copy 6 of 20

DRAFT REPORT FOR OFFICIAL USE ONLY

DRAFT

RADC-TR-71-121
Technical Report
May 1971

RADC/MULTICS EVALUATION

DRAFT

Rome Air Development Center
Air Force Systems Command
Griffiss Air Force Base, New York

DRAFT

DRAFT REPORT

FOR OFFICIAL USE ONLY

RADC/MULTICS EVALUATION

Rocco F. Iuorno
Frederick A. Normand
William E. Rzepka
Capt. Andrew Kobziar
et al

Multics

FOREWORD

This experimental exercise was performed totally in-house by the Advanced Software Group of the Information Processing Branch, Information Sciences Division.

The first section of the report introduces the reader to MULTICS and its concepts. The second section discusses the goals of the experiment, the plan for achieving these goals, and a summary of results, observations and conclusions produced by the experiment. Section III contains a detailed description of each of the jobs which were designed, programmed, and implemented for the application testing portion of the experiment. Section IV discusses in detail the operational testing performed including a description of the test programs and results obtained.

This report was written by:

Rocco F. Iuorno
Frederick A. Normand
William E. Rzepka
Capt Andrew M. Kobziar
Frank S. LaMonica
Douglas White
Robert McCauley

ABSTRACT

This report documents the work performed by personnel in ISIS from Jan 70 - Dec 70 in testing, evaluating, and getting hands on experience with the MULTICS operating system under development at MIT, Cambridge, Mass. Results from application testing and operational testing are reported. The application testing attempted to determine the value of MULTICS for handling future A.F. data processing problems. The operational testing included performance and reliability measurements. In the overall testing more emphasis was placed on the application testing since MIT, which has been running MULTICS as a service within the community since Oct 69, is addressing the operational characteristics continually. In addition, manpower and time constraints imposed on this in-house effort forced a compromise which favored placing the emphasis on determining what potential gain can be obtained in using MULTICS to solve A.F. future data processing requirements.

TABLE OF CONTENTS

SECTION I

INTRODUCTION

This report documents the experience obtained by RADC personnel in running the MULTICS operating system on the GE-645 computer. MULTICS has been under development at the Massachusetts Institute of Technology for several years. The development is being sponsored by the Advanced Research Project Agency.

MULTICS (Multiplexed Information and Computing Service) is a comprehensive, general purpose programming system. It was designed to meet present and near future requirements of a large computer utility. It is presently running at MIT seven days a week, 18 hours a day providing a large variety of service demands from man-machine interactions to batch processing. It is an evolving system which is continually being updated with improvements under development at MIT.

The objectives of MULTICS are:

1. To provide a user constructed, hierarchically organized, day-to-day information base.

2. To provide both interactive and batch processing with mobility between these modes.

3. To allow multiple processors and parallel processing.

4. To enable symbolic referencing to I/O devices with access control for protection of I/O streams.

5. To allow the administrative system to delegate resources allocation and to provide on-line documentation and on-line consultation.

6. To provide continuous 24-hour operation with an on-line debugging system and a comprehensive access protection system.

MULTICS' great advantage over other operating systems is its open endedness with respect to program size and system capabilities. The heart of MULTICS is a hierarchical file system that gives each user a virtual core memory of unlimited size and allows powerful file creation and manipulation. A traffic controller handles the allocation of hardware resources to the various states of users' executions (processes). The virtual memory feature is heavily dependent on special hardware in the 645 that aids paging and segmentation. As a result of the MULTICS design, each user is provided with the following capabilities:

1. symbolic virtual memory
2. symbolic input/output
3. interprocess communication
4. file sharing via pure procedure code
5. dynamic linking, making size specifications unnecessary
6. comprehensive file access control
7. continuous file backup

MULTICS is a suitable and even hospitable environment for large system programs such as compilers and data managers. It offers many aids to such programs, but does not require them to use these. This allows running of systems essentially independent of MULTICS. By being open ended, MULTICS can accommodate more different kinds of jobs. There are no intrinsic size limitations on either the supervisor or the programs running on the system. MULTICS will also provide a diversity of languages and language compilers to each user.

The following concepts are used in MULTICS:

1. User - A person in physical control over a remote input-output terminal who issues "commands" to the system which performs some action in his behalf. Absentee users (unattended terminals) are possible.
2. Process - Sequence of actions running on and controlling a processor.
3. Segments - Subsets of processes that are linear arrays of elements (procedure/data) in main memory.
4. Files - Segments in secondary storage.
5. Paging - Hardware/software complex to allow non-contiguous blocks of main memory to be referenced as a logically contiguous set.
6. Distributed Operating System - A user process is organized by MULTICS to consist of the user's procedures plus MULTICS modules required to carry out his tasks.

7. Dynamic Linking - Segments are brought into main memory only when needed; each procedure has a linkage section, and this is used to link symbolic references between segments the first time that the reference is encountered during execution.

8. Process Protection - In a time-shared environment such as MULTICS with a distributed operating system there is need for protection of both user and system segments (files). This is done by a number of mutually exclusive subsets called RINGS, that are concentric in design. Innermost Ring 0 contains system files and procedures. Outer rings contain user procedures. There are gates, passwords and traps to enable users to traverse the rings with proper authority.

9. Pure Procedures - Contain code that is not modifiable during execution. MULTICS is written this way so that users can share the same copy of MULTICS modules. Also user procedures and data can be shared.

10. MULTICS consists of the modules:

    a. Supervisor
    b. Traffic Controller
    c. File System
    d. I/O System

SECTION II

RADC MULTICS EXPERIMENT

2.1 The MULTICS project at RADC was initiated in January 1969. The preliminary goal of the project was to investigate the MULTICS activity at MIT to determine whether the system, in part or whole, could be exploited for Air Force use. This preliminary investigation included reading MULTICS documentation, visiting the designers at MIT, and establishing in residence at MIT an RADC representative.

From this study, in-house personnel acquired an understanding and appreciation of the value of such a system for military data processing needs. Experience with military systems pointed out several areas which could stand improvement: manipulation of large files, balancing work loads, satisfying a large number of users, performing a variety of processing tasks with one computer system, cost of program generation, security, and creation of specialized application programs. In matching these needs against MULTICS, it appeared that the development goals of MULTICS could offer solutions in these areas. Several questions were posed on whether such an ambitious design could be implemented in a military environment. Cost, reliability, training, maintenance were factors that needed to be addressed.

In October 1969 MULTICS was declared operational in the MIT environment. This event along with our observation of the

system running at MIT inforced the position that MULTICS could be a candidate for Air Force exploitation. In December 1969, it was decided that RADC undertake to evaluate the suitability of MULTICS as an operating system for the RADC-GE 645 computer. The purpose of this evaluation was to accomplish the following:

a. Determine the advantages MULTICS provides to our major R&D programs (e.g. data management, security and access of large files, design of system software).

b. Determine the resources needed to support and maintain MULTICS at RADC.

c. Determine whether MULTICS performs as theoretically claimed.

2.2 A plan for conducting the evaluation was established. The plan included provision for the following:

a. Installation of a terminal to the MIT MULTICS system.

b. Training of in-house RADC system programmers.

c. Acquisition of additional computer hardware.

d. Creation of application programs.

e. Creation of operational test programs.

f. Reporting results and conclusions.

See Figure 1.

Installation of a Terminal.

In January 1970 arrangements were made with MIT to get on the MIT-MULTICS-service. One hundred pages of storage was assigned to the RADC Directory (page is equivalent to

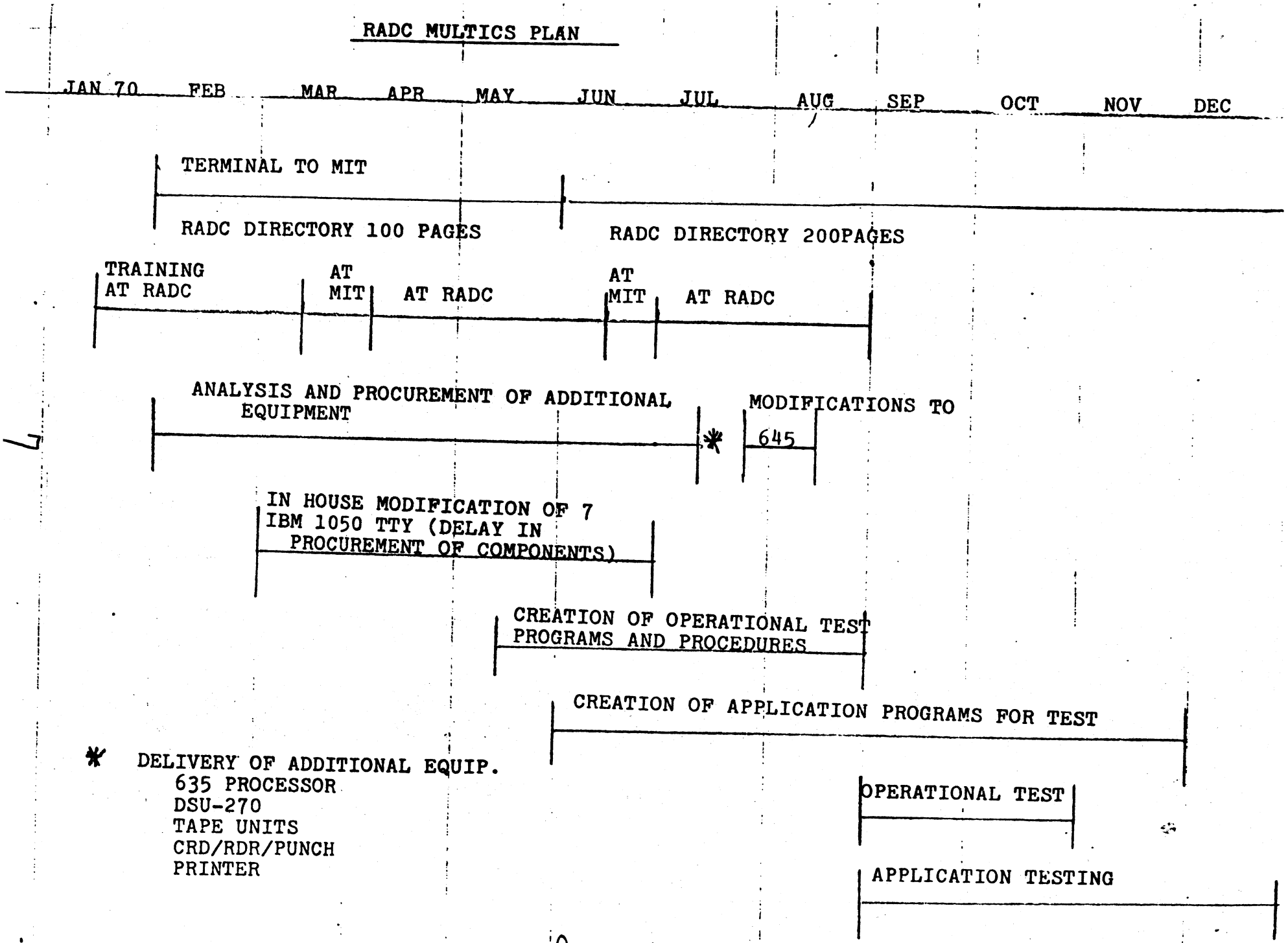
RADC MULTICS PLAN
JAN 70
FEB
MAR
APR
MAY
JUN
JUL
AUG
SEP
OCT
NOV
DEC
TERMINAL TO MIT
RADC DIRECTORY 100 PAGES
RADC DIRECTORY 200PAGES
TRAINING AT RADC
AT MIT
AT RADC
AT MIT
AT RADC
ANALYSIS AND PROCUREMENT OF ADDITIONAL EQUIPMENT
*
MODIFICATIONS TO 645
IN HOUSE MODIFICATION OF 7 IBM 1050 TTY (DELAY IN PROCUREMENT OF COMPONENTS)
CREATION OF OPERATIONAL TEST PROGRAMS AND PROCEDURES
CREATION OF APPLICATION PROGRAMS FOR TEST
OPERATIONAL TEST
APPLICATION TESTING
* DELIVERY OF ADDITIONAL EQUIP.
635 PROCESSOR
DSU-270
TAPE UNITS
CRD/RDR/PUNCH
PRINTER

figure 1

1024-36 bit computer words). In March of 1970 the RADC quota was increased to 200 pages. The purpose of the terminal was: to get hands-on experience with MULTICS, to train in-house system and application programmers and to develop operational and application test programs; all prior to the delivery of the RADC system.

Training.

To prepare for the test a training program was initiated which would not disrupt other R&D activities being pursued by the group. Two RADC employees were sent to MIT for a three-week period to receive extensive training on the philosophy of the MULTICS system. This was followed with six weeks of in-house training using the terminal to MIT. The same two employees with the addition of one engineer were sent to MIT for another two weeks to receive training on implementation of MULTICS and crash analysis. This was followed by several weeks of in-house training using the terminal. In June of 1970 three co-op students from RPI joined our group. They were given in-house training on MULTICS by the employees who were trained at MIT.

Hardware.

Arrangements were made to modify the RADC-GE computer installation so that the RADC-MULTICS activity would not disrupt the GECOS III operational requirements or the TSS/645 time-sharing service. The final configuration resulted in a GE 645/635 system which could be operated as two independent 128K core

systems with certain peripherals shared or as an independent 645 or 635 system utilizing 256K core for experimentation or operational use. In addition in order to accept the MULTICS software, several minor hardware modifications were made to the GE 645 by the GE field engineers. Another problem faced was providing suitable terminals required for the MULTICS experiment. The MULTICS system uses IBM 1050, IBM 2741, or Mod 37 terminals. Seven IBM 1050's were acquired from another project and suitable modifications were installed by RADC computer facility personnel to make them compatible to the MULTICS system. A hardware configuration chart is shown in Figure 2.

Creation of Application Programs.

The application testing consisted of designing and implementing several application programs characteristic of current and future military data processing requirements. The objective of the evaluation was to determine the value of MULTICS for: handling large file manipulations, providing resource sharing among users with unpredictable size jobs, performing data management functions, satisfying requirements for security and protection in a multi-programming environment, and providing high productivity in the generation of system and application programs.

Six separate "application" studies and programs were initiated and completed. They are: (1) TRAC Implementation,

RADC GE 645/635 CONFIGURATION

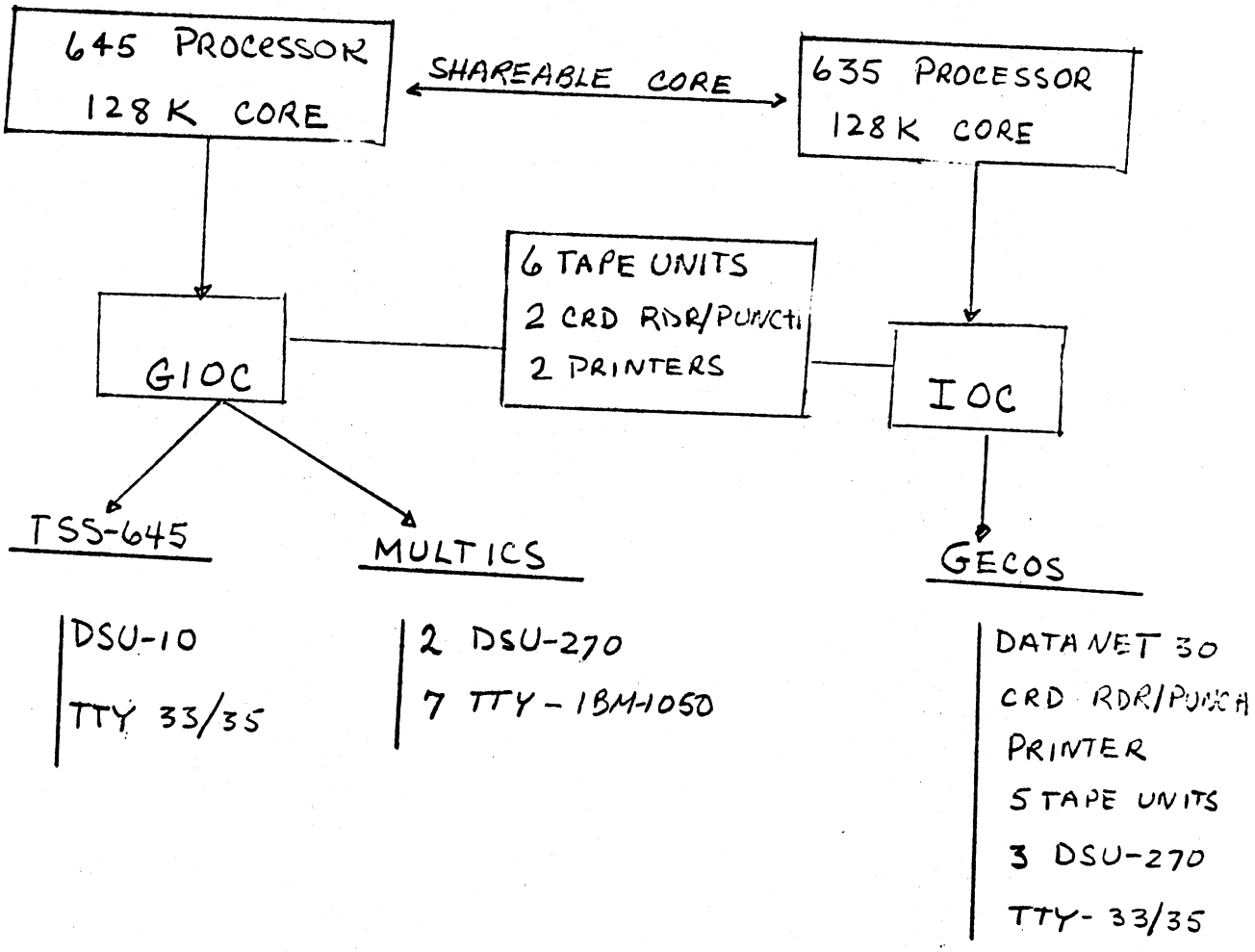

figure 2.

(2) Multi-segment Data Base Referencing, (3) On-Line Retrieval System, (4) Data Management Model, (5) Security Subsystem, (6) Comparison TSS/645 with MULTICS. These jobs, with the exception of "TRAC Implementation", are described in detail in the following sections. All of these application jobs place a heavy demand on an operating system for assistance during the creation of the respective subsystems. For conventional operating systems, these jobs would require much more effort than was actually expended during this evaluation.

(1) TRAC Implementation.

The TRAC implementation (text reckoning and compiling language) is representative of the data processing needs of large AF computer installations for adding software subsystems to existing operating systems. The task was initiated to observe the amount of effort required to add a complex software subsystem to the MULTICS system. TRAC is a string manipulative interpreter which provides a user with an on-line capability to perform string editing functions. The subsystem was written utilizing 2000 PL-1 statements. It took 200 hours to analyze and code the system; an additional 120 hours were expended at the terminal debugging and checking. TRAC has previously been coded by a contractor for both the CDC 1604 and GE 635 computer. Each effort took three months. Both of these versions do not have a file interface for the processor which would allow a programmer to store his programs or data. A study of providing this file interface under GECOS

for the GE-635 was made and found to be too difficult. Detailed knowledge of GE-635 machine language with its constraints and limitations was necessary and made the job almost impossible. Under MULTICS this capability was completely provided in three days without resorting to machine level code. This large task also demonstrated that communication between programs is almost an absolute necessity when developing an R&D piece of software. In systems like GECOS III, tailor-made interfaces have to be written to acquire program to program communication, this necessitates that all jobs to be interfaced be known in advance and further, all are to be loaded together. This is a particularly severe constraint for an R&D activity for which one cannot predict what jobs would be needed at execution time. This problem disappears when using MULTICS; interfaces are present and need not have to be a concern of the programmer for program to program communication.

(2) Multi-segment Data Base Referencing.

Previous staff studies of AF data processing requirements have uncovered the existing complexities and difficulties involved in handling large files of military information. The multi-segment data base referencing problem was designed to observe the procedures required to allow a user to view a multi-segment data base as a continuous block of data. This job demonstrated the ease with which a system programmer can on-line create, initiate, and link segments totaling up to 30 million words. The virtual memory concept with dynamic

storage allocation (on demand), presence of built-in based variables and pointers and the capability of symbolic referencing, reduced this unusually tedious task to a relatively simple one. The effort took two weeks to write and completely check out.

(3) On-Line Retrieval System.

The On-Line Storage and Retrieval System is a program which is representative of a broad spectrum of problems which involve querying and editing a data base. The extensive capability of the MULTICS system and the use of a powerful high level language made this programming task seem quite simple. Three weeks were required to write and debug the program. The program allows simultaneous on-line users to create, initiate, delete, update, and print out a large multi-segment data base. The job demonstrated the extensive debugging aids imbedded in MULTICS, the ability to reconfigure program procedures on-line, and the value of being able to invoke I/O routines simply by making symbolic references to a segment. It is important to note that the programmer did not lose any information during the course of the investigation, demonstrating the high reliability of the MULTICS file system with its file backup capability.

(4) Data Management Model.

The Data Management Model task was initiated to demonstrate the assistance MULTICS can provide for the

building of a data management system. A prototype model was implemented using the DM-1 philosophy of data management functions. DM-1 is a generalized data management design presently being implemented by RADC on the GE 635. It features a data and data description independence which permits a user to logically redefine a data base for solving his particular data processing application. The following MULTICS features were exploited in the MULTICS/Data Management Model.

a. User interactive environment providing multiple simultaneous update/access.

b. Inter-job communication.

c. Dynamic storage allocation (on demand).

d. Data packing.

e. Built-in functions for string handling.

f. Built-in functions for pointer manipulation.

g. Dynamic linking within the file system.

All of these features permitted the system designer to write in a higher level language a modeling package that can be used to experiment with various strategies for dealing with the data management functions of: data description, data structuring, access control, data entry, item definition and data administration. The model was created in three man-months. An estimation based on contractors experience of implementating DM-1 on GE 635 indicates that a package consisting of the same features implemented under GECOS would have taken 9 to 12 man-months. The difference is mainly due to the additional power

of MULTICS over GECOS III.

(5) Security Subsystem.

One of the urgent problems existing in a large computer system servicing a general community of military users is the protection of user files and the necessity for protecting classified information. A task was initiated to create a security subsystem which would provide on an optional basis protection of computer information beyond what an operating system provides for users. The intent was to observe the features of MULTICS which would be used to create the subsystem. The following features not usually found in an operating system were found to be required in order to create a cost effective system:

a. ring structure

b. lock and unlock procedures

c. inter-process communications

d. dynamic linking in the MULTICS file system

e. built-in sharing of procedures and data

In a conventional operating system like GECOS all of the above features would have to be imbedded in the operating system. The complexity of such a task is so great it is highly probable that is cannot be accomplished. One thing is certain, if it is possible the cost would be prohibitive.

(6) Comparison TSS/645 and MULTICS.

All of the jobs reported previously can be classified as those that make heavy demands on the resources

of a computer. It is very difficult for someone who has never made heavy demands on a computer system to distinquish between an operating system which satisfies these demands and an operating system which provides only Basic, FORTRAN and an editor. To satisfy the heavy demand user, the MULTICS design has a very large overhead which cannot be appreciated by the non-demanding user. In fact the non-demanding user may become frustrated with the large number of commands which to him are useless. Because of this overhead, responses to the non-demanding user will not be as fast as those attained with simpler operating systems.

In order to assess the utility of MULTICS to a non-demanding user, a comparison was made to TSS/645. The comparison was based on Basic and FORTRAN programming. The responses in MULTICS were slower than TSS/645 but still within the acceptable "think" time interval, less than five seconds. On the other hand, the following advantages of MULTICS were observed:

a. sharing of programs and data by several users.
b. ability to specify who can use data/program.
c. ability to "archive" or pack files in secondary storage using a single command.
d. file backup - restore to conditions just before a user error or system error.
e. all subroutines are external and are

compiled separately, making it faster and easier for debugging.

f. main system editor which operates independently of line numbers.

g. "Map" and "List" options available at compilation to simplify debugging.

It can safely be predicted that even non-demanding users will eventually appreciate the additional power MULTICS can provide them.

Creation of Operational Test Programs.

The operational testing was performed to gather information on the maintainability, reliability and performance of the MULTICS system. An examination was made of the current work load of programs in the RADC computer facility to determine the kinds of programs to be used for testing that would be representative of a typical computer installation. Sixteen jobs were designed and programmed by in-house personnel. They were grouped into three test packages or scripts: Basic (four programs), FORTRAN (six programs), PL-1 (six programs).

The operational testing consisted of running a series of 30 tests on the system and recording timings of each interaction, compilation and execution. The time data was averaged into two categories: average interaction time/test and average compile time/test. The parameters for each test were: number of users on line, script package under test (PL-1, FORTRAN, Basic), and the memory configuration (128K,

192K, 256K). During the course of the evaluation, logs were kept to record crashes and downtime. A detailed description of the operational testing and results obtained is given in Section IV.

MULTICS was installed at RADC the weekend starting 28 August 1970 with the help of two consultants from MIT. On 1 September 1970, operational testing and application testing of the RADC-MULTICS version began. MULTICS was scheduled 3 1/2 hours/day from 1 September to 31 December 1970 (end of test period). One half of this schedules time was devoted to operational testing with the other half used for application work. Operational testing was completed 31 October 1970, after which all schedules MULTICS time was used for application work.

The RADC-MULTICS team consisted of one Honeywell engineer, one RADC engineer, two RADC senior programmers, one RADC junior programmer and three co-op students from RPI. The RADC engineer maintained the operational tests. The senior programmers were assigned primarily to application testing. The three co-op students and the junior programmer came on the project in June 1970. After four weeks of in-house training on the system, they were assigned to write several of the script programs used in operational testing. They also made important contributions in the application area. The three co-op students returned to school in September 1970.

2.3.1 Value to future AF data processing:

The advantages that MULTICS provides over conventional operating systems for large file manipulation, on-line retrieval systems, security procedures in a multi-programming environment and data management systems are covered in detail in Section III. The following features of MULTICS, which are not found in conventional operating systems, were used and observed in the application studies. They clearly confirmed the definite advantages MULTICS provides for current and future data processing needs.

MULTICS provided a time-sharing environment which made available to all the simultaneous on-line users all parts of the computing system which includes: processor, memory, peripherals and system software. All of the system, which appeared to be dedicated to each user during his computing cycle, was available without the user having to input any part of the system. This experience was new in that with other systems special interfaces must be written by the user and must be loaded in core at the time of problem input to insure that special hardware devices and software packages will be available at time of execution. This programmer loading concept presents a severe constraint in the building of software systems where behavior cannot be predicted. For example, the on-line retrieval application demonstrated that a user must be responsive to system outputs which cannot be predicted and are too numerous to effectively pre-load all possible outcomes.

In MULTICS, with its facility for symbolic referencing, a user can obtain any system resource (software or hardware) simply by calling the item by name. The concept of "loading system programs" along with users programs is non-existent.

The MULTICS operating system provides to the user all the software support he needs at the time of execution. This utility is very advantageous for the work in data management. General data management system designs rely on operating systems to provide the software interfaces to handle I/O to secondary storage, external files, or other users. Conventional systems like GECOS III do not have this utility necessitating that the interfaces be built into the data management package. Since this utility was present within MULTICS, the data management application package was reduced in size and permitted the application programmer to concentrate on his data management functions never concerning himself with system support or machine-dependent functions. The implementation of the string language interpreter (TRAC) was a good example in which a programmer with very little knowledge of MULTICS was able to add this subsystem (2000 PL-1 statements) with relative ease.

An important characteristic of an operating system is its ability to enable the user to view the computer at a conceptual level and prevent the necessity of delving into internal mechanisms of the computer to get solutions. Conventional operating systems fail in this respect for most of them intro-

duce additional constraints and complicated formats to be interfaced with, hard to use compilers with cryptic error messages and file systems that have to be contended with rather than simply used. A good example was found in the TRAC Implementation Application which investigated both MULTICS and GECOS for their utility as development systems. It was concluded that a system without coherency of design and sufficient system aids such as standard command languages, flexible call and return sequences, actually slows down development work and redirects it to improving the operating system rather than solving the original problem. In this application the programmer could not cope with the complexities of GECOS and turned to MULTICS where he was able to concentrate solely on his application. All of the application studies demonstrated that sophisticated system and application programming could be accomplished using the MULTICS system without the programmer having to be an expert in the workings of MULTICS.

In the majority of current time-sharing systems, program size is limited. This is due to the critical need that executing programs are to be in core at time of execution. Since core is expensive and is fixed in size for most installations, the number of programs that can be loaded is depended on their size. This constraint reduces the time-sharing service to one that only accommodates the short programs. Not only do the programs have to be short, they must be fixed in length

thereby eliminating those programs in which size is unpredictable. For example, large file manipulations and on-line retrieval systems where program size is dependent on real time responses, and compilations where object code size cannot be predicted. In MULTICS no restrictions are imposed on size of programs. This is accomplished through "unlimited memory", achieved by paging and segmentation hardware and the dynamic linking capabilities of the MULTICS file system, which is essentially a disc extension to main memory. This feature allowed the processing of unpredictably large problems. For instance, the PL-1 compiler which is composed of approximately 150K words is handled effectively. Conventional operating systems where compilers must be core resident could not handle compilers of this size. The FORTRAN compiler which consists of approximately 20K words is presently overburdening current time-sharing systems. On a system like GECOS III, one FORTRAN compile would severely degrade the time-sharing service.

The "unlimited" or "virtual" memory concept made it possible to handle large files in the order of 30 million words without using sophisticated overlay techniques. The on-line retrieval application demonstrated the ability to update, modify, retrieve any portion of this large file while other user jobs were on the system. This feature created an automatic environment which permitted between 7-15 users to be on-line without regard to what each user required for computing support, i.e., compilations, large file handling, command language

interpretation, or simple Basic type programs.

The MULTICS system is totally written in PL-1. The PL-1 compiler is reentrant and it generates reentrant, recursive code. Because the system is written in PL-1, maintenance and debugging of the system is performed at the source level which represents a large savings in time over machine code maintenance and debugging procedures. This savings was demonstrated during crash analysis for system breakdowns experienced in the experiment. The fact that one Honeywell man and one RADC man were able to handle 1:5 million words of operating system is evidence for the utility of the system being written in a higher level language.

The concept of a higher order system programming language integrated in the MULTICS system relieves the programmer from having to write interfaces for system support. This utility demonstrated a much higher program productivity than could be attained with conventional systems like GECOS III. The TRAC language processor was written in five weeks using MULTICS, whereas it took three months to write it for GECOS III. The multi-segment data base referencing system took two weeks to write. A similar job under GECOS III would take several months. The Data Management model took three months to incorporate using MULTICS. A similar effort under GECOS III took nine months utilizing twice as many programmers.

Because all system and user programs are represented in the system in reentrant code, it can be made available to all

users without giving a copy to each user thereby reducing core requirements and errors in data transfer. This reentrant capability is an absolute requirement for data management systems where sharing of data files and user programs represent the major task required of the system. It permits maximum use of the computing system resources thereby increasing the number of on-line users. In addition file maintenance is performed on only one copy and is more easily administered for controlled dissemination.

MULTICS provides an interactive command language which has imbedded into it built-in functions for obtaining listings, debugging aids, segment manipulation, editing procedures. All of these built-in functions are accessible at the programming level.

The MULTICS file system provides the means for accessing secondary storage in machine-independent and device independent fashion. The user is aware only of sybmolic addresses. All physical addressing is performed by the file system unseen by the user. The user need not know how or where a file is in storage. The user sees infinite storage capacity and all files appear to be stored on-line. This built-in file system provided by MULTICS proved to be a great asset in our application studies. In particular the concept of "binding" or restructuring data bases for specific information handling

applications is a burden which the data management system must provide when working with conventional operating system software. Using MULTICS the data management model implementation created in-house, eliminated the binding facility because the operating system with its file system already possessed this facility.

The work performed in adding TRAC to MULTICS also highlighted the value of the file system. TRAC is an interpreter which operates in an interactive mode. In this form TRAC does not have the ability to save work performed at each session for continuation at a later time. Users had indicated a file interface to the system would enhance their string applications. Investigations of providing an interface with GECOS III indicated a series of complex modifications which were beyond the capabilities residing in-house; in fact it never was concluded that this work would be possible. This file interface capability with TRAC operating under MULTICS was designed in three days by a system programmer. The procedures for handling the interface already existed in MULTICS and the time expended for design was essentially devoted to getting familiar with MULTICS.

In an environment of sharing files and programs, the problem of security of files from other users is of utmost concern. In MULTICS each file has its own independent control for access. Access lists ordered by user identification indicates whether a user can read, execute, append, or write

for each file. In a conventional file system like the GECOS III file system access control is provided at a number of different hierarchy levels. If a user has access at a particular hierarchy level, he has access rights to all files residing at that level. In our study of providing protection for classified data, we found that MULTICS access control permitted the addition of additional software to obtain access control at the item level within a file.

Equally important in the security area is the protection of the system software from either accidental or intentional damage. The placing of system parts into a structure of concentric rings or levels where access rights at a higher level can never access a lower level insures system protection.

Probably the most noticeable feature of the file system during our experience was the ease of recovering files after system crashes. This was attributed to the file backup system provided in MULTICS which automatically records all updates to the file area.

2.3.2 System Reliability

From 1 September 1970 through 30 December 1970, the system has been running every day with very few problems. In four months of operation there were 28 crashes and in a majority of the cases fixes and restoration of the system were completed in less than 25 minutes. During this time, information was lost only four times, where in three of these

periods the information loss was minor. This can be attributed to the high reliability experienced with the file system. Mean-time between failures was nine hours with the longest uptime of 38 hours. Throughout the test period approximately 262 hours of uptime and 11 hours of downtime were recorded resulting in 4.2% downtime. No serious faults were found with the operating system, including the PL-1, FORTRAN and Basic compilers.

These results show that MULTICS can be supported at RADC and that it is a stable and reliable system.

2.3.3 System Maintainability

The MULTICS system currently running at RADC has not been updated with the changes that have been made by MIT since September 1970. The RADC system has been stable thereby minimizing the amount of crash analysis to be expected during the experiment. The results which show that system software faults occurred infrequently and that in each case were discovered and fixed locally may not be representative for the environment which must respond to weekly updates of the system. The question of whether the RADC environment should be one to be responsive to updates depends on whether MULTICS will be supported by Honeywell. During the experiment, one RADC and one Honeywell engineer effectively handled all the software and hardware problems experienced. It can be recommended that with a stable version of MULTICS, previously checked out in the MIT

environment, two well-trained engineers can maintain the system.

2.3.4 System Performance

From the start of our operational testing, it had been concluded that we could not demonstrate cost-effective computing with the RADC-MULTICS. Certain software and hardware system elements which have a direct bearing on system response and the number of users on-line were not part of the RADC configuration. It was for this reason that more emphasis was placed in the application testing rather than the operational testing.

The results from the operational testing, however, demonstrated a much better performance than expected. With a 128K memory, more than seven interactive users (no compilations) could be serviced with response times not exceeding five seconds. The upper limit could not be determined due to the fact that only seven terminals were available for testing. This configuration also showed that five Basic users with compiles could be serviced simultaneously or three FORTRAN users with compiles or two PL-1 users with compiles. Increasing the memory to 256K produced results that essentially doubled the number of users on-line. These results can be considered conservative since the script programs did not allow for "think time." Actual experience with PL-1 application programming on the 256K system indicated seven PL-1 users could be serviced with acceptable response times.

The test results demonstrated that an approach to cost-effective computing can only be attained with a 256K main memory configuration. System overhead made the 128K main memory system too sluggish, especially for the heavy demand user requiring compilations. The need for approximately 65K of resident memory for MULTICS reduces available core space for executing pages, thereby increasing the paging traffic which is direct overhead. The 256K memory reduced this paging traffic to make the system more efficient. The configuraiton at RADC differs from the MIT system in that a magnetic disc (DSU-270) is the paging auxiliary memory instead of a high speed drum. The speed of the drum is twice as fast as the disc and is reflected in the number of users that can be satisfactorily serviced simultaneously. In the RADC 256K system it is projected that 15 users (mixed) could be serviced while at MIT it has been reported that 40 users can be serviced. In addition it was found that as the system gets more use the need for auxiliary memory increases to hold the increasing amount of user programs. With four RADC users working with application studies in information processing, the total of MULTICS system programs and user programs exceeded the capacity of a DSU-270 (2.5 million words), necessitating the need for an additional disc. It is recommended that in order for MULTICS to be effective as a tool at RADC for application programs and still offer services to non-demanding users, the hardware configuration should have a minimum 256K core and

two DSU-270 discs.

The experience obtained in-house with the MULTICS experiment has been beneficial toward gaining a better appreciation of current and future military data processing needs. It is hoped that the application work performed can be continued to further explore the relation between operating system development and data processing applications. The data management model represents an excellent start on investigating cost-effective procedures for constructing software pachages which will perform data management functions in an on-line mode. The full impact of file sharing, simultaneous writes in same file, real time update, restructuring of data files on-line, need to be examined thoroughly. The importance of security in a multi-programmed environment cannot be questioned. The preliminary results obtained in the current study clearly show that this area should be pursued further. The ability to add large complex subsystems to the MULTICS system without making extensive alterations to the executive as demonstrated in our work should be of interest to some of the research programs residing in the division. In particular the On-Line Pattern Recognition System (OLPARS) developed at RADC should be examined for implementation under the MULTICS system. The ease with which we implemented complex software packages should be exploited for developing more in-house system and application programs which presently are being programmed by contractors. The system should be further examined for use

as a modeling tool to investigate new computer designs, network of computers, and design of specialized software subsystems.

The experiment, in addition to gaining knowledge of a well thought out operating system design, provided an appreciation and understanding of: test procedures for evaluating software, PL-1, system programming, "real" time-sharing environment with a mix of "foreign" users, services required of an operating system, hardware and software support required to maintain a large computer system.

3.1 MULTICS and Data Management

3.1.1 Introduction

One of the long-range goals of the MULTICS investigations at RADC concerned an assessment of the specific value of MULTICS to the Air Force in the area of Data Management. To meet this goal with a high degree of credibility it was decided that a demonstrable Data Management (DM) System be implemented in the MULTICS environment. This would not only necessitate a comparative study of DM functions and determine whether MULTICS is or is not a suitable host for them, but would also result in a viable software system capable of manipulating a "live" data base.

3.1.2 Data Management Functions & The MULTICS Environment

Process Control

A process can be thought of as the locus of control within an executing program. As such, a process has a unique system identification as well as attributes of state such as running (executing), ready (to run) and blocked. Another important characteristic of a process is that it is capable of communicating with the outside world; that is, with I/O streams and other processes (both user and operating system generated). The I/O streams are interesting because they are the communication mode that the user takes advantage of to monitor the performance of the process.

In MULTICS, the normal process I/O stream is to the interactive console typewriter. Thus, facilities for formatted and unformatted I/O to the console are integral parts of the

system, much as a card reading routine would be in a conventional operating system. In conventional O/S's, software (usually in assembly language) must be written to interface the program with a remote communications processor and then to the console. Necessarily, a portion of this programming will involve time consuming code conversions (ascii to BCD and vice versa).

Data Management Systems must be capable of establishing interactive communication links between users and processes, that is, executing programs that represent the DM system functions. In MULTICS, the implementor of such a system has the software tools for effecting such communication at his disposal. These tools are accessible from a higher level language, namely PL-1. Thus, MULTICS provides a natural communication environment for a Data Management System.

So far, the discussion has been centered on the ability of one user to communicate with an executing program. Recalling that the process is a program in execution and assuming that the program is some DM system function, it is important to know if several uses can communicate with the program simultaneously. DM systems must be capable of supporting a community of users who are concurrently interacting with a data base. This includes the condition that several of the users might be doing the same type of work (e.g. defining a data base, printing a file, etc.).

In a conventional operating system the traditional approach to creating a community of users who are interacting with a software subsystem usually begins by generating a copy (from a disc stored "master") of a particular functional module upon the request of each user. This copy is then loaded into core and the user is allowed to interact with the program under the guidance of an "exec" for this particular subsystem. The "exec" is needed in addition to the operating system in order to control the activities of the various users of the subsystem, facilitate inter-module communication, and prevent certain conflicts from arising among the users i.e., usually multiple simultaneously writing to the same secondary storage location. This approach certainly works, but it should be clear that it it quite expensive. When several copies of the same information occupy core the number of potential users interacting with the subsystem becomes limited because total core storage is limited. The alternative is to build a "swapping" time-sharing system, but then the advantages of multiprogramming are lost. The performance of such a system is directly proportional to the speed of the secondary storage device used for swapping. It is also the nature of these systems to be written in the hierogliphics of assembly language and to be quite complex.

In MULTICS this "traditional" approach to system writing has been abandoned in favor of the obviously superior "sharing" concept. Sharing involves the use of system-oriented compilers that generate pure (non self-modifiable) code which can be

entered, abandoned and reentered at any location. Thus, any number of users may "execute" through the same program simultaneously once again under the guidance of an "exec", but, in this case, the MULTICS exec. The advantages are many. Precious core is not wasted with superfluous copies of the same program. The subsystem writer does not have to write his own "exec". The MULTICS system handles the affairs of scheduling, communications, etc. The subsystem is much simpler to design, code and debug because the problem to be solved is foremost in mind. Thus, MULTICS again, provides a natural environment for establishing a community of users interacting with a data base, many of whom are actually using the same program simultaneously. And this is all done without extra programming - an automatic result of the sharing concept.

Hierarchy Management

Probably the most important aspect of the MULTICS concept of system writing involves the thorough integration of the executive with the file system. This planned design allows for bringing the normal user into intimate contact with the MULTICS file constructing primitives in a very simple and elegant manner. Gone is the typical assembly language interface of most operating systems to do no more than "open" disc space for a temporary time. In fact, most contemporary operating systems do not allow the user subsystem to use the file system constructing primitives at his discretion. The subsystem writer usually allocates a large block of disc storage prior to execution and then creates and manages his own logical file structure within that space. Since these

files are known only physically to the operating system as a gross block on disc, such strategy calls for the subsystem to support software identical in logic to that of the executive for creating, manipulating and deleting these files. This strategy results in a rather obvious waste of effort in conventional system building.

The MULTICS system allows the executing process to create, modify and delete files which are both physically and logically known to MULTICS. Thus, the system primitives of MULTICS can be called upon to do the above mentioned operations on these files. This realizes a significant savings to the builders of file oriented systems, in that they avoid the writing and maintenance of this conventional "overhead" software. In addition, MULTICS offers the standard file system error handler, a tool that replaces the assembly language interface of conventional operating systems for checking the status of file system operations.

As mentioned above, it is the total integration of the file system and the executive in MULTICS that allows users to execute the same file system primitive function as MULTICS itself uses. This integration includes an Access Control System which is in force to prevent users from mismanaging parts of the file hierarchy to which they have been denied access.

Because MULTICS knows both the logical and physical identity of all files any application system (e.g., a DM system) automatically becomes dynamically responsive to its users' particular storage requirements at a given time. Thus, when secondary storage is freed from some part of a DM system under MULTICS,

that storage becomes immediately available to other users of the DM system and to all MULTICS users. This fact cannot be replicated in conventional operating systems, but is an automatic result of the MULTICS File System concept.

Storage Management

Probably the most technically adroit feature of MULTICS is the virtual memory concept. Although the idea of a main memory limited only by the size of secondary storage initially presents visions of programs without size constraints, there is an equally powerful, if more subtle benefit of this philosophy. This is the fact that any word in the virtual memory is directly addressable. Coupling this with the concept of an integrated system programming language (PL-1), such addressing becomes symbolic. This provides a unique and powerful tool for system programmers, because it relieves them of any concern for the physical management of secondary storage. The disc management problem has always been a sizable drain on project programming resources and has caused the diversion of designer efforts from the actual problem to this tangent.

At the heart of a Data Management System is, of course, the efficient storage of data. It follows then that the operating system under which a DM is to be implemented must provide tools for the system programmer to easily accomplish the storing of data. In general, the MULTICS Virtual Memory provides the solid base for symbolic management of data storage areas.

Specifically, the MULTICS system programming language, PL-1, provides data structures and built-in functions for manipulating data pointers and information strings. These tools when used in combination provide a powerful storage management technique as explained in detail in the following paragraphs.

PL-1 data structures are comprehensive and flexible tools for managing storage space. In addition to the "static" type of storage used by most conventional compilers, the MULTICS PL-1 implements "push-down stack" storage and "based" storage.

"Stack" storage has the advantage of efficiency, since storage is allocated only when needed and deallocated when not in use. "Based" storage has the advantage of being allocated by the user at run time only if the user desires and only in the quantity that suits his requirements. Based storage provides the basis for efficient disc space management and is utilized in two ways. For "constant" type information, as usually found in the system directories of a data management system, a based PL-1 structure representing the elements that describe the data base is used. This type of structure can be declared so as to provide very fast access to the system directory variables. For the storing of the actual data in the data base, a based PL-1 structure that is an arbitrarily long bit string is used. Here, any degree of data packing density can be achieved, since the programmer has control of the information down to the bit level. Thus, a complete set of tools for the description of data is available in MULTICS.

To manipulate data and to actually place it into the based bit string described above, MULTICS PL-1 provides a group of string-manipulating functions as built-in parts of the language. Thus, functions for concatenating (which is also an operator in the language) strings, determining the length of strings, accessing sub-strings of strings and locating individual elements of strings are available, as well as functions for converting items to bit string representation.

To effect the actual storage of the data the based data structures described above are associated with pointers to the segments of their residence at run time. MULTICS PL-1 provides various built-in functions for manipulating the pointers. Thus, functions for obtaining the address of a word in storage, for creating the "null" pointer and offset, for adding the offsets of pointers and for extracting offsets and segment numbers from pointers are available. Thus, a complete set of tools for manipulating the pointers are at the disposal of the system programmer.

In order to obtain the pointers the MULTICS File System provides the necessary primitive calls for creating, modifying and deleting segments (files). In addition, entries into the File System are provided to obtain the pointer to a file simply by referring to the file by symbolic name. All of these calls to the File System are seen by the programmer as PL-1 procedure calls, thereby, avoiding the usual digression into assembly language to perform these functions.

Thus, the combination of PL-1 data structures, pointers and string handling functions provide all the necessary tools for the efficient storage of data in a DM system. In addition, it should be clear that structures and pointers can also be combined to provide a symbolic list processing capability. Thus, linked lists of pointers can be used to interconnect the data allowing for a flexible and efficient means of random access to data (as opposed to a purely sequential scheme) and of re-arranging (adding, inserting and deleting) the logical order of the data.

File Backup

It has been known for some time that an indispensible function of a data management system is the protection of the information stored in its files from system failures. Such an automatic backup facility has been designed as an integral part of the MULTICS system. Thus, the backup function is run as a special process at regular time-intervals as specified by the system administrator. Its function is to record on magnetic tape all information files that have been up-dated since the last time it was run. Thus, in the event of a system failure, the maximum amount of information loss can only include whatever has been up-dated since the last pass of the Backup function. All previous information updates since the last complete secondary storage dump (on tape) can be retrieved from the Backup generated incremental tapes.

Thus, MULTICS provides a very comprehensive vehicle for insuring against gross information losses due to system failure.

Access Control

MULTICS incorporates an integrated Access Control System that provides data protection at the file level through a series of access control lists associated with files and their parent directories. This scheme allows the system designer to grant (or deny) any of read, write, append, and execute permissions to any user or group of users as identified by the standard MULTICS user identification name. This scheme is extensible to any desired lower level of data including the bit through additional software. This aspect of access control and data security is discussed at length in the Section entitled "Access Control for Data Management Systems".

MULTICS access control is founded on the ring protection concept. This philosophy involves segregating users into several concentric rings, the rings closest to the center having the most access until the central ring or "ring 0" represents the hardcore MULTICS supervisor. This concept allows different users to be segregated by access class. For example, both the teacher and students of a data base designed for teaching purposes would need all permissions on the teaching programs and data. However, the students could be placed in a ring which would not allow them to use the files and programs that administered the examinations and accounted for course grades.

Conclusions

It should be evident that MULTICS provides the necessary tools for system programmers in the design of modules that realize the essential functions of a data management system. In addition, if a moment of thought is given to the five functions enumerated, that is, process control, hierarchy management, storage management, file backup and access control, it will be seen that these are among the essential elements of any comtemporary software system. Systems from text editors to computer aided instruction to chess playing all depend on these foundational blocks in order to maintain their viability. MULTICS provides an unprecedented array of system building features that is at least three years ahead of any conventional operating system. Now, and in the years to come, such an operating system will be indispensable for the serious research and development of large software systems for the Air Force and for mounting a realistic attack on their associated, sky rocketing costs.

3.1.3 The Data Management Model

Overview

MULTICS Data Manager (MDM) is a multi-user, interactive, formatted, data storage and retrieval model designed to allow naive users and sophisticated programmers to define, organize and work with a large complex data base. As a model it is not intended to represent a finished software system ready to cope with a hostile user community. Its primary purpose for existence is to demonstrate the unique system building powers

available under MULTICS. The knowledge of MULTICS and Data Management Systems gained in building MDM must be appreciated as a serendipitous event that is certainly sharable with any serious student of these disciplines.

The term "model", as used above, differs from the commonly held notion of a "mathematical model", in that it resembles a prototype system rather than a device which when stimulated with inputs produces a mathematical approximation of some situation. The model described here differs from a genuine prototype in that the implementor has taken liberties in certain areas of the programming of the model. For example, if one functional module of a system model successfully demonstrates the design methodology for attacking a certain problem, and this same problem is evident in other functional modules in addition to other interesting problems, then if the problem already demonstrated involves a large programming effort, it can be circumvented in the other modules. There are usually several ways of circumventing such problems and a supporting operating system such as MULTICS provides the tools to quickly realize these. However, the price involved in this "short cut" is usually the restriction of the generality of the implementation. Hence, the end result is a working model rather than a finished system. However, the model is sufficient in that it proves (or disproves) a given design.

It must be held in mind that the model approach does not allow for short cutting on the system design. This architectural plan for a system must be logically complete. It is

the intention of this report to present such a plan in the following sections. Wherever the implementation significantly exercises the liberties of model building, and departs from the design requirements, an explanation of the model will be given.

MDM Design Philosophy

Primary in the design philosophy of MDM has been the notion of providing a community of both naive users and sophisticated programmers with the ability to interact with a common data base or with several data bases.

This design is general in nature so that MDM is not tied to a specific group of files. Users are provided with tools for defining a data base, entering information into the data base and then operating on that data. All references to data involved in these functions are by symbolic name and not physical location.

The organization of the files in MDM follows a fairly standard multi-level, hierarchical structure with any number of levels of subsumed files. The logical files are stored in individual, unique MULTICS segments. The segments are internally structured in a simple, linked-list arrangement that provides sequential access to individual data items within them.

The interactive user interfaces to MDM by means of a command language that allows him to move about the file hierarchy, settle on a particular file of a data base and then operate on the data within that file. He may then change his position in the hierarchy to another file at the same level, or different level or may opt to change to a different data base.

The programmer approaches MDM through the vehicle of his PL-1 application program. All of the program modules that represent the functions of the MDM command language exist as independent entities which the application program can call upon to manipulate data. Thus, the application program utilizes these functions in a manner identical in nature to that of the interactive user. However, the application program must incorporate a PL-1 data structure that can adequately describe the manner in which the data is stored. This enables the program to symbolically reference the individual data items.

MDM assumes the existence of a system administrator. It is his function to provide users with access to MDM facilities and establish certain informtion groups necessary to the existence and proper operation of MDM.

Internal Data Structure

Since it is the design philosophy of MDM to provide the user with symbolic access to data, the representation of the logical data base has been separated from the internal storage mechanisms. The connecting link between the physical and logical aspects of MDM is held in the System Directories for the data bases.

Logical Data Structure

The logical data structure of MDM is divided into four levels of data aggregation. They are:

1. Data Field - The logical atomic unit of data in the system; it has a name and value.

2. Data Record - A logically defined collection of data fields.

3. Logical File - A named logically defined collection of data records all of which have the same logical definition. Logical files may also include one or more subsumed files.

4. Data base - A named collection of all logical files that comprize a user's data set.

A data base is logically arranged in a multi-level, hierarchical structure. See Fig. 3. Each data base is presumed to exist as a free-standing logical entity that is not related to any other data base.

MDM File Hierarchy

As shown in Fig. 3, a data base consists of a group of files arranged in a multi-level hierarchy. There is no logical limit in MDM to the number of files that can be defined in a data base. The practical limit is dictated by certain MULTICS system conventions. The files that appear at the top of the hierarchy are referred to as "primary" or "level 1" files. These are the main organizational files of the data base. At least one field of the records which comprize a primary file will usually contain a unique identifier such as a name or number. The primary files may associate with any number of subsumed files which act as repeating groups of information. Thus, for each record of a primary file one or more occurrences of records will appear in

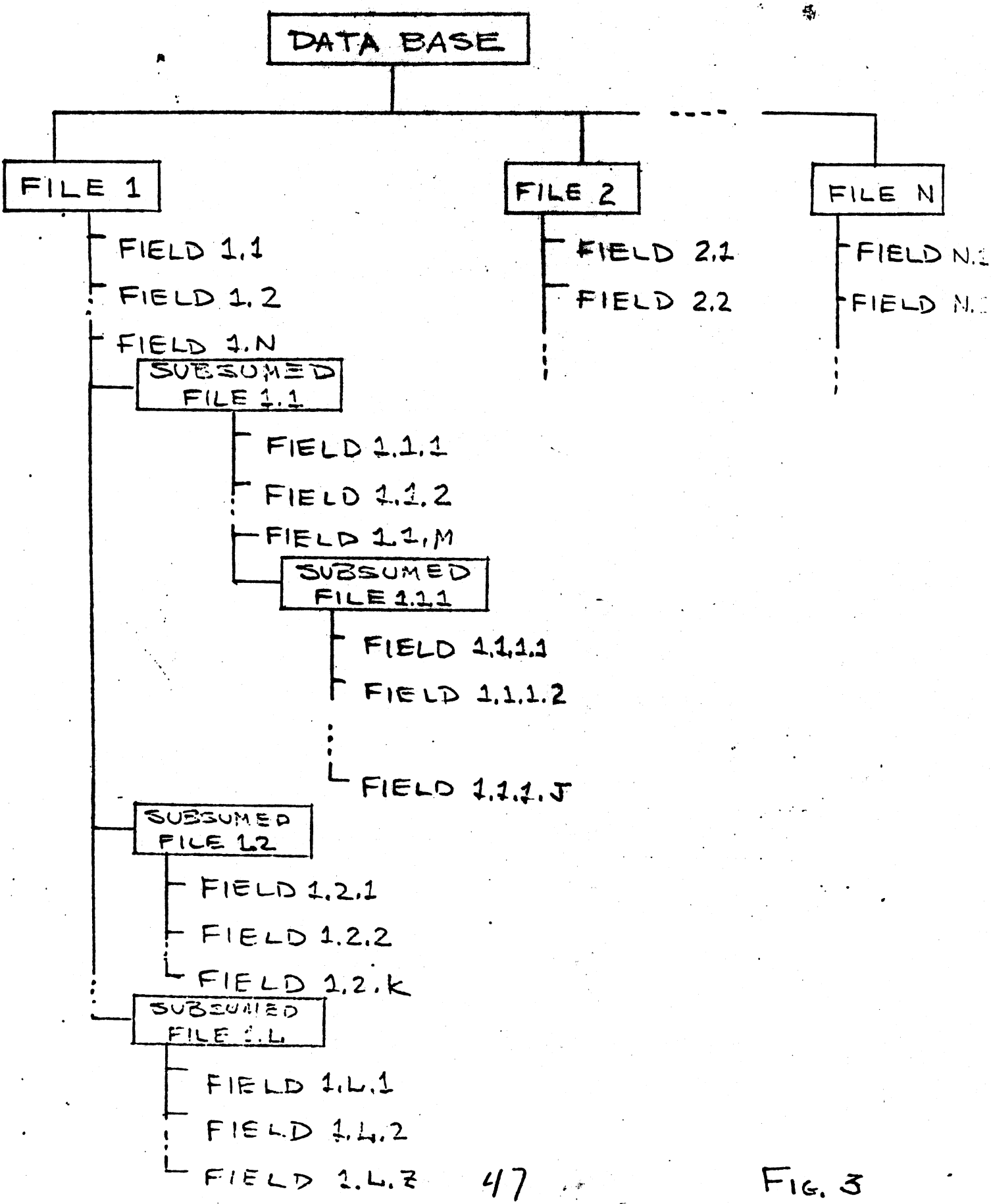
DATA BASE
FILE 1
FILE 2
FILE N
FIELD 1.1
FIELD 1.2
FIELD 1.N
FIELD 2.1
FIELD 2.2
FIELD N.1
FIELD N.2
SUBSUMED FILE 1.1
FIELD 1.1.1
FIELD 1.1.2
FIELD 1.1.M
SUBSUMED FILE 1.11
FIELD 1.1.1.1
FIELD 1.1.1.2
FIELD 1.1.1.J
SUBSUMED FILE 1.2
FIELD 1.2.1
FIELD 1.2.2
FIELD 1.2.K
SUBSUMED FILE 1.L
FIELD 1.L.1
FIELD 1.L.2
FIELD 1.L.3

FIG. 3

each subsumed file. The files formed in this manner which are immediately below the primary files are called "level 2" subsumed files. They, in turn, may contain any number of subsumed files in the same manner, and each level is associated with a number in increasing order, i.e., 3, 4, 5, ... Thus, a logical way of handling a large number of classes of data is achieved. The present version of MDM implements only primary level files.

MDM's logical files consist of records that are composed of fixed length data fields, variable length data fields and any number of subsumed files. The logical limit on file size in MDM is approximately 32 million computer words (1 GE - 645 computer word = 36 bits).

Probably the main characteristic of the records that belong to a particular file is that they all have the same structure. Thus, they represent a repeating group of information and are organized in this manner to take advantage of certain storage and access benefits that result.

Records are either fixed or variable in length depending on whether the fields that comprize them are fixed or variable in length. A fixed length record (Fig. 4) represents a repeating group of information any bit of which is always a fixed length away from the corresponding bit in the previous or successive record. This special relationship that exists for fixed length records can be utilized by system implementors to increase storing and accessing efficiency. For variable length records (Fig. 5) this relationship does not hold. Storing and accessing are