CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

DESIGN OF DATA MANAGEMENT SYSTEMS

A graduate project submitted in partial satisfaction of the requirements for the degree of Master of Science in Engineering

by

Daniel Joseph Schumaker

May, 1975
The thesis of Daniel Joseph Schumaker is approved:

Committee Chairman

California State University, Northridge

May, 1975
The work documented in this thesis was conducted for the Pacific Missile Test Center during the Technical Professional training program and funded by the TERPES (Tactical Electronic Reconnaissance Processing and Evaluation Segment) project.

Special gratitude is due to Wendell Oyama for his patience and helpful comments during the design phase of the project. Acknowledgement is also due to R. B. Lindsay for his continued concern and support during some of the more discouraging events.
TABLE OF CONTENTS

Preface ........................................ iii
List of Illustrations ............................ v
List of Symbols .................................. vi
Abstract ....................................... vii
Introduction .................................... 1
Data Management Methodologies .............. 2
Overview of the TERPES System .............. 8
Magnetic Tape Simulator for Disk Usage .... 10
The TERPES File Librarian ..................... 16
References ...................................... 23
Bibliography .................................... 24
Appendix A ...................................... 25
Appendix B ...................................... 48
Appendix C ...................................... 52
# LIST OF ILLUSTRATIONS

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>26</td>
</tr>
<tr>
<td>Figure 2</td>
<td>27</td>
</tr>
<tr>
<td>Figure 3</td>
<td>27</td>
</tr>
<tr>
<td>Figure 4</td>
<td>28</td>
</tr>
<tr>
<td>Figure 5</td>
<td>35</td>
</tr>
<tr>
<td>Table 1</td>
<td>35</td>
</tr>
<tr>
<td>Table 2</td>
<td>36</td>
</tr>
<tr>
<td>Figure 6</td>
<td>37</td>
</tr>
<tr>
<td>Figure 7</td>
<td>37</td>
</tr>
<tr>
<td>Figure 8</td>
<td>38</td>
</tr>
</tbody>
</table>
LIST OF SYMBOLS

ABT- Available Block Table
BSF- Backspace File
BSR- Backspace Record
DIS- Display
DOA- Direction of Arrival
EOB- Electronic Order of Battle
EPL- Electronic Parameter List
FDT- File Data Table
FLT- File Location Table
FSF- Forwardspace File
FSR- Forwardspace Record
FT- File Table
LOC- Location
NAV- Navigation
ONPLT- On-Line Plot
P_n- Pointer to Current Record
P_{n-1}- Pointer to Previous Record
P_{n+1}- Pointer to Next Record
RPT- Report
RR- Read Routine
RWD- Rewind
SCR- Scratch
WR- Write Routine
ABSTRACT

DESIGN OF DATA MANAGEMENT SYSTEMS

by

Daniel Joseph Schumaker

Master of Science in Engineering

May, 1975

A description of the basic concepts of data management techniques is presented, including a discussion of the advantages and disadvantages of each concept. The operational characteristics and requirements of a data processing system currently being developed to provide information concerning the location and operating characteristics of land based radar stations are summarized in order to provide an insight into the trade-off considerations of several associated software designs. The design, development, and testing of a magnetic tape simulator for implementation on a random access device are described within the context of data management concepts. Finally, the design of a file librarian which moves files between a random access secondary storage device and core storage is recounted and several ideas for improving future designs are mentioned.
INTRODUCTION

Two program modules for use in the TERPES (Tactical Electronic Reconnaissance Processing and Evaluation Segment) system were developed and are the subject of this thesis. Both modules involve the features and functions of a data management system. The first chapter discusses the basic concepts and methods used in developing data management systems, including factors to be considered in applying the concepts to the design of future systems. The second chapter explains the requirements and restrictions imposed on the TERPES data management system due to the hardware configuration and anticipated operating environment. This chapter provides a framework for discussing the relative merits of the data management concepts proposed in the design of programs discussed in subsequent chapters. The next section describes the design of a magnetic tape simulator implemented for a disk. Included in the description are the alternative design considerations and associated implications, operational characteristics of the implemented design, and considerations for future use and further optimization. The last section recounts the design of a file librarian which utilizes a hybrid data management organization. The considerations involved in the initial design and re-design are disclosed and constructively criticized. This chapter also discusses problems associated with the management of secondary memory and file protection schemes from a general viewpoint and as implemented in the file librarian. Finally, the testing methods employed in verifying the operation of the file librarian are discussed and the program is evaluated to reveal inadequacies and areas for improvement.
DATA MANAGEMENT METHODOLOGIES

The primary purposes of a data management system are to store, retrieve, and update the data for various users. Such a process involves several important secondary considerations, such as that of peripheral independence, data privacy, fault protection, and storage allocation. As an example, the MULTICS (Multiplexed Information and Computing Service) system allows symbolic addressing in order to provide flexibility of secondary storage due to reliability and capacity changes. Furthermore, this system provides for retrieval of data accidentally released by a user or lost due to a total mishap.¹

The reasons for differing data organizations are due to the demands of the users. In some instances data must be retrieved rapidly with updating of the existing data being allowed to proceed at a much slower pace. However, in other cases it may be necessary that storage of data proceed as rapidly as possible with retrieval being allowed to occur more slowly. Furthermore, the data storage medium may not be appropriate for the intended needs. In this instance it is the responsibility of the data organization to provide a bridge over this gap.²

All data organizations can be constructed from one or a hybridization of three basic organizations: sequential, random, and list.³ In order to discuss the above three organizations in detail it is necessary that the terms record, file, and key be defined. A record can be considered as a collection of data items. An example of a record might be a collection of the name, age, weight, and salary of an employee. A file is a collection of records. When a data item
within records is used to order records in a file it is termed a key.

A sequential data organization is arranged such that records are stored in areas relative to other records in a specified sequence. This sequence may be established according to a key or merely in the order in which the data is collected. The major advantages of a sequential data organization are rapid retrieval of consecutive records and the ability to quickly search a sorted file by means of a binary search where half of the remaining records are discarded after each comparison of the requested key and the key of the stored records. However, the ability of rapidly accessing the next record becomes a disadvantage when an unsorted file has to be searched for a specific key as each record prior to the desired one has to be examined.

Another disadvantage of a sequential data organization is that replacing records with records of a different length, inserting records, and deleting records becomes very cumbersome. When such updating is required the general procedure is to insert, delete, and modify the records as the file is rewritten into another area. Finally, the sequential data organization provides no means of storing and retrieving records by means of several keys except to store a version of the file sorted according to each key.

In a random data organization the records are stored and retrieved according to a deterministic relationship between the key and the physical address of the storage medium. As most random data organizations make use of a direct access storage device three primary methods of accessing records are used: direct address, dictionary look-up, and calculation. The direct access method involves the user knowing the physical
address of unused storage and merely associating this address with a record. The dictionary look-up method involves the storage of both a key and a physical address in a dictionary. To retrieve a record the dictionary is searched for the key and the corresponding address is used. The advantage of this method is that each record has a unique address which can be determined rapidly if the dictionary is ordered. Because the dictionary may become as large as the data and retrieval becomes slow when the dictionary is not ordered, the advantages of this method are frequently offset by these disadvantages. The calculation method involves converting the key of the record into a direct address which might not be unique. If two keys yield the same address an overflow condition is said to occur. This overflow condition can be handled by placing a pointer from the record at the overflow address to the overflow record which is placed at another location on the storage medium. Retrieval consists of calculating the address of a record, comparing the key of the stored record with that of the requested record, retrieving the record if the keys are the same or accessing another record as addressed by the overflow pointer, comparing the keys, and continuing the process until a match of keys is discovered. Another method of handling the overflow situation is to use another calculation method each time an overflow condition occurs.

The advantages of random data organizations are that a record can be obtained by a single access and updating, storage, and retrieval can be performed without affecting other records on the storage medium. The disadvantages of this type of data organization are that rapidly accessing a large number of records is not possible due to the hardware accessing mechanism, handling the overflow problem can become
quite difficult when the calculation method is used, and manipulating large dictionaries can become unwieldy when the dictionary method is used.

The final type of data organization is called the list organization which is somewhat similar to the method of handling overflow in the random data organization. The three main types of list organizations are: simple list, inverted list, and ring. The fundamental idea to a list organization is that pointers are used to detach the logical organization from the physical organization.

In a simple list structure pointers to the first logical records of the organizations are stored in a known and easily accessible area. The pointers to other records of the logical organization are stored within the records. As any data item in the record may be considered a key, several lists can pass through a single record. Insertions are readily performed in such organizations by merely changing the pointer in the record preceding the new record and placing a new pointer in the record just inserted. Deletion only involves changing the pointer in the preceding record to point to the record following the record being deleted. However, deletion is somewhat more complicated than insertion, especially in records which are members of several lists, as more pointers have to be stored in the records to indicate the location of the previous records. In large files these lists may become quite lengthy thus requiring extensive searches. This problem can be alleviated, at the expense of requiring more storage, by creating sublists, each of which has its own starting point.

An inverted list is produced when sublists are used to the maximum extent such that each key of every record appears in the index
list. The advantages of this type of structure are that it is possible to process a request without having to access the file, every data item is available as a key, and all data is accessible with equal ease. The disadvantages of this type of data structure are that it requires a dictionary which may be as large as the data to be accessed.

The ring structure is an extension of a list organization such that the last pointer of a simple list points to the first record of the list. The advantages of this data structure are that it is easier to insert and delete records than with a list structure and any record in one ring can be retrieved and processed. Furthermore, any record in the ring can be used as a branch point to retrieve other logically related records which may also be in a ring. Such nesting of ring structures is called an hierarchical storage organization.

The prime disadvantage of all list organization structures is the storage overhead caused by the pointers. Inverted lists cause a reduction in the number of occurrences of a key and may result in some storage space reduction. Consequently, the advantages of storage and retrieval must be balanced against the storage cost of the pointers when considering a list data organization.

Another problem encountered in data management is the allocation and de-allocation of storage space. The following factors regulate the functioning of this process: 1. The new records should be located such that minimum time is needed for retrieval, 2. Space should be allocated so as to decrease unusable fragmentation, 3. Compacting of unused space should take place frequently enough such that large blocks of space are available for re-use, and 4. A protection method should be established such that recovery can be made from hardware or soft-
ware errors. Attaining these factors can be extremely difficult when dealing with random and list organizations.⁵

Some of the more complex data organizations which can be constructed from the previous three data organizations are the indexed sequential structure, tree structures, and multilist structures. The indexed sequential organization has the property of being accessible sequentially or by means of indices. The records are stored sequentially on the storage medium and the indices are stored in a sequentially contiguous area. A multilist file structure is composed of a sequentially organized index with each index entry pointing to a list of records. The records of the file are grouped into units called pages such that entire pages are accessed when a record is stored, updated, or retrieved. In a tree structure several levels of indices are used such that indices at any level point only to indices further down the tree or to indices at the same level.⁶
OVERVIEW OF THE TERPES SYSTEM

The purpose of the TERPES (Tactical Electronic Reconnaissance Processing and Evaluation Segment) system is to provide post-flight analysis of data tapes containing information concerning the location and operational characteristics of ground based radars. Within one hour of data acquisition the system must be able to provide a summary report of all new information and within six hours all processing of either analog or digital data tapes must be completed including all necessary reports. The system must also provide a means of man-machine interactive processing while allowing for degraded modes of operation and future expansion of mission processing.

The configuration of the envisioned operational hardware is as shown in Figure 1. The computer is a second generation type with an average instruction cycle time of 8 microseconds, 16 input/output channels with separate processors, and a main memory capacity of 32K words (30 bits/word). The interactive display consists of a mini-computer with 8K of 16 bit words, a peripheral keyboard, symbol and vector generators, and light pen. The disk is functionally compatible to a CDC model 9740 disk unit which has a data transfer rate of 2.5 mega-bits/second, a storage capacity of 416 million bits, and a spindle speed of 2400 RPM.

The major constraints associated with the development of the software are directly related to the hardware and include program size and execution time, as "...it is more difficult to prepare system programs for smaller computers when limited by speed or memory size...". Several other factors which must be kept in mind during software devel-
opment are reliability, configuration control, and the need for efficient updating. Configuration control is a factor because it necessitates that all software modifications be performed at the developing agency and not at the operational facilities. As modifications would be quite time consuming and costly, this ultimately leads to the need for greater reliability.
MAGNETIC TAPE SIMULATOR FOR DISK USAGE

It was deemed necessary for the TERPES system that a means be provided for the continued processing in a degraded mode due to a failure of one or more magnetic tape units. As the only other secondary storage medium in this system was a random access disk, it was proposed that the disk be used to simulate the four magnetic tape units. Such a problem appears somewhat unrelated to a data management organization. However, this problem can be viewed in such a light if the broad view of the purposes of a data management organization, as presented in the previous section, are considered.

Because magnetic tape is a sequential storage medium, it was only necessary to find a means of converting the random accessing characteristics of a disk to the sequential characteristics of tape. A random data organization was promptly discarded as it lacked the major ingredient of accessing speed, so characteristic of a sequential organization. Another factor in eliminating this organization was that a dictionary, which ran counter to the requirement of using as little main memory space as possible, would have been necessary. The sequential data organization was also discarded, for if the disk had been a sequential device the problem would have been solved. As a result of the preceding survey it was concluded that a list organization was required. The simple list structure was finally adopted as being the best suited type of list organization for the simulator. This was found to be the case due to the fact that an inverted list structure would have required too much main memory storage and the ring structure would have resulted in an exceedingly lengthy execution time.
during backspace operations. The simple list structure which was finally implemented made use of pointers indicating the location of the previous record and the next record and were stored in the first two words of each record.

One of the requirements for the simulator was that the input/output characteristics be the same as those of the magnetic tape units used in the system. This meant that all operation codes for the simulator, the method used to call the simulator, and the status conditions returned to the user had to be identical to those of the tape units. Another of the requirements was that a means had to be designed such that automatic recovery would result if a segment of the disk surface became damaged and unusable for storage.

In order to better understand how the design of the tape simulator was accomplished it is necessary to understand the operation of the disk and how it was related to the tape units. The surface of the disk could be considered as composed of a number of concentric paths. These paths are commonly termed tracks. Furthermore, individual tracks could be regarded as being divided into several arcs of the same length. These arcs are called sectors. As the disk unit was composed of several surfaces with separate read/write heads, which position as a unit to a specific track, a convenient concept was that of the cylinder. A cylinder was viewed as consisting of all of the tracks at the same radius from the spindle. These concepts are further clarified in Figure 2.

The reason the cylinder concept was important was that accessing consecutive tracks within a cylinder was faster than accessing tracks on different cylinders due to large seek times involved in moving the
As a high data transfer rate was desirable, the cylinder concept was used in representing a magnetic tape as a sequence of cylinders connected in a head to tail fashion as shown in Figure 3. It should be pointed out that a certain amount of wasted space was associated with this design because records could not begin in the middle of sectors. However, this loss of storage space was felt to be tolerable as the disk had a storage capacity considerably larger than that of four tapes.

Two methods were initially considered in order to provide automatic recovery of processing due to a damaged surface. The first method consisted of maintaining a list of all damaged sectors such that writing on these areas would not be attempted. The second method consisted of calculating the last address of the present record, writing the present record, comparing the actual ending address of the present record with the calculated address, and terminating normally if the addresses agreed. However, if the addresses were not in agreement the present record would be rewritten with new pointers indicating a new address for the next record. The first method was originally adopted as it appeared to fulfill the design criteria of shorter execution time and smaller main memory requirements better than the second method. However, after considering both approaches in more detail it was determined that the first method was faster only in instances where prior knowledge about the location of bad sectors existed and that it would be slower when bad sectors were first detected. The actual reasons the first method would have been slower in this instance were that an extra table look-up would have been required and a new bad sector entry would have to have been added to this table.
As the bad sector table was to be located on the disk this would have required additional accessing time. Consequently, the second method was accepted and appeared in the final implementation of the simulator.

The simulator program was coded in a modular fashion as can be seen from the functional flow diagram in Figure 4. One irregularity in this modular concept occurred in the write routine and is evidenced by the fact that two different operations branch to this routine. One operation is a record write while the second is a file mark write. The reason for this irregularity was to save main memory space. The disk handler, which was developed by another software designer, and the simulator were initially 2100 locations in length. As this was considered excessively lengthy, the modification of the disk handler was undertaken with the result of decreasing the combined length of the disk handler and the simulator to approximately 1200 locations.

The verification of the tape simulator consisted of three phases. The first phase consisted of testing all program paths which did not involve automatic recovery techniques. This phase was the most time consuming of all three phases due to a dynamic modification of the disk handler during execution. The reason for this problem was not readily apparent and resulted in a thorough investigation of both the tape simulator and the disk handler in op step mode during execution and at machine code level with paper and pencil. The efforts of these investigations produced no explanation of the problem. Finally, during a discussion of this problem with the designer of the disk handler an hypothesis as to the cause of the dynamic modification was presented. This hypothesis was later tested and verified. The problem was found to be caused by a control table used by the disk handler.
which had been destroyed during training exercises on the operation of the disk. After this problem was corrected the test revealed only three minor errors in the simulator which were subsequently corrected.

The second phase of testing consisted of examining the automatic recovery techniques for any possible errors. A test program which required records to be written over various groupings of simulated bad sectors was created and executed. An illustration of the bad sector layout and anticipated automatic recovery results afforded by this test program can be seen in Figure 5. The results indicated that the automatic error recovery was unsuccessful when a bad sector immediately followed a record, when a bad sector occurred at the beginning of a cylinder, and when a file mark was immediately followed by a bad sector. The causes of these errors were diagnosed and the simulator changed accordingly. The test was again executed and it was concluded that the tape simulator was free of errors.

The final phase of testing involved gathering data about the operating speed of the tape simulator such that an evaluation of the simulator's performance could be performed. The test program for this phase consisted of requiring the tape simulator and the magnetic tape units to perform the same sequence of functions. The data obtained from this test allowed the tape simulator and the magnetic tape units to be directly compared and an average execution time to be obtained for each of the functions. The data derived from this test can be seen in Table 1.

The results of the final test indicate that the tape simulator is considerably faster when searches of records and files are required or when considerable rewinding occurs. Furthermore, an in-
crease in sector length was found to cause a slight increase in the data transfer rate for read and write operations, although under no circumstances were these operations considerably faster than for the magnetic tapes. Notwithstanding the fact that the simulator is error free and provides for automatic recovery of certain disk hardware faults, the basic list structure employed is somewhat unreliable. For example, if a single pointer becomes the subject of an error the entire file becomes useless as no records beyond the one containing the faulty pointer can be accessed.
THE TERPES FILE LIBRARIAN

The purpose of the file librarian was to provide for all on-line file storage and retrieval, and control all file input and output for the TERPES operating system. Provisions were to be made for the following four general types of files: input and output files which were uniquely defined to specific I/O devices, system files, scratch files, and report generator files which contained format and header data for report generation.

Several requirements and constraints were initially imposed on the file librarian. These requirements were based upon information about similar systems and opinions as to the envisioned needs and operating environment of the TERPES system. The quantity of files and records required to be managed are shown in Table 2. The main memory allocation for the file librarian was to be less than 4100 locations and the execution speed was required to be such that the processing time limits previously discussed could be met.

The design of the file librarian consumed more time than was originally expected, as the operation of other sections of the system which would have an impact on the file librarian had not been clearly formulated. Consequently, the file librarian had to be re-designed twice. The preliminary design, as shown in Appendix B, was based on a combination of list and random data management organizations. In the first design the File Data Table (FDT) contained information which was common to all records of a file such as the record length. A pointer was provided in the FDT which was used to locate a File Location Table (FLT) which contained a dictionary of the physical addresses of each
logical record number. This dictionary also contained pointers which could be used to determine the logical record sequence based on the random storage of the records on the storage medium. The advantage of this organization was the ease with which records could be inserted or deleted. While this data organization required a large dictionary, it was felt that the dictionary could easily be placed on secondary storage while still meeting data storage requirements. Furthermore, it was felt that disadvantages so accrued would be more than offset by the general capabilities offered by this method. However, this opinion was not held by other influential members of the project. The envisioned operation was again discussed and recommendations proposed. The recommendations included increasing the processing speed of sequential operations and providing protection against the accidental deletion of records by a user.

As a result of the preceding recommendations the second design was constructed around a combination of sequential and random data organizations. The concept was to divide a file into physically random blocks. Each block would consist of a group of records ranging in length from one sector to one cylinder. Within each of the blocks the records were to be arranged sequentially. This type of data organization was quite similar to that of the previous design with the two exceptions being the requirement that records be stored sequentially within blocks and the association of a direct address with blocks instead of with individual records. The method proposed to provide protection against the accidental deletion of records by setting a flag to indicate released records and deferring the actual deallocation of storage space until the file was closed. Furthermore,
another function was added such that records which had been flagged as being released could be accessed. Therefore, a user was protected from himself as long as he didn't close a file. An illustration of the data organization concepts of this design can be seen in Figure 6. This second design was reviewed and found to be unacceptable due to the large main memory storage requirements necessary for the protection of records. This problem was solved by moving the protection storage requirements from main memory to the disk areas being protected. A pictorial representation of the data organization and program flow can be seen in Figure 7 and Figure 8, respectively.

The functions to be provided in the file librarian consisted of reading records, writing records to the end of a file, replacing existing records with new records, releasing records so as to be un-readable, opening files for reading or writing of records, and closing of files to make them unaccessible. A more detailed description of the various functions and a visualization of the results of the operations can be seen in Appendix C. Two other functions which involved a combination of a read and a write or a read and a modify operation were initially desired in order to eliminate the user's need for reserving a working area. However, after development progressed the need for these operations declined and they were subsequently abandoned.

The two major areas of effort in the development of the file librarian were secondary storage allocation and information protection. The first aspect of storage allocation needing investigation was whether static or dynamic allocation methods should be utilized. Static allocation assumes that the availability of memory resources
can be prespecified and that a program's sequence of references to information can be determined by preprocessing the program or examining the text during compilation. On the other hand, dynamic allocation assumes that memory resources can't or shouldn't be prespecified and that referencing information can only be determined during execution. As the TERPES system was developed around the idea of interactive processing between an operator and the computer, it was readily apparent that dynamic storage allocation was a necessity. The next storage allocation feature to be surveyed concerned the mechanism of transferring data between main memory and secondary memory. The present mechanisms in use in performing this function are called segmentation, in which the address space is organized into variable size "segments" of contiguous addresses, and paging, in which the address space is organized into fixed size "pages" of contiguous addresses. The latter method was finally chosen because a great reduction in the system overhead, associated with the movement of data to and from main memory, could be obtained by regarding the high speed storage as being divided into fixed sized blocks. Furthermore, since pages are all of equal size, space allocation is immensely simplified and the problems of compacting information are eliminated.

As the page concept had been accepted it was only necessary to determine the size of pages which would best meet the demands of the user without sacrificing processing speed. A study conducted by the University of Virginia Applied Mathematics and Computer Science Department to determine an optimal page size for a multiprogrammed system indicated that under routine operating conditions two-thirds of all pages used contained less than 40 words. However, as the
data demands of the TERPES system didn't seem comparable to those of a mix of student jobs, the results of the University of Virginia study were not weighted very heavily in arriving at a decision. The conclusion, based upon hardware considerations and expected processing demands, was to use pages of 449 words. The final feature of secondary storage allocation to be researched was the actual method of allocating storage space. Most information on the topic of dynamic allocation discussed main memory storage allocation. The theory of main memory allocation policies are basically the same as those of secondary memory. Consequently, the results were felt to be applicable to secondary storage allocation. The two most popular schemes for allocating storage are called the best fit and the first fit algorithms. In the best fit algorithm available storage areas are listed in order of increasing size. The allocation of a storage area involves searching the list of available areas until the area which results in the least amount of unused space is obtained. The first fit method involves listing the available storage areas in order of increasing initial address and searching the list until an area is found which is larger than the area necessary for storing the data. The first fit method has been found to be the more efficient of the two algorithms with regard to processing speed and secondary storage requirements. Due to the faster operation of the first fit method and its lesser demand for a compaction routine, the first fit method was selected for the TERPES system. However, the total storage allocation algorithm is a combination of two methods. The operation of the algorithm is such that each file is initially allocated the storage space of an entire disk cylinder. After all cylinders have been allocated and a request
for more storage space is received a de-allocation process is invoked to place areas not presently occupied by records into the table of available areas. Finally, the first fit method is used for all further storage allocation.

The measures of protection provided by the file librarian in the TERFES system are quite unsophisticated when compared with those of complex systems such as MULTICS. The primary areas where protection features were implemented in the system were in providing recovery from a damaged disk surface and preventing the writing or reading of a file which was not opened for such an operation. Protection is not provided to prevent access of files by other users except to require knowledge of the file name. Because certain system file categories may only contain 1000 different files (e.g. DOA123, DOA122, etc.), the possibility of unauthorized accessing of these files is extremely high. Another area where protection is not provided by the file librarian is in the movement of data to and from main memory. In some systems such protection is provided by hardware protection mechanisms which effectively partition main memory into blocks which can't be accessed by unauthorized procedures. In other systems such mechanisms can be provided by software. However, in cases where software is used to provide the protection mechanisms, processing speed decreases and storage requirements increase. In many systems the lack of the preceding protection mechanisms might impair system operation. If prudence is used in the overall system design such mechanisms can be safely eliminated in the TERFES system because of the strict enforcement of configuration control.

The method used in testing the file librarian was to proceed from
the component level to the system level in discrete steps. Therefore, the first routines to be tested were ends in themself and the last routines to be tested were at the functional level requiring the use and interaction of all other routines. Presently, all support routines of the major file librarian functions have been tested and verified. A test program has been developed for the system level test of the file librarian and execution has revealed that errors are still present.

Several important conclusions drawn from experience on the TERPES system are that better project management techniques need to be employed and that the system specifications need to be more clearly stated. Such needs have also been seen at the management level and an evaluation of different management techniques is presently underway. The primary problem in the development of the system is that the "bottom-up" approach of design is being used instead of the "top-down" approach. An excellent example of the "bottom-up" approach employed in this system was placing the design and development of the "executive" at the bottom of the priority scale. In the "top-down" approach the most general aspects of the system such as an executive are designed first and effort continues until the most basic functions are designed. Such a method results in less re-design due to greater visibility, more effective communication, and greater specificity of the system needs. This has certainly not been the case with the TERPES system as several modifications have been required due to changing requirements and a lack of foresight in identifying interface requirements.
REFERENCES


3Ibid., p. 118.


5Dodd, p. 127.


8Corbato and Vyssotsky, p. 187.


10Ibid., p. 209.


12Ibid., p. 209.


BIBLIOGRAPHY


Figure 1: TERPES Hardware Configuration
Track 72 (each surface) or Cylinder 72

Figure 2: Disk Subdivisions

Inter-record Gap

Tape

Record 1

Record 3

Wasted Space

Sector 2

Disk

Cylinder 1

Cylinder 2

Track 2

Figure 3: Disk Simulation of Tape
Figure 4: Functional Flow Diagram of Tape Simulator
Diagram flowchart showing the logic of controlling a disk drive system.
Figure 5: Automatic Recovery Test Layout

Table 1: Performance Test Results
(All Values in Seconds)

<table>
<thead>
<tr>
<th>Function</th>
<th>Unit</th>
<th>Disk 4VU</th>
<th>Disk 4VU</th>
<th>Disk16VU</th>
<th>Disk16VU</th>
<th>Disk16VU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Write</td>
<td>Tape</td>
<td>72.315</td>
<td>62.125</td>
<td>35.835</td>
<td>34.103</td>
<td>41.966</td>
</tr>
<tr>
<td>Read</td>
<td>Tape</td>
<td>70.228</td>
<td>21.040</td>
<td>21.043</td>
<td>22.003</td>
<td>22.003</td>
</tr>
<tr>
<td>Copy</td>
<td>Tape</td>
<td>141.612</td>
<td>94.001</td>
<td>66.762</td>
<td>59.576</td>
<td>63.662</td>
</tr>
<tr>
<td>Rewind</td>
<td>Tape</td>
<td>46.793</td>
<td>46.793</td>
<td>46.793</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>ForwardSpace</td>
<td>Record</td>
<td>21.355</td>
<td>0.067</td>
<td>0.067</td>
<td>0.067</td>
<td>0.068</td>
</tr>
<tr>
<td>Backspace</td>
<td>Record</td>
<td>25.547</td>
<td>0.067</td>
<td>0.067</td>
<td>0.067</td>
<td>0.068</td>
</tr>
<tr>
<td>ForwardSpace</td>
<td>File</td>
<td>20.445</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
<tr>
<td>Backspace</td>
<td>File</td>
<td>19.152</td>
<td>0.010</td>
<td>0.011</td>
<td>0.011</td>
<td>0.011</td>
</tr>
</tbody>
</table>

* 4-indicates 4 sectors per track
16-indicates 16 sectors per track
V-(first letter) indicates file consisting of 22K, 12.4K, 1K, and 128 word length records
S-(first letter) indicates file consisting of four 1K word length records
V-(second letter) indicates write operation includes verify
U-(second letter) indicates write doesn't include verify
<table>
<thead>
<tr>
<th>File Type</th>
<th>Max. No. Files</th>
<th>Words/Record</th>
<th>Max. No. Records</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAVXXX</td>
<td>*</td>
<td>16</td>
<td>15000_g</td>
</tr>
<tr>
<td>DOAXXXY</td>
<td>*</td>
<td>20</td>
<td>15000_g</td>
</tr>
<tr>
<td>LOCXXXYYZZ</td>
<td>*</td>
<td>80</td>
<td>1</td>
</tr>
<tr>
<td>EOB</td>
<td>1</td>
<td>80</td>
<td>15000_g</td>
</tr>
<tr>
<td>EPL</td>
<td>1</td>
<td>80</td>
<td>15000_g</td>
</tr>
<tr>
<td>SCR</td>
<td>*</td>
<td>256</td>
<td>1000_g</td>
</tr>
<tr>
<td>RPT</td>
<td>5</td>
<td>256</td>
<td>1000_g</td>
</tr>
<tr>
<td>DISXXX</td>
<td>*</td>
<td>256</td>
<td>100_g</td>
</tr>
<tr>
<td>ONFLXXX</td>
<td>*</td>
<td>256</td>
<td>100_g</td>
</tr>
</tbody>
</table>

* Combined total number of files will be less than or equal to 1000_g.

Table 2: TERFES Data Handling Capabilities
Figure 6: Re-Designed Data Organization

Figure 7: Final File Librarian Data Organization
Figure 8: Functional Flow Diagram of File Librarian
GETO

FTSCH

Is File Open For A Read

Set Status To "Illegal Call"

RETURN

PHYSAIR

Is Function A GETR

N

Y

Save Initial Pointers

Is Record Outside Of File

Set Status To "End Of File"

RETURN

Is Record In Buffer Area

DISCARD

RETURN

Is Last In EGA

Indicate Next Record Is In Input Block

Has Record Been Released

Is Function A GETR

N

Y

Move Record From Buffer To User's Area

Update Pointers To Locate Next Record

N

Y

Have All Records Been Read?

Set Status To "Good Execution"

RETURN
File Librarian Design

In order for the file librarian to perform its functions of record manipulations between core memory and auxiliary storage, the following tables are required.

1. File Data Table (FDT) -- this table contains information which is common to all records in the file and which locates another larger table that contains information applicable to individual records. These tables (1 FDT/File) will be stored on the disk and individually paged into core memory when its associated file is referenced. Three of these tables will be allowed in core memory simultaneously.

<table>
<thead>
<tr>
<th>NAME</th>
<th>RECSIZE</th>
<th>MAXREC</th>
<th>FIRSTREC</th>
<th>LASTREC</th>
<th>HEADPTR</th>
<th>SEQPTR</th>
<th>STATUS</th>
<th>SPARE</th>
<th>FDT</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME - indicates the file name and number</td>
<td>RECSIZE - specifies the length of records in this file</td>
<td>MAXREC - specifies the maximum number of records in this file</td>
<td>FIRSTREC - indicates the first sequential record of the file by logical number</td>
<td>LASTREC - indicates the last sequential record of a file by number</td>
<td>HEADPTR - indicates the physical location of a header (when applicable) which contains such information as date, pilot, etc.</td>
<td>SEQPTR - indicates the next record which will be accessed if the operation is sequential. The record is referenced by logical number.</td>
<td>STATUS - gives such information as to whether or not the file has been opened for read, write or read and write operations.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FLTADR - gives the physical address of the File Location Table (FLT)
SPARE - extra words which may be required in the future.

2. File Location Table (FLT) -- this table indicates the physical
location of the records of a file. It also indicates the next se-
quential record in the file.

| ADIR | Record No. 1 | The function of the various items are
| REC PTR |
| ADIR | Record No. 2 | ADIR - indicates the physical address
| REC PTR |
| ADIR | Record No. 3 | of record number X of the file
| REC PTR | which links to this table by
|  |  | means of the File Data Table
|  |  | REC PTR - gives the record number of
|  |  | the next sequential record
|  |  | in the file
| ADIR | Record No. n |
| REC PTR |

FLT

After a File Data Table is paged into memory a number of entries
from the FLT, beginning with the entry which locates the present record
being processed, will be paged into core memory from the disk. After
this table has been modified and its associated FDT is required to
leave core memory it will be paged back onto the disk with all changes.

3. Available Block Table (ABT) -- this table indicates the physical
locations on the disk which have not been written into and which are
therefore available for storage of a certain type of record. Files and
records of these files will be typed according to record length.
Specific cylinders of the disk will be reserved for certain types of
records. If this method proves too wasteful of disk storage space, the
cylinders will be resectored to provide a different mix of records
which can be contained on a track. A specific location of the disk
will be reserved for these tables. After a section of one of these tables has been paged into core memory, it will remain there until records of another type of file are accessed.

The function of the various items are as follows:

**SIZE** - indicates the number of continuous sectors available

**ADR** - indicates the beginning physical address of an available block

**LINK** - sequentially links the available blocks of the ABT such that incoming or outgoing available blocks can be coalesced
File Librarian Instruction Set

1. PUTM - Put Modify

Move X records from working area WKAREA and replace records in a library file.

![Diagram of PUTM]

2. PUTA - Put Add

Move X records from working area WKAREA and add to the end of the library file.

![Diagram of PUTA]

3. OPEN

Set the library file to the open status. If the file is open to another process then an error status is returned.
4. **CLOSE**

Set the library file to the closed status. The closed status prevents calling processes from reading or writing onto the library file. If the file to be closed is a scratch file then the file is destroyed.

5. **GETO - Get Only**

Move X records from a library file to working area WKAREA.

6. **GETR - Get and Release**

Move X records from the library file to the working area WKAREA, then release X records from the library file.

7. **GETX - Get Extended**

Move X records, including records which have been previously released, from a library file to working area WKAREA.
8. RELEASE

Destroy X records from the library file.