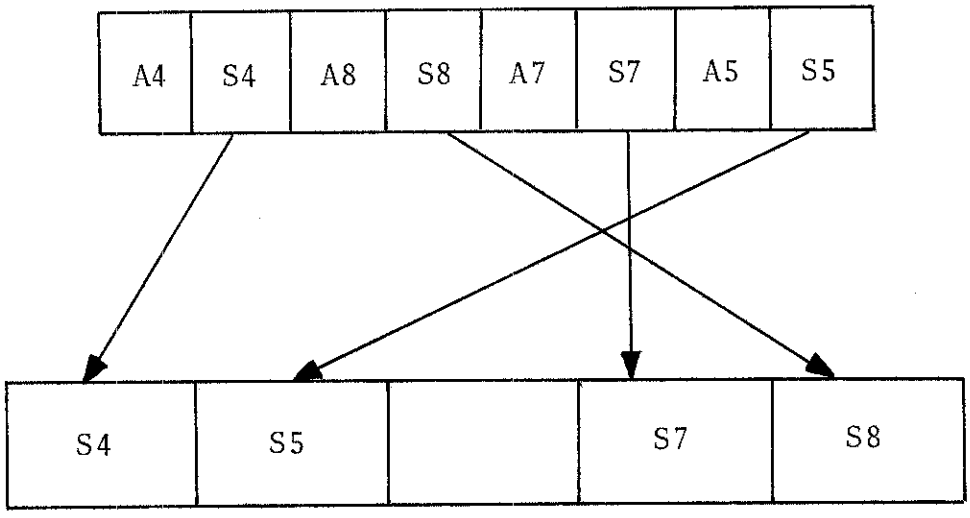


Figure 13. Diagram of the Attributed String Organization

CHAPTER BIBLIOGRAPHY

1. Arora, S. R. and W. T. Dent, "Randomized Binary Search Technique," Communications of the ACM, XII (February, 1969), 77-80.

2. Bourne, C.P. and D. F. Ford, "A Study of Methods for Systematically Abbreviating English Words and Names," Journal of the ACM, VIII, (April, 1961), 538-552.

3. Chapin, Ned, "A Deeper Look at Data," Proceedings of the ACM National Conference 68, (1968), 631-638.

4. Chapin, Ned, "A Comparison of File Organization Techniques," Proceedings of the ACM National Conference 69, (1969), 273-283.

5. Davidson, L., "Retrieval of Misspelled Names in an Airlines Passenger Record System," Communications of the ACM V, (March, 1962), 169-171.

6. Kindred, Alton R., Data Systems and Management, Englewood Cliffs, N. J., Prentice-Hall, Inc., 1973, 129-130.

7. Knuth, D. E., Fundamental Algorithms: The Art of Computer Programming, Vol. 1, Reading, Massachusetts, Addison-Wesley, 1968.

8. Lefkovitz, David, Data Management for On-Line Systems, Rochelle Park, N. J., Hayden Book Co., Inc., 1974, 56-57.

9. Morris, R., "Scatter Storage Techniques," Communications of the ACM, XI, (January, 1968), 38-44.

10. Roberts, David C., "File Organization Techniques," Advances in Computers, edited by Morris Rubinoff, New York, New York, Academic Press, 1972, 115-165.

11. Sussenguth, E. H., Jr., "Use of Tree Structures for Processing Files," Communications of the ACM, VI, (May, 1963), 273-279.

CHAPTER III

FUNCTIONAL REQUIREMENTS FOR FILE ORGANIZATIONS

This chapter is devoted to certain definitions, underlying concepts, and the functional requirements that are placed upon the design of files for online systems by virtue of their incorporation into a system. One requirement for many applications is the ability to process data as it becomes available. Records in a file must be logically organized so that they can be retrieved efficiently for processing. This chapter looks at what the characteristics of a file organization should be in regards to online systems, and the user demands for file organizations.

Characteristics of File Organizations

In data processing today, it is common for a computer installation to do a number of different types of processing. An installation must provide for one combination or another of data base processing, online processing, batch processing, and inquiry and transaction processing. The inherent characteristics of the file must be considered in selecting an efficient method of organization. In addition to the purpose for which a file is to be used, there are other characteristics to be considered in planning both the manner in which the file is to be organized and the manner in which it is to be accessed (3).

Among these are volatility, activity, and size. Thus, there are a variety of requirements an access method must provide. The following are some requirements that need to be met when choosing a particular file organization.

a. Volatility: This refers to the addition and deletion of records from a file. A static file is one that has a low percentage of additions and deletions. No matter how the file is organized, additions and deletions are of significant concern, but they can be handled more efficiently with some methods of organization than with others.

b. Activity: Activity refers to the number of records actually processed or updated during a single run through the entire file. Activity is considered from three standpoints: (1) percentage, (2) distribution, and (3) amount.
The percentage of activity is one of the factors to be considered. If a low percentage of the records are to be processed on a run, the file should probably be organized in such a way that any record can be quickly located without having to look at all the records in the file.
The distribution of the activity is also a consideration. With some methods of organization,

some records can be located more quickly than others. The records processed most frequently should certainly be the ones that can be located most quickly. The amount of activity also makes a difference. An active file (that is, one which is frequently referred to) must be organized very carefully, since the time involved in locating records may amount to an appreciable period of time. At the other extreme, an active file may be referred to so infrequently that the time required to locate records is immaterial.

c. Size: A file so large that it cannot all be online (available to the system) at one time must be organized and processed in certain ways. A file may be so small that the method of organization makes little difference, since the time required to process it is very short, no matter how it is organized. The growth potential of the file is also a consideration. Usually, files are planned on the basis of their anticipated growth over a period of time. Initial planning must also consider how growth that exceeds this size will eventually be handled.

User Demands for File Organizations

The common user demands on files considered here fall into four categories: naturalness of the file organization

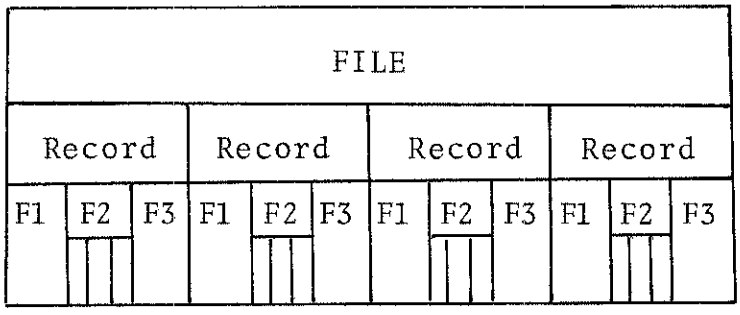
to the data, ease of access to data in the file, ease of maintenance of the file, and extent of software and hardware support for the file organization.

Naturalness

The naturalness of a file organization to the data is partly a subjective matter dependent upon human familiarity and ease of understanding. People tend to prefer file organization techniques which they believe they understand and with which they have had experience.

From their experiences with file folders, filing cabinets, and alphabetizing, people find it comfortable to regard a file as composed of records, usually ordered (sequenced) on the basis of some key. This is illustrated graphically in Figure 14. Each record is composed of data items, some of which may in turn be deemed to be composed of other data items. This is essentially a tree structure of "is composed of" relationships among properties. People's experience with this common way of organizing data outside of an automatic computer in turn often leads them to conceive of data structures for use with a computer in terms that reflect the file organization techniques they know.

Since a file is a collection of symbols that people consider to be grouped into file components (usually "records"), the distinction between one component and another and within a component in a file depends upon a human-imposed structure.



Example File--3

(A file consisting of four file components [records] each with three fields, one of which has three subfields).

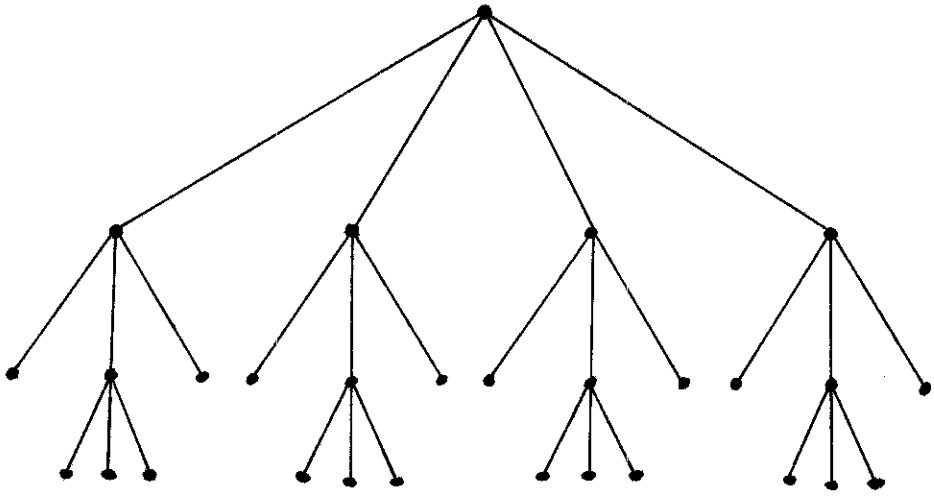


Figure 14. Vertical Structure of Example File--3

The more easily the people can form such concepts, the less difficulty people feel they experience in working with the data, and the more "natural" they feel the file organization to be.

Access

Users commonly demand access to a file in one of three manners (1). The first of these is access by attribute, usually to find the value (2). Commonly, this access involves a logical ANDing of attributes. An example is to find in the files the amounts of the balances due from customers. Historically, making provision for this manner of access has dominated the selection of file organization techniques.

A second manner of access is by property, usually to find other properties, or values. The access utilizes both the attirubte and the value in the access process, and commonly involves logical ANDing. An example of access by property is to find in the file the balance due from the customer named XYZ Corporation.

A third manner of access is by value to find the attribute. This is the least common of the ways of access and in one form is equivalent to content addressing. In practice, it is nearly always followed by additional access by attribute or by property. An example of access by value is to list all the students in a file with a grade point average of 3.5 or greater.

Maintenance

Users universally demand the ability to change the content of a file--i.e., to be able to maintain or update a file. But for a user to perform maintenance on a file requires that he have the ability to perform access on one or more means just described. File maintenance consists fundamentally of three operations: augmenting the data in files--that is, incorporating additional properties in the file; altering the existing contents of the file--that is, changing the values in some properties but not the attributes; and deleting data from the file--that is, removing existing properties from the file.

The augment and delete operations involve changing at a conceptual level the quantity of the data in the file. That is, augment conceptually increases the amount of data in the file. The file after augmentation consists of more non-filler symbols such as non-blanks or non-zeros than it did before augmentation. Conceptually, deletion from a file results in a decrease in the amount of data in the file. The file after deletion consists of fewer non-filler symbols than it did before.

These operations are conceptually defined since physically, the amount of storage space occupied by the symbols included in a file in any given implementation may not be altered by some deletion or augmentation operations. For example,

consider the case of a file conceptually organized into fixed length records of identical format. If the user desires to delete one or more of these records, one way of doing it is to overwrite into the records to be deleted a delete symbol, replacing some previously existing symbol, such as part of the key. The result leaves unchanged the total number of symbols in the file from a physical point of view, but conceptually the file has been decreased in size.

Support

Even though the choice of file organization technique should depend heavily upon the access and the maintenance that the user desires to perform, in practice it more often depends upon the hardware and software support. Users often must select file organization techniques based upon what will work with the software and hardware they have available.

The only question of efficiency that historically has been seriously considered in detail has been the use of storage space. Since historically internal storage has been very limited, people have tried to minimize the amount of internal storage needed for storing data records. In the past few years, with the wider availability of large amounts of internal storage, the imperative need for this requirement has begun to fade. But internal storage, while now available, is still costly, and hence in practice people still conserve its use.

Closely related to this has been the historically limited capacity of external storage units, and their comparative slow transfer rates. These limits were eased by hardware developments in the early and mid-1960's but the practice of seeking file organization techniques that minimize the use of external storage still persist because external storage too is not a free good.

Historically, matters of operating speed have taken a back seat to matters of storage use because the slower speed could be more easily tolerated than the lack of sufficient storage space. With the gradual easing of the storage limitations, people now are starting to pay more attention to ways of speeding operations involving files.

Because of the demand to keep the trauma of file conversions small and because of the intellectual heritage of the older limitations, the software generally available for file operations has not kept up with the advances in hardware. Yet a user when faced with the cost of developing his own software to support a particular file organization technique, often judges that he loses less in time and money by electing to use what is available, even if it meets his needs only roughly.

CHAPTER BIBLIOGRAPHY

1. Chapin, Ned, "A Deeper Look At Data," Proceedings of the ACM National Conference 68, (1968), 631-638.

2. McGee, William C., "File Structures for Generalized Data Management," Proceedings of the IFIP Congress 68, (1968), 68-73.

3. Introduction to IBM Direct-Access Storage Devices and Organization Methods, Form GC20-1649-9, IBM Corp., White Plains, New York, 1976.

CHAPTER IV

FILE ORGANIZATION FOR LARGE AND SMALL SCALE COMPUTER SYSTEMS

The manner in which data is managed in a computer system and in a computer installation determines the degree to which user needs can be satisfied and governs the efficiency of file structures. Input/output and external storage devices are the limiting factor on the efficiency of most information-based file systems, even though devices such as the channel, and operational techniques such as multiprogramming, are commonly used to increase overall computer system performance. Therefore, the manner in which input and output are performed, the methods used for data storage and organization, and the techniques employed to access data are of prime importance to most computer system users.

A basic knowledge of file structures is essential for an understanding of data management processes in a computer system. Thus, the following discussions are not exhaustive but do provide important terminology to be used in the comparison of file structures imposed by the data management facilities of two available computer systems.

This chapter describes the file organization techniques imposed by the data management facilities provided by the

operating system of a large scale computer system, and a small scale computer system. The first computer system to be studied is the IBM System/370 (OS/370) which is a logical outgrowth of and an extension of the IBM System/360 (OS/360) computer (7). The second computer system to be studied in this comparison is the Harris 1600 Distributed Processing System (2) with the Extended Communications Operating System (ECOS) serving as its operating system (3). Table II shows how certain system characteristics from each system depict the system's capability and size.

File Organization and Access for A Large Scale Computer System

When a file is implemented, the implementation must be performed within the constraints imposed by an available computer system. These constraints are in two areas: physical constraints arising from the hardware characteristics of the computer and its peripherals, and software constraints imposed by the data management facilities provided by the operating system. This section discusses the latter constraints as imposed by the data management facilities of IBM System/370 (OS/370).

File Organization

Data files are organized according to the manner in which they are used. Five methods of organization are identified for the IBM System/370 (OS/370):

1. Sequential
2. Direct
3. Indexed Sequential

TABLE II

COMPARISON CHART

Characteristics	Large Scale Computer System (IBM System/370 Model 135)	Small Scale Computer System (Harris 1600 RCP*)
Main Storage	245,760 bytes (240K)	65,535 bytes (65K)
Cycle Time	275 to 1430 nanoseconds	1 microsecond to 750 nanoseconds
Disk Capacity	800 megabytes (3330 Disk Storage Facility)	12 megabytes (Harris 1665 Cartridge Disk Drive)
Disk Access Time	30 milliseconds (average)	50 milliseconds (average)
Floppy Disk	Yes	Yes

*Remote Communications Processor

4. Partitioned.

5. Virtual Storage Access Method (VSAM).

Sequential Organization--Sequential organization denotes that the data records must be referenced in a manner dependent upon the sequence in which the data records are physically stored. Card decks and magnetic tape files are always organized sequentially and files on direct-access storage devices are frequently, but not necessarily, organized in this manner.

This type of data organization is characterized by the fact that records are written and retrieved in physical rather than logical sequence (7, p. 81). One of the major advantages of sequential organization is that the access mechanism is automatically positioned to access the next record. Operational difficulties are encountered with sequential organization when records must be altered, deleted, or inserted. When sequential media is used, the file must be copied to another volume with the modifications being made as required. With direct-access media, records can be altered provided that the altered record is not shorter or longer than the original. Performing update operations on files with blocked records is also a problem, since the entire block must be rewritten.

A sequential organized file is a good system for handling a large volume of data that rarely changes and is processed as an entire unit (8, p. 95). The records are usually read

or updated in the same order in which they appear. For example, the hundredth record is usually read only after the first ninety-nine records have been read. This organization is generally used when most records are processed each time the file is used.

Direct Organization--A file organized in a direct manner is characterized by some predictable relationship between the key of a record and the address of that record on a direct-access storage device (5, p. 5-4). This relationship is established by the user. Direct organization denotes that a data record may be referenced without "passing over" or referencing preceding information (6, p. 95). This organization method is generally used for files whose characteristics do not permit the use of sequential or indexed sequential organizations, or for files where the time required to locate individual records must be kept to an absolute minimum.

Direct organization has considerable flexibility. The accompanying disadvantage is that although the programming system provides the routines to read and write a file of this type, the user is largely responsible for the logic and programming required to locate records, since he establishes the relationship between the key of the record and its address on the direct-access storage device.

Indexed Sequential Organization--Indexed sequential organization permits data records in a file to be referenced

sequentially or directly. The defining characteristic of an indexed sequential file is that data records are logically arranged by collating sequence, according to a key field contained in each record (6, pp. 95, 116). Indexes of keys are maintained to provide direct or sequential access. When an indexed sequential file is referenced sequentially, the keys in an index are processed in order; associated with each key is the physical location on a direct-access storage device of the corresponding data record. When an indexed sequential file is referenced directly, the key is looked up in the index (analogous to a table) to determine the physical location of the desired record. Data files with indexed sequential organization must reside on a direct-access storage device.

An indexed sequential file is similar to a sequential file in that rapid sequential processing is possible. Indexed sequential organization, however, by reference to indexes associated with the file, makes it also possible to quickly locate individual records for non-sequential processing (5). Moreover, a separate area of the file is set aside for additions; this obviates a rewrite of the entire file, a process that would usually be necessary when adding records to a sequential file. Although the added records are not physically in key sequence, the indexes are referred to in order to retrieve the added records in key sequence, thus making rapid sequential processing possible.

The indexed sequential method of organization is of special concern, since it allows the file to be processed directly or sequentially. Records can be inserted or deleted, and basic and queued access can be used.

An indexed sequential file must reside on a direct-access storage device. Just like any other file, it possesses a file control block stored in the volume table of contents (VTOC) for that direct-access storage device. Each indexed sequential file can use three different storage areas.

1. The prime area contains data records and track indexes. The prime area is always used.
2. The overflow area contains overflow from the prime area when new records are added to the file. Use of the overflow area is optional.
3. The index area contains master and cylinder indexes for the file and is used when the file occupies more than one cylinder.

The access to records is managed through indexes. When records are written in the prime area, the system keeps record of the highest key (i.e., the last record) for each track and forms a track index--one entry per track. There is a track index for each cylinder of the file. If the file occupies more than one cylinder, then a cylinder index exists for each cylinder; each entry in the cylinder index reflects the key of the last record in the cylinder. A master index is developed for groups of cylinders to increase the speed of searching

the cylinder index. An indexed sequential file is depicted conceptually in Figure 15.

The track index contains one entry for each track of a cylinder. Each track index entry includes a normal entry and an overflow entry. Initially, the normal and overflow entries are the same. When an overflow is associated with a track (due to an insert operation), the overflow entry contains the key to the highest overflow record and the address of the lowest overflow record associated with that track. A track index is generated automatically for each cylinder.

As each track index is generated, the data management system creates a cylinder index entry. If more than one cylinder index entry is created, then a cylinder index is formed. The master index is usually optional; the user can specify the number of tracks of the cylinder index that corresponds to the master index entry.

Partitioned organization--Partitioned organization denotes a data file that is divided into sequentially organized members. Each member is composed of data records. Logically, a partitioned data file is a file of files (6, p. 96). Each file or member is assigned a name (such as the name of a program) and each name is stored in a directory, along with the physical location of the beginning of the member. The directory is stored along with the file. Members may be called by name for processing, and members may be added or deleted as

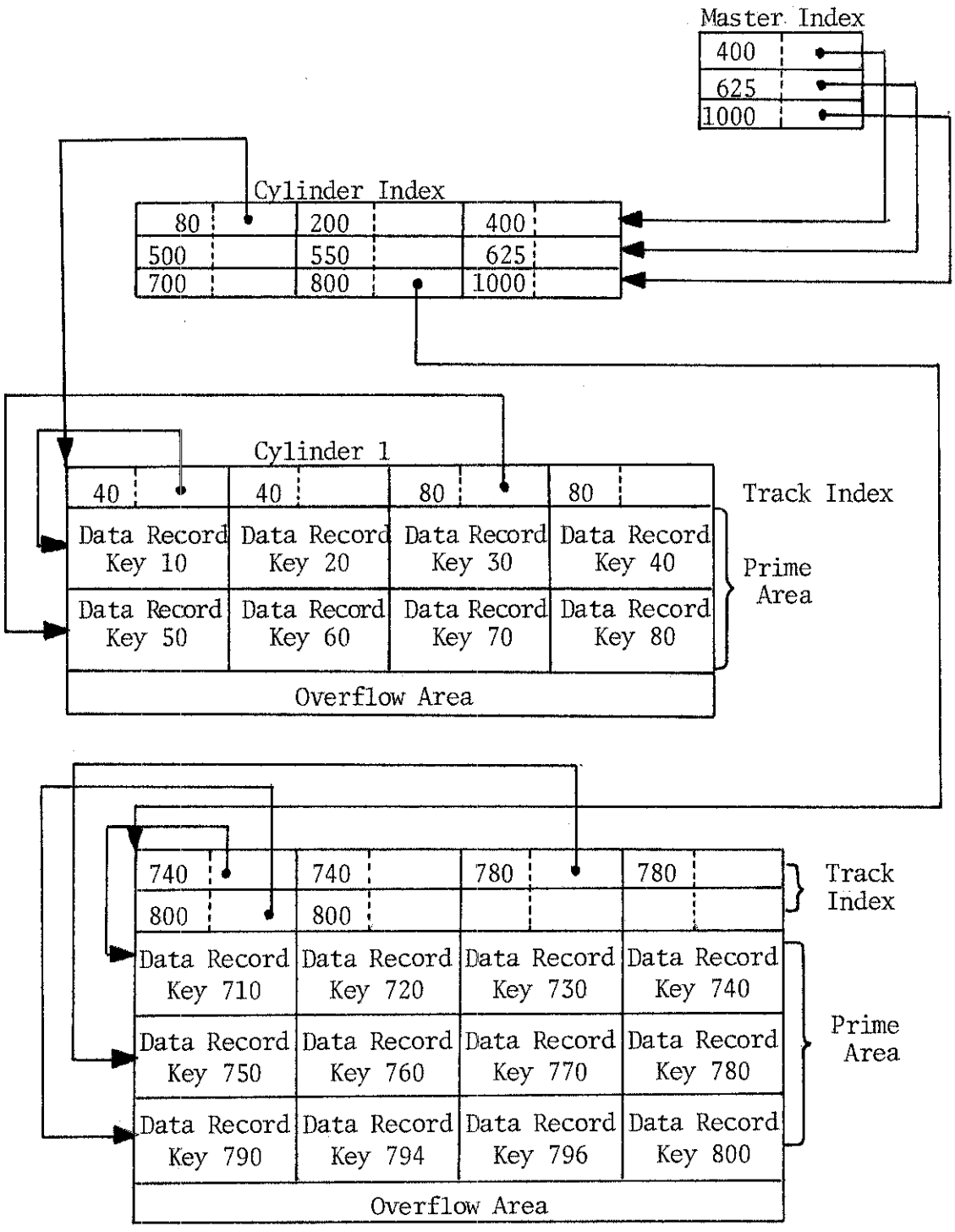


Figure 15. Conceptual View of Indexed Sequential File Organization

required. The records within the members are organized sequentially and are retrieved or stored successively according to physical sequence (5).

Partitioned organization is used mainly for the storage of sequential data, such as programs, subroutines, and tables. For example, a library of subroutines may be a partitioned file whose members are the subroutines.

The IBM System/370 (OS/370) provides the user with the subroutines necessary to create and maintain his own partitioned file. OS/370 also uses partitioned files to maintain its own libraries (9, pp. 385-387).

Virtual storage access method (VSAM) organization--The data organization for virtual storage access method differs from the preceding organizations so as to establish a data organization that will be, from a user's point of view, device independent. The data organization should be suitable for all kinds of accessing (indexed, addressed, direct and sequential) and should be extendable to anticipated requirements (5, p. 5-4). Once a user adopts the VSAM organization, his data will be portable from an IBM System/370 to another IBM System/370. This will facilitate migration from smaller systems to larger systems.

Data records of fixed or variable length are stored in the same format in both indexed-sequenced and entry-sequenced files. The records of an indexed-sequenced file are in collating

sequence, defined by a key field in the records; the records of an entry-sequenced file are in the same sequence as the order in which they are entered in the file. An index is used to physically locate and sequentially order the records of an indexed sequential file.

VSAM is designed to meet most of the common data organization needs of both batch and online processing (5, p. 10-3). Batch processing requires the efficiency of sequential and indexed data; online processing requires efficient direct access for random requests. VSAM permits both direct and sequential access. Access can be by key, by relative address, or by relative record number. Different types of processing can be intermixed in the processing of a common data base. You can select the type of access or the combination of types that best suits user applications.

Access Methods

The characteristics of a file as described in certain control blocks determine the access method that is assigned to a file. An access method is the combination of a file organization and the technique used to access the data (6, p. 112).

Two different techniques are normally available to the user for transferring data between storage and an external storage medium. These techniques are referred to as access methods and are implemented as access routines supplied by the data management subsystem of the operating system (OS/370)

(6, p. 96). The access routines are available through macro instructions recognized by the IBM System/370 assembler system.

The access techniques are combined with the file organization to determine access methods. The basic access technique is used with all forms of file organization; queued access is used only with sequential and indexed sequential organization. Six access methods are given in the following list and summarized in Table III:

1. Basic Sequential Access Method (BSAM)
2. Basic Partitioned Access Method (BPAM)
3. Basic Indexed Sequential Access Method (BISAM)
4. Basic Direct Access Method (BDAM)
5. Queued Sequential Access Method (QSAM)
6. Queued Indexed Sequential Access Method (QISAM)

The processing of each of the above access methods is performed by a distinct access method routine; however, all users of the data management system use the same routines (6, p. 112).

TABLE III

ACCESS METHODS

File Organization	Data Access Technique	
	Basic	Queued
Sequential	BSAM	QSAM
Partitioned	BPAM	
Indexed Sequential	BISAM	QISAM
Direct	BDAM	

The access techniques are classified by their treatment of buffering and input/output synchronization with processing. The two techniques are named queued access and basic access.

Queued access--The queued access technique provides automatic blocking and deblocking on data transfers between main storage and input/output devices. Queued access also provides look-ahead buffering and automatic synchronization of input/output operation and processing. The access method routines controls the use of buffers such that sufficient input blocks are in storage at one time, preventing the delay of processing unit operation. When using queued access, the user need not test for input/output completion, errors, or exceptional conditions. After the completion of an input/output macro instruction, control is not returned to the user application program until the operation is logically complete.

When queued access is used, buffering achieves its greatest utility. Two transmittal modes are covered.

1. Move mode--the record is moved to the application program's work area or from the work are to an output buffer.
2. Locate mode--the record is not moved but the address of the buffer holding the input record is placed in a general-purpose register (for input) or the address of the next output buffer is placed in a general-purpose register (for output).

The move and locate modes are referred to as simply buffering, because the buffers in a buffer pool are associated with a single file (6, p. 114).

Basic access--The basic access technique does not provide automatic blocking and deblocking; neither does it provide anticipatory buffering or automatic event synchronization. Basic access is used when the sequence in which records are processed cannot be predicted in advance. With the basic access technique, the user must perform his own blocking and deblocking. Moreover, input and output macro instructions only initiate input/output processing; both operations must be checked for completion with an appropriate macro instruction. In other words, automatic event synchronication is not provided.

When basic access is used, buffers are controlled and used in two ways.

1. A buffer is obtained directly from a buffer pool with a macro instruction. That buffer is subsequently used with an input/output operation.
2. A buffer is obtained dynamically by requesting a buffer with the input/output macro instruction.

File Organization and Access for A Small Scale Computer System

In order to understand what a small scale computer system is all about, it is necessary to understand the system components and their organization. This section is concerned with the general job of describing the file structure organization and access within a small scale system. The small scale system to be described is the Harris 1600 Distributed Processing System (2), with the Extended Communications Operating System (ECOS) serving as the operating system (3).

File Organization

In the small scale computer system the file organization is not usually left up to the user but is predefined for the various peripheral devices. Similarly, the method of access is system-defined and is ultimately connected with the file organization. Each file-oriented peripheral device has a file structure, which represents the method of recording, linking, and cataloging data files. The file structure dictates the organization of the file on the device and the method of file access. This organizational structuring is important because a file can be effective for a user application only if it is designed to meet specific user requirements. Such factors as size, activity, and accessibility must be considered when determining the structure of a file (1). Thus, the types of file organizations are restricted to just a few. We will discuss them. They are

1. Sequential
2. Relative Record
3. Keyed Sequential
4. Partitioned.

Logical sequential organization--A logical sequential organized file consists of a data area and a unit of allocation (UA) area. The data area is made up of UAs assigned to the file. Records in the data area are of variable length and may span over sector and UA boundaries. Each record contains a header consisting of forward and backward chain pointers, length of the record, and a flag indicator. The logical order of records in a file is maintained through chain pointers in the record headers. The chain pointers are not absolute disk addresses, but are used in conjunction with a starting address for the file or with the UA descriptor area to arrive at absolute disk addresses. The physical sequence of records is always in chronological order (records are always added to the end of the file, even if they are inserted records). When a record is inserted, the chain pointers involved are updated. A deleted record is flagged as deleted but remains logically attached to the file, and the chain pointers are updated only after an inserted record is added.

The UA descriptor area consists of sector-size UA directory blocks. It is created after the initial contiguous allocation is used up. As new UAs are assigned to the file,

the addresses of the UAs are entered in the UA descriptor area. A directory block may contain up to 121 entries describing UAs. When additional entries are needed, a new UA directory block (a sector) is chained to the existing one(s).

Relative record organization--A relative record organized file consists of a data area and a UA descriptor area. The UA descriptor area is identical to the one used in logical sequential files. The records are addressed explicitly by a relative record number or implicitly by addressing the next sequential record.

The internal organization of a relative record organized file is the same as that of a logical sequential file, with the following restrictions:

1. There is no initial dummy record; the first record starts in the first byte of the data area.
2. All records in the file must have the same length.
3. Logical record sequence matches physical record sequence; this is enforced by not allowing deletion or insertion of records in existing files.

These three restrictions make possible the calculation of the relative position of any record.

Keyed sequential organization--Each record in a Keyed Sequential file contains a key by which the record is identified. Records in a file are logically arranged according to ascending collating sequence of the key field in each record.

The key field is part of the data in the record. The length and the position of the key field in a record is constant within a file but may vary from file to file.

Keyed sequential files use logical sequential organization. That is, they contain a data area and a UA descriptor area. In addition, a key directory area is kept with the file. The key directory is a partial index of keys in the file. The frequency of entries in the directory is specified when the keyed sequential file is created (an entry in the directory for every so many records). An entry in the directory consists of a key and location of the associated record. Space for the key directory is allocated in Units of Allocation (UAs). As extra space is needed for the directory, additional UAs are chained to the existing one(s). Entries in the directory are of fixed length and may span over sector and Unit of Allocation (UA) boundaries.

Partitioned organization--A partitioned file is divided into sequentially organized members, each consisting of one or more records. Each member is identified by a unique name, up to 8 characters long. A partitioned file is organized as a logical sequential file. In addition, a directory is kept containing member names and their starting locations. In the data portion a special record, the start of member record, is maintained. This record is used as a member separator when reading the file sequentially, or is used to rebuild the member

directory during recovery. The member separator record contains the name of the member which follows the record.

Access Methods

The file organization and associated file structure of the Harris 1600 Distributed Processing System (ECOS) has been previously described from a user viewpoint. Because of its complexity, the operating system (ECOS) provides data management facilities and access methods which handle the file structure changes that can result from the insertion and deletion of records.

The data management facilities in the small scale system is concerned with the movement of data between main storage and external devices, and with the maintenance of data on direct-access devices (disks). The Harris 1600 Distributed Processing System (ECOS) provides three functions of data management facilities.

1. Basic (physical) input/output via direct interface to the operating system.
2. Queued (logical) input/output via the file management services.
3. Logical file manipulations via the file system utility programs.

The following discussion describes those aspects of file processing which are performed by the small scale system access methods.

Logical sequential access method (LSAM)--The logical sequential access method allows a user to process files which are organized as logical sequential, partitioned or keyed sequential. The sequence of records in files is determined by the user at the time of entry of the record in the file (i.e., the logical sequence of records in the file does not necessarily correspond to the chronological sequence of entry of records in the file). Data records are stored as variable-length records on disk; however, processing may be limited to accept and/or produce only records of a fixed length.

For user convenience, macro instructions are provided to record or change the current-position pointer. Certain accesses cause an implicit change in the current-position pointer after performance of the requested operation. New records may be added to the file in one of two ways. The first way is via the insertion of records between two existing records (the end of file and start of file are considered to be records). This way has a tendency to cause inefficiencies in placement of the records on disk and also requires more accesses to the disk for each record than the second way. The second way in which records may be entered in a file is via sequential output. This method places the records in the file in the order in which they are provided by the user and moves the end of file (conceptually) with the entry of each new record. When the file is closed, the end of file is recorded after existing buffers are purged.

As an example of the logical sequential access method (LSAM) within the small scale system, a file system performance statistics is used to characterize the performance of LSAM. The logical sequential access method (LSAM) is the most numerous used access method in the Harris Distributed Processing System (ECOS). The statistics exemplify the GET-PUT times of LSAM disk accesses, allocation, and transfer rates when copying cards to disk as a function of the buffering level (see Appendix II). This example indicates that even with the small scale system file access is not always slow.

Relative record access method (RRAM)--The relative record access method allows a user to process files organized as relative record files. The records are fixed length. The concept of a record is essentially an addressable area on the disk. The concept of a current-position pointer is used. The record to be accessed is determined by the value of the current-position pointer and the type of access desired. Certain accesses also cause a change (incrementally forward by one record position) in the current-position pointer. New records (or areas on the disk) are created as a part of a relative record file by PUT macros issued when the processing type is output and the addressing mode is sequential. In any other case, only existing records may be accessed.

Since the records are fixed length, the relative position of any record in a relative record file may be calculated

(instead of searched for), thus allowing rapid access to the file without a sequential search through each record. Access to the file is further optimized if the file is allocated contiguously on the disk, thus obviating the necessity to perform a directory access to determine the physical disk address based on the relative address.

Keyed sequential access method (KSAM)--The keyed sequential access method allows the user to process files which are organized as keyed sequential files. Since keyed sequential files are basically the same as logical sequential files, KSAM allows the user virtually the same sequential processing capabilities as does LSAM. The major difference in sequential processing is that the current-position pointer value may be reset based on a key contained in a record (i.e., set the pointer to the record containing the specified key).

The user may also access a file by providing an explicit address for each record processed. The explicit address is in the form of a sequence of characters (up to 256) called a key. The user provides a key which is considered to address the first record in the file which contains the matching sequence in the location specified to contain the key. The key location is specified when the file is created.

Access to the file is expedited by the use of a directory attached to the file. When the file is created, a key extraction interval is specified. The entries in the directory

consist of keys and record pointers that are extracted from records as the file is being recorded in the sequential output mode. There is an entry in the directory for each nth record (where n is the key extraction interval specified at creation). While the directory is being built, the records entered into the file are validated as being in ascending collating sequence by key. If records are inserted, deleted, or replaced in the file after it has been built, the directory remains unchanged.

Partitioned organization access method (POAM)--The partitioned organization access method allows a user to process files which are organized as partitioned files. Partitioned files are organized in the same manner as logical sequential files with the addition of an auxilliary directory and special records that mark the boundaries of each of the sub-files (members).

When a file is opened as output sequential, the records entered in the file are written after the existing end of file record. The records are made a part of the file when the user issues the BUILD macro instruction. The BUILD macro specifies a member name to be associated with the new records. If a member of the same name already exists, it is deleted. The new member is then entered in the directory, the end of file is moved to the end of the new member and a member separator

record replaces the old end of file. The file is then ready for entry of more records to form another new member.

Access to the file other than building a new member is similar to access to files via LSAM. The user may issue macro instructions to set the current-position pointer to the first record in any member. The user then has access to the member as if it were an individual file. The current-position pointer may be saved and restored later even though the user has switched to processing of another member.

There are many file processing considerations that are applicable to more than one file type. These considerations can be described in many ways. But the most important aspects of the file processing characteristics of the Harris 1600 Distributed Processing System (ECOS) are summed up as mode, type, and addressing. These three processing characteristics are described as follows.

(A) Mode

Data of a record is made available to or by the file system in one or two modes, move mode or locate mode. In the move mode, the record is moved from an input buffer to a user's work area or from the user's work area to an output buffer. In the locate mode, the file system provides the user a memory address where the record that has been read can be found or where to place the record to be written. In locate mode the maximum record length is

limited to 256 bytes. The register specified by the user is used to communicate the starting address of buffers in locate mode. If a file is opened for output in locate mode, the specified register will contain the address of the first output buffer.

(B) Type

(B.1) Input

Input processing allows retrieval of records from files and positioning within the file according to the addressing technique.

(B.2) Output

Output processing is allowed only under the sequential addressing technique. It allows the extension of a file by adding new records to the end of the file, and it allows rewriting all of part of a file by positioning to any point internal to the file and creating new records (this deletes old records from the starting position to the end of file).

(B.3) Update

Update processing allows positioning within a file, retrieval of records, and based on the access method used, it may allow all or some of the following operations: deletion of records, insertion of records, or replacement of records. This type is applicable only to disk files.

(C) Addressing

There are two basic techniques used in addressing records in a file: Random and Sequential. In the random addressing mode, an identifier, called a key, explicitly addresses each record that is accessed. In the sequential addressing mode, records are implicitly addressed according to a current-position pointer. During processing the value of the current-position pointer may be altered by a special set of services called "positioning" services. Services which access data records use the current-position pointer to implicitly specify the address of the record to be accessed. Some services which access data records also update the current-position pointer after accessing the record.

CHAPTER BIBLIOGRAPHY

1. Eckhouse, Richard H., Jr., Minicomputer System, Organization and Programming (PDP-11), Englewood Cliffs, N. J., Prentice-Hall, Inc., 1975.

2. Harris 1600 Remote Communications Processor System Description Manual, Manual No. 160001, Dallas, Texas, Data Communications Division, Harris Corporation, (June, 1976).

3. Harris 1600 Extended Communications Operating System (ECOS) User Reference Manual, Manual No. 160002, Dallas, Texas, Data Communications Division, Harris Corporation, (March, 1977).

4. Harris 1600 Systems ECOS Utilities Manual, Manual No. 160032, Dallas, Texas, Data Communications Division, Harris Corporation, (February, 1977).

5. Introduction to IBM Direct-Access Storage Devices and Organization Methods, Form GC20-1649-9, IBM Corp., White Plains, N. Y., 1976.

6. Katzan, Harry, Jr., Computer Data Management and Data Base Technology, New York, N. Y., Van Nostrand Reinhold Co., 1975, 89-139.

7. Katzan, Harry, Jr., Computer Organization and the System/370, New York, N. Y., Van Nostrand Reinhold Co., 1971, 45-85.

8. Lewis, T. G., and M. Z. Smith, Applying Data Structures, Boston, Massachusetts, Houghton Mifflin Co., 1976, 94-96.

9. Rudd, Walter G., Assembly Language Programming and the IBM 360 and 370 Computers, Englewood Cliffs, N. J., Prentice-Hall, Inc., 1976, 385-387.

CHAPTER V

COMPARISON OF FILE ORGANIZATIONS

In comparing the file organization techniques of large and small scale computer systems, the first thing that is apparent is the obvious differences in size of the two computer systems (see Table II). The manifestation of this difference is more storage capacity and increased input/output capacity in the large scale system. Indeed, even within the same system, various file organization techniques differ widely in their organization and performance. Depending on the type of transactions involved (e.g., creation, insertion, deletion, or random retrieval of a large set of records) some techniques may out-perform others in particular situations (2). Thus, it is appropriate to compare the organizational differences at this point.

A critical analysis of the large and small scale system's file organization and access methods is presented here for comparison. The characteristics of each system play a significant role in its data handling capabilities. The respective system file organization techniques show two important similarities and one significant difference in their approaches to file manipulation. The data management facilities and file organizations available under each system represents the

similarities. The difference is found to exist in the access methods used by both systems.

Similarities

The similarities that exist between the large and small scale system's data management facilities and file organizations are summarized in the following paragraphs. Both data management facilities indicate similarity in the input/output operations performed.

The data management facilities in the large scale system provide a software interface between processing programs and auxiliary storage (2). Functions performed by this data management are: (1) assigning space on direct-access volumes, (2) maintaining a catalog of file names, (3) performing support processing for input/output operations and performing buffering and blocking/deblocking.

The data management facilities in the small scale system provide a high-level interface to the input/output devices via a file system (1). The file system is concerned with the movement of data between main storage and external storage devices, and with the maintenance of data on direct-access devices (disks). Functions performed by this data management are: (1) utilize basic input/output operations (i.e., READ, WRITE, etc.), (2) provide record blocking/deblocking, (3) divide a disk volume into Units of Allocation (UAs) for addressing and space allocation, (4) maintain an entry of all file

names in a Volume Table of Contents (VTOC), and (5) allow disk file control services for user application program (e.g., CREATE, DELETE, OPEN, CLOSE, etc.).

The two computer systems show similarity in the structure of their file organizations. Both computer systems follow the same philosophy in design and file processing in each file organization available. The file organizations of each system are indicated in Table IV, with the access methods listed on the same line bearing a close resemblance to one another in the first two columns.

Differences

In contrasting the access methods used by both the large and small scale systems, probably the largest difference is in the way the access methods are chosen. There is an access method for each type of file organization and possibly specialized access methods for specialized file techniques, such as virtual storage access method (VSAM).

The large scale system's access methods are determined by the combination of a given file organization and the technique used to access the data (i.e., basic or queued). The reason for different access techniques is to give the user various options between ease of programming and the amount of control over the desired input/output operation.

The basic access technique can be used with all forms of file organizations within the large scale system and the queued

access technique is used only with sequential organization (for example see Table III). The access methods are identified primarily by the file organization to which they apply. Although an access method is identified with a particular file organization, there are times when an access method identified with one file organization can be used to deal with a file usually thought of as organized in a different manner.

On the other hand, the small scale system's access methods are predefined with each file organization. Each file organization has an associated access method. The logical sequential access method (LSAM) is the only access method that can be utilized to access other type file structures (e.g., keyed sequential or partitioned).

Additionally, in creating a file the initial and continuous space allocation is accomplished by the system. Then the user may specify the file organization, data characteristics, security passwords, and eventually disposition of the file (whether it is permanent or temporary). Default file specifications are used when desired or possible. This provides the user with less programming effort and knowledge about the file.

The structure and method of implementation of the various access methods between computer systems bear a striking resemblance. Table IV indicates the access methods and access modes (i.e., the basic methods of accessing or addressing data on a direct-access device) that correspond between the large and small scale systems. The only access method that is

TABLE IV

FILE ACCESS METHODS AND ACCESS MODES

File Organizations and Access Methods		Access Modes	
Large-Scale	Small-Scale	Large-Scale	Small-Scale
Sequential	Logical Sequential	Sequential	Sequential
Direct	Relative Record	Random	Random
Indexed-Sequential	Keyed Sequential	Random, Sequential	Random
Partitioned	Partitioned	Sequential	Sequential
Virtual Storage		Sequential	

implemented on the large scale system that is not available on the small scale system if the virtual storage access method. This access method is employed by the large scale system's operating system which includes the facilities for a powerful data management system. The small scale system does not have implementation of virtual storage access techniques. This type of access method within a small scale system could prove to be costly in terms of time to design and implement, and in terms of storage capacity.

Table V presents a comparison of both computer systems: the large scale system, and the small scale system. There are several advantages in the methodologies used by both systems. The two principal advantages of the large scale system are (1) the user states the access method requirements and the operating system decides which access method will be used, and (2) more storage capacity and increased input/output capability available by the system (see Table II). The primary advantages of the small scale system are (1) less effort and knowledge of the file is required, (2) additional file storage space is automatically allocated for the user, and (3) the file structure is indicated by the file organization used.

TABLE V

COMPARISON OF SYSTEMS

System	Advantages	Disadvantages
Large-Scale Computer System (IBM System/370 [OS/370])	(1) user states the access method requirements, the operating system decides which to use (2) large storage capacity and increased input/output capability	(1) user responsible for describing the file to be processed (2) the amount of secondary storage must be allocated by user (3) no user awareness of storage structure used for file organization
Small-Scale Computer System (Harris 1600 Distributed Processing System [ECOS])	(1) less effort and knowledge of file required by user (2) additional file storage allocation is automatically allocated (3) the file structure is indicated by the file organization used	(1) user has no choice of access methods for file organizations with default options (2) hardware shortcomings cause slower file or record processing

CHAPTER BIBLIOGRAPHY

1. Harris 1600 Extended Communications Operating System User Reference Manual, Manual No. 160002, Dallas, Texas, Data Communications Division, Harris Corporation, (March, 1977).

2. Katzan, Harry, Jr., Computer Organization and the System/370, New York, N. Y., Van Nostrand Reinhold Co., 1971, 45-85.

CHAPTER VI

SUMMARY

In summary, the comparison of file organization techniques used on large and small computer systems have proven to be different and similar in certain areas. The difference appeared in the access methods used by both system, along with the file organizations and data management facilities showing the similarities. The respective systems used in this comparison were: the IBM System/370 (OS/370) and the Harris 1600 Distributed Processing System with the Extended Communications Operating System (ECOS).

The major parallels that exist between the large scale computer system (IBM System/370), and the small scale computer system (Harris 1600 Distributed Processing System) are the file organizations and the data management facilities. Both systems achieve two basically different functions. First, they are responsible for manipulating data on storage devices in an efficient manner. Second, both systems give the user the ability to create his own name space and to store and retrieve data from this space in a flexible manner. Consequently, in comparing the file organization techniques of the large and small scale systems, they both maintain these basic functions.

However, the differences between each system's access methods reflect the large scale system's capability for more data manipulations and large file operations. The small scale system implies that its access methods are less flexible and require decreased efforts by the user.

Historically, IBM has made available supporting software for all of its file organization techniques. The IBM supported file organization techniques have in turn served as a model or as a starting point for many other users and computer systems. Hence, the file organization techniques of the Harris 1600 Distributed Processing System (ECOS) appear to have most of the generality of the IBM system.

However, with the technology advances in small scale computer systems, the cost of such systems is already significantly lower than the large scale computer system. Thus, coupled with improved reliability and maintainability of file organizations, the small scale computer system can become as efficient in data manipulations as the large scale computer system. Most important, however, the file organization techniques of the small scale system offer the opportunity to move data between main storage and external devices and provide maintenance of data on mass storage devices (disks) identical to the large scale computer system.

Although the small scale system's data capabilities are limited with the present-day core memories, the future can be

different with the new memory technology advancements. The introduction of magnetic bubble memories (1) will increase the data capabilities of the small scale system.

Magnetic core is presently the most commonly used memory element in small scale computers, largely because of low cost. Small scale computer memory ranges from 1K to 128K words, with core memory speeds typically ranging from 0.6 to 2 microseconds.

The magnetic bubble memories have an average access time of 4.0 milliseconds. This speed is somewhat slower than core memory, but is much faster than direct access storage devices.

The magnetic bubble memory has a large number of applications that can be incorporated with specific computer systems. The following are suggested applications for the magnetic bubble memory for three types of computer systems.

1. Micro-processor based systems
    a. Data terminal
    b. Intelligent terminal
    c. Portable data storage
    d. Hobbyist applications
2. Large scale computer systems
    a. Fixed head disk replacement
    b. Buffer storage for peripherals
3. Small scale computer systems
    a. Memory extension

b. Limited replacement where performance and size outweigh media cost (i.e., floppy disk, cartridge disk, and reel-to-reel tape).

While looking at future considerations of file systems, the user has become the main objective. Perhaps the most pressing problem in the effective use of computer systems today is that of organizing data to be responsive to the varied requirements of users.

Recent and future developments of file systems suggest that computer systems are becoming more user oriented than they have been. Increasingly, information management systems will make more of the optimization decisions relating to file organization and compromises between different user requirements. The trend appears to be turning the computer toward user requirements rather than bending the user to the requirements of the computer. This trend will continue to result in progressively easier to use systems.

CHAPTER BIBLIOGRAPHY

1. Magnetic Bubble Memory Marketing Specifications, Dallas, Texas, Texas Instruments, Inc., (September, 1976).

APPENDIX I

DEFINITIONS OF KEY WORDS AND PHRASES

This appendix defines some of the special key words and phrases used throughout this paper. The entries are arranged in alphabetical order. The main purpose of this appendix is to be consistent in the use of key words, phrases and terminology within the framework of this paper.

Access method--a technique for moving data between main storage and input/output devices.

Attribute--a characteristic; for example, attributes of data include record length, record format, file name, associated device type and volume identification, use, and creation date.

Binary search--a dichotomozing search in which the number of items of the set is divided into two approximately equal parts at each step of the process. Appropriate adjustments are made for dividing an odd number of items.

Block of records--a group of records that is considered to be one physical segment of data for the purpose of recording that data on a storage device. Certain data management routines can collect these records into blocks or extract them individually from the blocks.

Buffer--a storage area used to compensate for a difference in operating speeds of two physical devices.

Buffer pool--a collection of contiguous buffers, which can be assigned to a single file or to a group of files.

Current-position pointers--a pointer which contains the address of the current record.

Data management--a major function of operating systems that involves organizing, cataloging, locating, storing, retrieving, and maintaining data.

Data set--one or more data records making up a logical grouping of information that may be referred to by a unique name. Data sets are usually considered to reside on a storage device such as disk, drum, or magnetic tape.

Direct access device--any one of a group of data storage devices capable of performing random or direct data retrieval and access.

File--a collection of data which is treated as a unit, each datum consisting of three elements:

(a) a unit--an entity (object, person, concept, etc.) that may be considered for data processing purposes in terms of a finite number of properties;

(b) a property--a characteristic to which measures can be assigned;

(c) A measure--a value capable of being expressed in a finite number of information units (bits, characters, etc.).

File organization--the organization or structure of a file. In most instances the file organization is directly related to the access method that was used to create the file. For example, an indexed sequential file organization is created by the Indexed Sequential Access Method (ISAM).

FIFO--first in, first out queue discipline.

Filial set--a collection of sons descended from a particular node in a tree.

Fixed-length record--a record having the same length as all other records with which it is logically or physically associated.

Key--the identity field or identifier of a record. A name or value associated with a particular record for the purpose of unique identification of that record or to synonomously associate it with other records.

Leaf--a terminal node of a tree.

LIFO--last in, first out queue discipline.

Linked list--a data storage structure in which each record contains a pointer which indicates where the next record is located.

Pointer--a special reference field incorporated into the structure of a record; a pointer contains a reference to the next related record in logical (not physical) sequence.

Record--a collection of related data items. A collection of related records makes up a file.

Record overflow--track overflow; a hardware/software feature that allows the automatic continuance of a data record between two contiguous tracks on a direct access device.

Ring--a linked list that closes upon itself.

System catalog--a file consisting of entries corresponding to each disk volume and disk file accessed by the system. The entries within a system catalog contain the information that connects file names to corresponding volumes and VTOC entry numbers.

Terminal node--a node of a tree which has no successors (a leaf).

UA--see units of allocation.

UA bit map--an index that indicates free or allocated UAs on the volume.

UA descriptor--a file identifier which contains pertinent information about a file (i.e., file name, starting address, number of records, etc.).

Units of allocation (UAs)--a method of dividing a disk volume (sectors) for addressing and allocating purposes.

Update--a method of modifying a master file with current information, according to a specified procedure.

User area--disk area that is available for user files.

Variable-length records--data set records whose length is not necessarily consistent throughout the overall data set.

Each record contains a four-byte prefix specifying the total length of that record including the prefix bytes.

Volume--the number of records within a data set. A complete unit of storage for a data storage device; a disk pack, a drum, a bin of a data cell, or a reel of magnetic tape.

Volume Table of Contents (VTOC)--a file containing volume-related information, UA bit map for the volume, and entries identifying each file in the volume.

APPENDIX II

FILE SYSTEM PERFORMANCE STATISTICS FOR SMALL SCALE SYSTEM

Function	Transfer Rate	System Utilization	Buffering Level Input	Buffering Level Output	Avg. Disk Accesses Per Transfer
COPY CARDS→DISK	551.56 cpm	28.59%	0	0	2.34
COPY CARDS→DISK	581.69 cpm	19.30%	0	1	0.42
COPY CARDS→DISK	579.87 cpm	18.51%	0	2	0.22
COPY DISK →DISK	381.49 cpm	23.54%	0	0	3.83
COPY DISK →DISK	638.25 cpm	24.42%	0	1	1.89
COPY DISK →DISK	744.57 cpm	27.21%	0	2	1.72
COPY DISK →DISK	452.80 cpm	22.07%	1	0	2.86
COPY DISK →DISK	478.36 cpm	22.48%	2	0	2.68
COPY DISK →DISK	1087.45 cpm	26.72%	1	1	0.95
COPY DISK →DISK	1360.39 cpm	27.79%	2	2	0.56

*Milliseconds

cpm--cards per minute

APPENDIX II--Continued

Average Accesses/Sec	Average Access Time*				UA Allocation
	Physical	Soft	Total-Get	Total-Put	
21.55	28.25	6.24	26.82	80.70	PREALLOCATED
4.03	24.56	11.77	87.68	15.26	PREALLOCATED
2.16	30.08	18.10	92.11	10.60	PREALLOCATED
24.34	33.68	5.02	54.92	93.32	PREALLOCATED
20.08	40.17	5.41	56.74	29.40	PREALLOCATED
21.29	36.54	4.80	56.24	14.87	PREALLOCATED
21.61	38.53	4.72	28.74	94.96	PREALLOCATED
21.40	38.68	4.78	17.76	98.72	PREALLOCATED
17.21	45.30	3.17	21.51	24.54	PREALLOCATED
12.62	60.30	2.95	15.93	19.49	PREALLOCATED

*Miliseconds
cpm--cards per minute

BIBLIOGRAPHY

Books

Eckhouse, Richard H., Jr., Minicomputer System, Organization and Programming (PDP-11), Englewood Cliffs, N. J., Prentice-Hall, Inc., 1975.

Katzan, Harry, Jr., Computer Data Management and Data Base Technology, New York, N. Y., Van Nostrand Reinhold Co., 1975.

Katzan, Harry, Jr., Computer Organization and the System/370, New York, N. Y., Van Nostrand Reinhold Co., 1971.

Kindred, Alton R., Data Systems and Management, Englewood Cliffs, N. J., Prentice-Hall, Inc., 1973.

Knuth, D. E., Fundamental Algorithms: The Art of Computer Programming, Vol. 1, Reading, Massachusetts, Addison-Wesley, 1968.

Lefkovitz, David, Data Management for On-Line Systems, Rochelle Park, N. J., Hayden Book Co., Inc., 1974.

Lewis, T. G. and M. Z. Smith, Applying Data Structures, Boston, Massachusetts, Houghton Mifflin Co., 1976.

Rubinoff, Morris, Advances in Computers, New York, N. Y., Academic Press, 1972.

Rudd, Walter G., Assembly Language Programming and the IBM 360 and 370 Computers, Englewood Cliffs, N. J., Prentice-Hall, Inc., 1976.

Articles

Arora, S. R. and W. T. Dent, "Randomized Binary Search Technique," Communications of the ACM, XII, (February, 1969), 77-80.

Bourne, C. P. and D. F. Ford, "A Study of Methods for Systematically Abbreviating English Words and Names," Journal of the ACM, VIII, (April, 1961), 538-552.

Chapin, Ned, "A Deeper Look at Data," Proceedings of the ACM National Conference 68, (1968), 631-638.

Chapin, Ned, "A Comparison of File Organization Techniques," Proceedings of the ACM National Conference 69, (1969), 273-283.

Davidson, L., "Retrieval of Misspelled Names in an Airlines Passenger Record System," Communications of the ACM, V, (March, 1962), 169-171.

McGee, William C., "File Structures for Generalized Data Management," Proceedings of the IFIP Congress 68, (1968), 68-73.

Morris R., "Scatter Storage Techniques," Communications of the ACM, XI, (January, 1968), 38-44.

Roberts, David C., "File Organization Techniques," Advances in Computers, edited by Morris Rubinoff, XII, (1972), 115-174.

Sussenguth, E. H., Jr., "Use of Tree Structures for Processing Files," Communications of the ACM, VI, (May, 1963), 273-279.

Publications of Learned Organizations

Harris 1600 Remote Communications Processor System Description Manual, Manual No. 160001, Dallas, Texas, Data Communications Division, Harris Corporation, (June, 1976).

Harris 1600 Extended Communications Operating System (ECOS) User Reference Manual, Manual No. 160002, Dallas, Texas, Data Communications Division, Harris Corporation, (March, 1977).

Harris 1600 Systems ECOS Utilities Manual, Manual No. 160032, Dallas, Texas, Data Communications Division, Harris Corporation, (February, 1977).

Introduction to IBM Direct-Access Storage Devices and Organization Methods, Form GC-1649-9, IBM Corp., White Plains, N. Y., 1976.

Magnetic Bubble Memory Marketing Specifications, Dallas, Texas, Texas Instruments, Inc., (September, 1976).