

ESD TR-66-113 III
Esti file copy

ESD-TR-66-113
Vol. 3

ESTI RECORD COPY
RETURN TO
SCIENTIFIC & TECHNICAL INFORMATION DIVISION
(ESTI), BUILDING 121B

ESD ACCESSION LIST

ESTI Call No. AL 58526

Copy No. 1 of 1 MTR-197^{CYS}

Vol. 3

A METHOD FOR THE EVALUATION OF SOFTWARE:
EXECUTIVE, OPERATING OR MONITOR SYSTEMS

SEPTEMBER 1967

A. E. Budd

ESQ
1967

Prepared for

EDP EQUIPMENT OFFICE

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE

L. G. Hanscom Field, Bedford, Massachusetts



This document has been approved for public release and sale; its distribution is unlimited.

Project 8510

Prepared by

THE MITRE CORPORATION
Bedford, Massachusetts

Contract AF19(628)-5165

ADDG12M

When US Government drawings, specifications, or other data are used for any purpose other than a definitely related government procurement operation, the government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data is not to be regarded by implication or otherwise, as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

Do not return this copy. Retain or destroy.

A METHOD FOR THE EVALUATION OF SOFTWARE:
EXECUTIVE, OPERATING OR MONITOR SYSTEMS

SEPTEMBER 1967

A. E. Budd

Prepared for

EDP EQUIPMENT OFFICE

ELECTRONIC SYSTEMS DIVISION
AIR FORCE SYSTEMS COMMAND
UNITED STATES AIR FORCE
L. G. Hanscom Field, Bedford, Massachusetts



This document has been approved for public release and sale; its distribution is unlimited.

Project 8510
Prepared by
THE MITRE CORPORATION
Bedford, Massachusetts
Contract AF19(628)-5165

AD 661999

FOREWORD

The work reported in this document was conducted by The MITRE Corporation for the EDP Equipment Office, Electronic Systems Division, Air Force Systems Command, L. G. Hanscom Field, Bedford, Massachusetts, under Contract AF 19(628)-5165.

REVIEW AND APPROVAL

This technical report has been reviewed and is approved.



S. P. STEFFES
Colonel, USAF
Chief, EDP Equipment Office

ABSTRACT

This report contains features of executive, operating or monitor systems considered important for evaluation and comparative analysis. These features are identified in a form expressly for inclusion in the Three Step Method for Software Evaluation (Volume 1 of this series) under Category TWO: Executive, Operating or Monitor Systems. Included in this volume is a composite list of functions contained in current executive systems. These functions provide the basis for a standard approach to the software category of executive systems particularly needed for evaluation and comparative analysis.

TABLE OF CONTENTS

		<u>Page</u>
SECTION I	INTRODUCTION	1
	FUNCTIONAL DIVISIONS	3
	BENCHMARK CONSIDERATIONS	5
SECTION II	EXECUTIVE SYSTEM FUNCTIONS	6
	JOB AND/OR TASK SCHEDULING	6
	I/O ALLOCATION, MONITORING AND CONTROL	10
	USER AND SYSTEM STORAGE ALLOCATION	12
	LIBRARY MANIPULATION AND MAINTENANCE	15
	EDITING OF USER AND SYSTEM PROGRAMS	16
	FACILITY AND USER TIME ACCOUNTING	18
	GENERATING AND UPDATING THE MASTER SYSTEM	19
	OPERATOR AND OFF-LINE COMMUNICATION	20
	ERROR RECOGNITION AND RECOVERY	21
	DOCUMENTATION	23
SECTION III	MATRIX OF FEATURES	24
	JOB AND/OR TASK SCHEDULING	25
	I/O ALLOCATION, MONITORING AND CONTROL	28
	USER AND SYSTEM STORAGE ALLOCATION	30
	LIBRARY MANIPULATION AND MAINTENANCE	33
	EDITING OF USER AND SYSTEM PROGRAMS	35
	FACILITY AND USER TIME ACCOUNTING	37
	GENERATING AND UPDATING THE MASTER SYSTEM	39
	OPERATOR AND OFF-LINE COMMUNICATION	40
	ERROR RECOGNITION AND RECOVERY	41
	DOCUMENTATION	43
SECTION IV	DESCRIPTION OF EXECUTIVE, OPERATING OR MONITOR SYSTEM FEATURES	45
	JOB AND/OR TASK SCHEDULING	45
	I/O ALLOCATION, MONITORING AND CONTROL	52
	USER AND SYSTEM STORAGE ALLOCATION	57
	LIBRARY MANIPULATION AND MAINTENANCE	62
	EDITING OF USER AND SYSTEM PROGRAMS	65
	FACILITY AND USER TIME ACCOUNTING	69
	GENERATING AND UPDATING THE MASTER SYSTEM	72
	OPERATOR AND OFF-LINE COMMUNICATION	74

TABLE OF CONTENTS (Continued)

	<u>Page</u>
ERROR RECOGNITION AND SYSTEM RECOVERY	76
DOCUMENTATION	79
SECTION V QUESTIONNAIRE	81
INTRODUCTION	81
RESPONDENT'S NOTE	81
FUNCTIONAL DIVISIONS	81

SECTION I

INTRODUCTION

Generally speaking an operating system is that portion of a software package which is at the highest level of control. Its parts include a monitoring portion which provides a simplified interface for the user and the input/output subsystems; and a scheduling function to provide user programs with first-in/first-out compilation and/or execution of his jobs or tasks. In some instances, more complicated priority oriented designations are used instead of the commonly used first-in/first-out scheme providing different levels of priority. Other portions of the operating system include editing capabilities for updating and maintaining master system tapes and control of library routines for general utility functions and other commonly used programs. Functions associated with these systems include manual operations, such as tape and card changing, in addition to those performed within the programmed portions.

In order to relate the user's requirements to the operating system, each computing system has defined a language unique to its own capabilities and limitations. Each language is sufficient for its own needs and has been designed to best utilize the features of one particular computing system. Where one system provides for instructions on control cards only, another allows console intervention as well as control card instructions. Where one system provides library routines at compile time, another provides them at program load and execute time. Where one has provision for compile, compile and execute, and execute, another system will accept compile and load in addition to these. Where one system uses the letter designation "XEQ" to mean load and execute a binary program, another uses "LOAD" to mean the same thing. Card formats differ both in relative positions used and in the fact that one has a fixed format while another has a variable one. In any event, a series of instructions acceptable to one computing system is unacceptable and often meaningless to others. At the present time, "machine independent languages" do not include operating systems by any stretch of the imagination.

Considerable emphasis is currently being placed on operating, executive or monitor systems by EDP equipment users, manufacturers and software development groups. These systems were reasonably simple when first introduced but have grown much more complex due to increased hardware complexities such as simultaneous input/output channels, addition of one or more central processing units having access to the main memory, and others. One important reason for this current emphasis is that executive systems are highly machine dependent. Each executive system was designed around a particular machine and only produces or implements those functions which the hardware does not perform. If some feature of the hardware does not allow one to implement the kind of scheduling algorithm that is needed, then the executive system

designer ends up putting this in his executive system by way of software. Another reason is that no standards exist. In the compiler area, there are known techniques for building compilers - it is relatively easy to find out what they are and to find out how to implement these particular techniques. This is not true in the executive system area. If one is trying to implement a scheduling algorithm, he finds out what hardware features are available and uses those in conjunction with whatever else he feels is necessary from his own knowledge in order to implement that particular scheduling algorithm. A standard approach to an executive system is what is needed here, rather than a standard executive system. Executive systems are highly dependent on the particular machine for which they were designed and therefore, it is meaningless to try to define a standard system. However, it is certainly possible to define a standard approach to executive systems for purposes of equipment evaluation at least.

Multiprocessing and multiprogramming hardware requires an executive framework in order to perform effectively. You must have some function at the highest level of control that can assign particular hardware components to perform requested functions. This requirement is due to the fact that you have several input-output channels operating simultaneously or several central processing units with the capability of operating simultaneously.

Many problems are encountered with timing any kind of equipment demonstration, either a benchmark program written in some compiler language, machine language, or a benchmark such as reading-in data from tape or from a real-time device. All the general timing problems involve investigation of the executive system functions. In some systems, for example, library routines needed within a program are read-in during the compilation phase. In others, these routines are read-in to primary storage at the same time the program is read-in for execution. In still other systems, library routines are read-in during program execution only as they are needed. This means that some systems will have compilation times which include library routine read-in while others will have the time required for reading library routines included in execution timing. The difficulty arises from the fact that the user is not aware of these differences unless the executive system receives sufficient investigation. Investigation of the executive system's features and functions should identify the differences between vendor's computing systems. It is important to find a point in time at which the executive system has stopped doing something, or the compilation process has been completed, execution has been completed, or tape being read-in has stopped, and it is in the executive system that these points in time occur. At any rate, a study of the difficulties in timing end up to be a study of executive system functions.

Many of the current executive functions were formerly performed by compilers. In the past, compilers had a series of cards which were

input and allowed things such as tape assignments, compile and go statements, and others. These functions formerly performed by compilers are now separated from the compiler and performed by the executive system. Most vendors have a large percentage of programmers working on executive systems for different computers or on different computer configurations. Most users end up having to have one or more system programmers to maintain and make small changes, or update the latest changes from the vendor, to the executive system. This is a very time-consuming type of thing. In some cases, Air Force installations have had to produce whole new executive systems which is a difficult and time-consuming thing to do.

FUNCTIONAL DIVISIONS

This report contains the important functions and features of executive, operating or monitor systems. These features are a composite from which the evaluator, in conjunction with the user, may select providing they are consistent with the system requirements. The evaluator would first select those features which are important to the user. This list would be part of the basic system requirements of the evaluation team in which one vendor's answer or result for one feature would form one entry. This would provide a convenient method of comparing the features of one vendor as well as comparing the vendor's response to each feature.

The evaluator must work with the user to determine his specific needs. For instance, the user may require immediate response by the system to certain real-time data inputs requiring priority interruption of the program being executed. On the other hand, another installation may require a strict first-come, first-served scheme since everyone has about the same priority but in addition require a complex scheme for communicating with other off-line and on-line computers. Whichever the case may be, these needs must be determined through the user so that the resulting system selection will meet his specified requirements.

The list of functions in SECTION TWO is a composite of those performed in various operating systems now being planned, being implemented or already operational. The emphasis has been centered on the function which is or will be performed by the system rather than the method or technique of implementing its execution.

Some of the items have been named by computer manufacturers' software groups or independent software groups as functions contained within some already available compiler, assembly program or other distinguishable piece of software. This confusion has arisen because one or more of the functions in question were originally programmed into that software unit. When the software unit was produced, no operating system existed as a separate entity and therefore the function was

included for convenience. However, as the concept of executive control has developed, these functions have switched their alliance to operating systems. At any rate, only those functions which pertain to executive or operating systems are included.

The features are presented in three sections. The first of these, SECTION THREE, contains summary titles of each feature in the matrix form required for the three step procedure described in Volume 1 of this series. The next section, SECTION FOUR, contains a description of what is meant by each summary titled feature, which measures of software capability are affected by this feature and, where necessary, further explanation indicating what aspects are especially useful to an installation or specific application. The last section, SECTION FIVE, contains many of the questions appearing in SECTION FOUR and additional questions to make up a composite questionnaire from which evaluators may select from for inclusion in proposal requests or verbal briefings. The questions are presented separately for convenience and will assist the evaluator in obtaining appropriate information for feature evaluation.

It should be remembered that these features and questions are a composite and therefore, only those features or questions deemed important relative to the specific application or installation under consideration should be selected.

Each section including the executive functions section is presented in the following ten functional divisions:

1. Job and/or Task Scheduling - relating to the capability of the system to schedule jobs, programs or tasks by suitable assignment of equipment components.
2. I/O Allocation, Monitoring and Control - concerns the allocation of input/output channels and devices, as well as monitoring their status and effecting their control.
3. User and System Storage Allocation - concerns the methods and techniques of allocating both primary and secondary storage; and includes the interface with jobs, programs and tasks.
4. Library Manipulation and Maintenance - deals with sub-routines, subprograms or portions of the programming system whose purpose in being is solely to provide maintenance of commonly used library routines in addition to the actual manipulation of them.
5. Editing of User and System Programs - primarily concerns reporting of program, software and machine errors as well as debugging facilities for both the user and the system programs.

6. Facility and User Time Accounting - the accounting of user and system program execution times, and of individual component use times.
7. Generating and Updating the Master System - updating and adding new software components to the master system.
8. Operator and Off-Line Communication - oriented to the content and makeup of messages and directions to off-line computers and human operators.
9. Error Recognition and Recovery - features and functions concerning errors in the central processor, input/output channels and devices, including related system recovery procedures.
10. Documentation - relates to manuals and listings produced for the user-programmer and the system programmer.

BENCHMARK CONSIDERATION

In order to remain unbiased with respect to computing system vendors, benchmark programs should, ideally, be written in machine independent languages. This is not possible for executive systems since their input languages are highly dependent on specific features of the hardware. However, if the major and minor functions of the category of executive systems were used as a machine independent language, then the evaluation process could use these functional definitions for timing and equipment demonstrations, as well as remain unbiased with respect to vendors.

One of the major problems with current executive systems concerns the amount of computer time used in execution of the individual functions. For instance, one system may contain all the features and functions required by the evaluator-user team but be so inefficient in implementation that it causes the overall computing system to fail the timing requirements. On the other end of the scale, a different system might meet the timing requirements but fail to provide the functions and features required for adequate operation of the user's installation. The evaluation process must provide a realistic investigation so that an adequate balance may be reached between the functions required by the user and the demonstration of these functions within reasonable timing limits.

It is with these considerations in mind that SECTION TWO, containing the major and minor functions of executive systems, is presented to assist evaluators by providing a standard approach to the category of executive systems.

SECTION II

EXECUTIVE SYSTEM FUNCTIONS

A comprehensive list of automatic operating system functions which cover a broad range of possibilities is described below. Whenever applicable, an explanation is given describing problems concerning implementation of the particular function since one or more special techniques may have been used. The machine dependence of certain selected functions is discussed; that is, is this function one which has been designed and implemented relative to one specific hardware feature found on only one machine? In contrast to this, there are subsets of functions which are more machine independent and are frequently needed whenever an operating system is implemented.

In any list of functions it is very difficult to exclude the interdependencies within any one operating system design. Many of the following functions are integrated by the designer into one design and done so in a manner which tends to emphasize similar characteristics of each function, combined in a way that will conserve primary storage space and execution time. However, in this report the goal is to isolate each function so that its description concerns the characteristics of that function only. In some cases, it may seem reasonable to assume a particular function should fall into a different general functional category than was selected. It should be realized that the one selected provided the primary concern and therefore could appear in another category only in a secondary way.

Nine major aggregates of functions have been identified and each of the executive or automatic operating system functions has been suitably described and integrated within one of these major aggregates.

JOB AND/OR TASK SCHEDULING

This particular major function is present in all operating systems, to some degree, in one form or another. The particular section of the operating system which handles priority, either on a first-come, first-served basis, or by some generalized scheme which includes first-come, first-served as a special case. Definitions of the term "job" and the term "task" vary from system to system. However, in this report we describe a job as one complete set of coding, no matter what language it is written in, which includes its input and output as one unit. Any one programmer might submit many jobs but any one of those jobs is a complete entity in itself containing all the required processes needed

to perform a meaningful and complete function. By a task is meant one of the processes. For instance, reading in input from one tape; or reading in one card from a card reader; or the calculation of a square root; or printing out one line would be considered a task. It takes many tasks to make up one job. Included in this section are those functions which relate to priority scheduling of either the jobs or tasks, or some appropriate combination of both.

Compilation, Assembly, Loading, Execution, Test Execution and/or Appropriate Combinations of User Programs

The operating system provides a framework in which the user may request any desired sequence of compiling, loading, execution, etc. Portions of the assembler or compiler are ready or executed at the appropriate time, the compiled or assembled program and data are placed suitably in the machine for test or complete execution. A smooth and efficient transition from one program or job to the next is one of the most important functions it can perform. This function is independent of any specific machine and is characterized by the fact that all relevant operating systems provide it and all EDP users require it to some degree.

Running of Several Programs Serially or in Parallel

Each operating system has its own definitions, flexibilities, input language, etc. A program may really be an entire programmed system, in one case, or a very particular and commonly used small routine with no input/output requirements in another case. To what degree programs, systems or routines may be operated simultaneously depends on the objectives of the designer as well as the capabilities of the particular machine. At any rate, a structure must be provided with limitations and flexibilities which allow several programs or a continuous stream of programs to run without interruption. This may be performed by running these programs in parallel or simply in some serial fashion. Relatively little programming development work has been done for parallel operations and, as a result, paper proposals must be analyzed very closely.

Run-to-Run Program Parameter Changes or Modifications

Once a program is checked out and considered to be ready for production, the user may require several such runs, each with one or more of its input parameters changed. In fact, it may be convenient to check out a program (after the initial or first phase of checkout) by setting up a series of runs changing one or more parameters at a time to establish or verify its checked out status. Each operating system requires different rules or techniques for executing these runs.

Inter- and/or Intra-Program Task Execution in Parallel

In systems having a multiprocessing capability a flexibility is sometimes provided the user-programmer to submit jobs or tasks in parallel; that is, tasks may be executed within a certain job in parallel which would be considered either multiprogramming or multiprocessing. The function of the operating system is to provide the environment such that tasks may be executed in parallel within a program providing, of course, the hardware capability is present. Important in this area is what limitations have been imposed and for what reasons. For example, is it possible within the user program to direct the operating system to begin reading in or reading out certain data portions in advance of their actual need. Or is it possible to request that a certain subroutine, such as a square root routine, might be executed using a different central processor at this time even though it would not be needed until later; thus providing the user with a facility that could make his overall program run faster. This particular function has many ramifications and in some cases has been so limited that multiprocessing was impossible even when the hardware capability was present. This function depends on certain unique hardware features not found in all machines.

Controlling Use of Common Subroutines (i.e. Reentrant Coding Routines) by Several Central Processing Units

In a multiprocessing environment more than one central processing unit may require operation of the same subroutine. The operating system must ensure each of these central processors a correct copy of the subroutine as well as providing each processor with its own temporary storage. The framework for controlling this function should be flexible enough to handle future central processing unit additions and associated equipment changes.

Monitoring and Control of Utility and/or User Program Timed Requests

Continuous monitoring is required, in some cases, to provide the user with the capability of instructing the computing system to perform a particular function at a certain pre-specified time. In other cases a timed interrupt can be programmed to provide this monitoring. The requests could be relative to certain real-time devices, or to a specific standard I/O requirement, or even to a scheduled maintenance type request. Within the programming system, requests for certain I/O data transfers could be controlled in much the same way. In most systems this monitoring and control must be done by a previously defined and agreed upon schedule of priorities.

Automatic Program, Job, or Task Scheduling Control

A programmed request may necessitate reading in a program from the user's own library of programs. This user may have a variety of different programs which upon his request are compiled or executed within his current program, job, or task. This particular function is much more dependent on the design objectives of the entire programming system than it is on the particular machine used.

Monitoring of User and Program Priorities

An initial request containing a low priority, in some systems, would get this priority raised providing certain prearranged units of time have passed without the program being executed. These priorities must be monitored then for all programs, both active and queued. This particular function may depend on another function of scheduling control if the scheme used for scheduling and priority is considered complicated. This monitoring function is dependent on both hardware features and operating system design objectives.

Automatic Rescheduling and System Reorganization for Priority Programs

In the event of high priority requests, the operating system may be required to perform component rescheduling and rearranging of the entire computing system status in order to provide an environment for the execution of the high priority program. It could happen that with suitable rescheduling and reorganizing of the system status the high priority program may operate with only minor interruption to the program which it interrupted. This function is dependent on the machine hardware features and on the design objectives of the operating system.

Generation and Checking of Program and Data Identification

In order that the correct priority scheduling of jobs and tasks may be done, the operating system must check the programmer's identification cards or records, and suitably generate any internal communication needs for other central processing units as well as establish logical switches so that appropriate monitoring of the entire computing system may continue. In addition, many installations require project numbers and insist on formatted or coded priority requests to ensure the accurate implementation of a particular priority standard. This is a primary scheduling function and is essentially machine independent.

Maintenance of Task, Program and/or Job Queue Tables

The scheduling of jobs, tasks and programs will require bookkeeping of certain identification data relative to each. This is dependent on the level of complication of the priority scheme used. It is the scheduling function that keeps these tables updated so that the proper sequence of programs may be executed.

I/O ALLOCATION, MONITORING AND CONTROL

Within an operating system there are portions of the program which relate to allocating input/output devices and channels as well as monitoring and controlling the devices and channels. Naturally some machines require more or less of this monitoring and control than others do. In some cases symbolic addressing of I/O devices and channels provides more flexibility and less machine dependence for the individual programs. Under this major function are listed all of those minor functions which relate to allocation, monitoring and control of I/O, but excluding the functions relating to storage allocation since they will be integrated under another major function.

Future Additions of Standard I/O Equipment

Flexibility is often needed in the design of those systems having a symbolic I/O addressing capability, so that any particular installation which needs additional tapes, discs, drums, printers, etc., in the future should be allowed this addition with only a small modification to the operating system. It is common to start with minimum I/O and wait until more units are actually needed before they are purchased or rented.

Initiation and Control of Program Delays on I/O Requirements

In some cases, machine language I/O instructions are used by the operating system but unknown, for the most part, to the user program. Delaying certain operations until a channel is available or until a rewind is completed on a magnetic tape are functions executed within the operating system. There is a necessary balance here between savings in time and checkout problems for the programmer, on the one hand, and, on the other hand, necessary flexibility in the use of the I/O equipment. This function, by necessity, is completely machine dependent.

Addition of Special I/O and/or Communication Equipment

Installations which at the beginning have special I/O equipment such as certain real-time devices, or plan in the future to add this type of equipment, do well to have a flexibility built into the operating system such that these additions may be made expeditiously. If the framework of the operating system has limitations not conducive to future, predicted requirements, then a very large portion of the operating system may have to be modified when that equipment is added. This equipment may include different types of sensors or various types of long-line communication equipment. The system specification may require sufficient flexibility to allow utilization of this type of facility addition.

Future Modification of Simultaneous Channel Capability

Sufficient flexibility may be required to provide for either future increases or decreases in the number of simultaneous channels used. Increasing the number of channels available in a system, resulting in more flexibility for allocating and controlling I/O equipment, may provide savings not realizable at the time of installation but rather may be appropriate for future considerations. This capability should be contained in the operating system at the time of hardware selection. Each operating system would implement this function to best suit the equipment used.

Conversion of User Symbolic I/O Assignments to Actual Channel and Unit Numbers

In a system which provides the capability of assigning I/O equipments with symbols, the operating system must effect the conversion to channel and unit numbers. This function is essentially independent of the machine since it is required by any system allowing symbolic assignments of I/O equipments; however, in some cases, the conversion is done by using a unique hardware feature.

Monitoring and Updating of Entire Computing System Status

The operating system must know, at any time, the status of all components in order to effectively and efficiently schedule programs, jobs, or tasks. Some operating systems contain tables providing an environment for this updating and monitoring. The primary concern is the problem involved with allocation and control of I/O equipment; although scheduling jobs is another major category of some importance to this function.

Input/Output Equipment Scheduling, and Interrupt and Channel Monitoring

One function of the operating system is to honor requests from user programs in the use of scratch tapes, input data tapes, areas of secondary and primary storage for temporary use. This schedule is initiated so that smooth program-to-program or job-to-job control transfers may be efficient and smooth. The algorithm for effecting this equipment scheduling must provide for smooth and efficient operation. It is within this portion of the program handling the scheduling that monitoring of appropriate and associated interrupts is done since interrupts that convey the completion of an I/O transfer may affect this schedule. This function is highly machine dependent.

Queuing Messages, Programs and Data From and to Remote Consoles

Remote consoles require immediate attention. Incoming data, messages and programs require the operating system to react in some way so that appropriate storage of this data, message or program may be saved for some time later on when execution may take place. This is mainly a problem of I/O allocation, monitoring and control with scheduling and storage allocation important but secondary functions. This function depends on the particular hardware installation and can be aided by hardware features to effect efficiency.

USER AND SYSTEM STORAGE ALLOCATION

Certain subroutines and portions of the executive system are dedicated to allocation of storage for the user program; that is, the user may need blocks of core, blocks of disc, drum, etc. Those minor functions which assist in storage allocation for both the user and for the operating system storage allocation requirements or compiler storage allocation requirements are listed within this major function.

Manipulation and Activation of Program Overlays

Any user programmer must segment his problem (if segmentation is indeed required) according to a prearranged framework. Boundaries and limits to any set scheme are defined by the method selected for implementation. The operating system provides the impetus for effecting the movement of these segments in addition to establishing specific characteristics for its use. In most cases, this function would depend more on the design concept than the particular equipments used.

Selection of Combinations of Debugging Operations

Provision for the selection of a series of debugging operations as an aid to check-out is often present in the basic structure. A user programmer would be able to request stopping operation at some point so that a memory dump could be taken, a part of the program traced or any combination of these in a suitable series. This is commonly called part of the executive function since a long string of these requests may be handled, by the operating system, as a single request from the user. The important thing in this function is how it is implemented, that is, how flexible is it for the programmer's use. The implementation of this function is also concerned with how the storage is allocated both in working and secondary storage.

Running Foreground and/or Background Jobs or Tasks for Multiple-Access Systems

In some installations it is a requirement to be able to run jobs presented for the most part on-line. However, jobs which take several hours to run might require a capability of doing the longer jobs whenever there is no on-line job to be run; and then when an on-line request comes in, interrupting the longer job as necessary. This particular function is dependent on the machine; however, the requirements on the user-programmer could easily be independent of any one machine. Storage allocation is the primary concern for this function, although the scheduling of jobs or tasks including I/O equipments is also important.

Allocation of Primary and Secondary Storage

One particular method or technique must be selected to allocate all or portions of the storage devices associated with a computing system. Many operating systems have selected a technique to manipulate relocatable code produced by the compiler or assembler. Absolute memory addresses are assigned at load time in other systems, providing storage allocation in a rather limited way. Does the system simply break various storage devices into small segments and allocate one or more upon request? How may the user or programmer reconstruct his particular program operation in the environment of relocatable code if that is used? Dynamic allocation of storage is often much more complicated than a static allocation scheme and is much more difficult to isolate as a function.

Job, Program and Task Loading and Component Initialization

The actual program used to read in the job, program or task is often included in the priority scheduling. Component initialization, such as reading identification labels on magnetic tapes, is performed

within the operating system in some cases. This particular function depends more on the design of the operating system features than on the particular machine used. The primary problem of implementing this function concerns storage allocation for loading programs and data.

Memory Buffer Area Determination for I/O Assignments

When certain user requests occur, the operating system must determine the memory buffer area needed. The size of this memory buffer area will depend on how many other programs are presently using I/O equipments. The operating System could read in many records at one time into this memory buffer or read only one record at a time. Whichever is done depends on user request, computing system status, memory area available, the machine configuration such as number of simultaneous channels available, etc., and is an important storage allocation function.

Reading and Storing Job Segments of Program and Data on Secondary Storage Media

Portions of the operating system are produced strictly for reading and storing segments of the user program job and the related data on whatever disc, drum or other secondary storage media is available. These two processes may be done in conjunction with reading and storing other library and general utility routines. This function is machine dependent and additionally depends on how job segmenting is implemented as a technique. The primary concern is within the storage allocation category.

Relocation of Program Segments and/or Data Arrays and Tables Required for Priority Program Requests

In the event of a high priority request the operating system must assess the computing system status indicators to determine if data arrays or program segments must be relocated. The operating system must then indeed relocate any segments or data arrays in order to provide the high priority program with the needed components. The function of priority scheduling must indeed request or communicate with the storage allocation function in order to both determine what storage is available and to obtain the required storage. This function then depends on three major areas. It depends mainly on the function of storage allocation but also, to a lesser degree, on the priority scheduling of jobs and tasks, and on the library manipulation and maintenance functions.

Monitoring Requests and Returns of Primary and Secondary Storage Blocks

In an environment where the operating system controls storage allocation, user requests for additional storage or returned allocations must be honored and suitably recorded. Tables may be updated whenever a request is made which provide a current status of the primary and secondary storage allocations made. This function depends on the allocation scheme or technique used more than on a specific hardware feature. However, some modern equipments provide unique features which may make a particular scheme highly machine dependent.

LIBRARY MANIPULATION AND MAINTENANCE

Any subroutines, subprograms or portions of the programming system which provide maintenance of library routines for purposes of updating those now in existence, adding new ones, or deleting old ones are included here. Also the manipulation of the library whenever compilation or execution requests by user programs are recognized is included under this major function.

Maintenance (Scheduled and Nonscheduled) Program Operations

Provision is often available for the operation of certain maintenance type programs to ensure a level of reliability. The inclusion of these either scheduled or nonscheduled (the nonscheduled being larger and much more complete) allows maintenance personnel to submit jobs and/or tasks as part of the normal facility operations.

Simplified Access to Compilers, Assemblers, Library Routines and Other Languages

An uncomplicated, yet flexible, method for user programmers to utilize available compilers, assemblers, library routines, etc., is hard to attain. However, the framework is needed, for some applications, to appropriately mix statements in compiler language with machine language and certain other routines to obtain efficient and flexible programmed systems. Many systems provide only one program language to be written in at any one time. This function is independent of unique hardware features.

Instruction Linkage Between Compilers, Macro-Assemblers, Assemblers, Report Generators, Sort Routines, Etc.

The operating system must set up the appropriate instruction linkage for the user program so that any of the utility programs or

program systems may be integrated and manipulated under its jurisdiction. Many operating systems do not provide the flexibility of this instruction linkage and the linking must be done by the user programmer. In some cases, this linkage will initiate reading of the appropriate utility routine into core storage. In others, it will effect a transfer to that utility routine only. At any rate, this function will depend to a large extent on implementation techniques adapted for library maintenance and/or its manipulation.

Reading and Transfer of Control to System Utility Routines

Should a memory dump, or a sorting operation, or a report generation, or other request be made by the user, the operating system must perform read in of this particular routine plus provide the control transfers both to and from the operating system. How this particular function is implemented depends on how the instruction linkage between compilers, assemblers, etc., is implemented. In many cases, dependence is on the objectives of the software design and not on unique hardware features.

Substitution and Execution of Linkage for Program Segmentation

The act of substituting appropriate instruction linkages, including the execution of this linkage, is performed by the operating system so that problems of program segmentation do not require user coding. The user-programmer must be cognizant of the program segment but the operating system can ensure the presence of required input or output data tables or arrays for the appropriate segment. This particular function is more dependent on the type of language used than on the particular machine; however, there are hardware features which make this particular function easy to implement.

EDITING OF USER AND SYSTEM PROGRAMS

The executive system very often provides notes for the user so that he may reconstruct what happened during the operation of his program or job. Also these notes are provided to programming system communication functions; that is, communication needed between, say, the compiler, the operating system or any utility programs. This depends on how the particular system was designed and implemented.

Substitution and Deletion of Coding for Breakpoint, Snapshot or Memory Dumps and/or Tracing Routines

Small groups of instructions or code are placed at appropriate points in the user's program at his request, so that memory

dumps may be obtained at the required time, or tracing routines may operate on a prespecified portion of the program when certain pre-arranged conditions are satisfied. The instruction group may be as small as a single transfer instruction for snapshots, or sufficiently large to fill unused core storage with special instructions for system recovery during debugging runs. Substitution and deletion of this code by the operating system provides the framework necessary for efficient program checkout. Usually, however, manipulation of the input/output channels and devices are not provided for in the debugging part of the operating system. Some tracing routines, for instance, have provided no operations at all relative to I/O manipulations.

Integration of Programming Patches

After a programmer has made one or more runs on the computer in an attempt to check out his program, he often wants to add patches or small groups of instructions which will correct certain errors or omissions. Sometimes this is done at the source language level only and other times systems have the flexibility of doing this at the machine language level. If integrating these programming patches is not possible at the machine language level, then entire systems may have to be recompiled whereas if a small machine language patch could be made it could be run and tested without that recompilation. In some cases recompilation involves large amounts of machine time for both on and off-line computers.

Generation of Output Listings for Programmer Post-System Analysis

The executive system must make up certain printed listings useful to the programmer for analyzing a program or program system after execution. In many cases, even though each line has been generated at a different time during the execution, the operating system provides the programmer with a contiguous listing relative to the sequence of events occurring during the operation. In some cases each line is an English description of an event which occurred, but in other cases it is simply a number which refers the programmer to a list of possible output events in the operating system documentation. Oftentimes a compiler or some other unit of software produces this type of message requiring the operating system to suitably integrate it into the output listing. This function is essentially independent of any particular machine hardware features.

Formatting and Recording of Machine Errors

Editing for the user and programming maintenance personnel must provide a record of the particular machine errors which have

occurred during operation. In some cases these particular machine error descriptions are recorded with the job output. In others, the output includes simply a number used as reference into a table in the operating system documentation. This particular function is independent of the machine.

Monitoring and Control of Out-of-Range Data Quantities and Arithmetic Operation Violations

The executive system provides the flexibility of monitoring and controlling data calculations at prespecified times so that any quantities not within the predicted range may be tagged and a suitable output message generated and recorded. Also included are arithmetic operation violations not detected by the hardware. This type of function is, although performed by the operating system, defined and initiated by the user-programmer. This function will depend, for its implementation, on what hardware features are provided for monitoring data quantities and arithmetic operations and, in addition, on what hardware features are provided so that software monitoring and control may be effected.

FACILITY AND USER TIME ACCOUNTING

Automatic operating system functions which have been implemented to keep track of time for the overall facility, for particular user jobs (or tasks), are included here.

Execution of Job or Program Abortive Procedures

In the event the executive system detects user program errors which appear to cross certain prespecified boundaries, the operating system must execute a procedure which will provide suitable information to the programmer so that he may detect his error or errors and then abort that job. Also, the status of the computing system components must be updated so that the next program or programs will have the appropriate machine components available.

Updating Job, Program and Facility Time Accounting

In a system which provides preset timed interrupts, a job may be started after a clock is set to an appropriate upper bound for running programs not yet checked out. At the end of the appropriate time, prearranged within the facility, the operating system may execute the job or program abortive procedure. If a clock is provided then it is possible, within certain definable limits, to state the exact amount

of time used by a particular job and include times for each of the components used. It is also possible to provide the facility maintenance personnel and operating system maintenance personnel with data on times used for secondary storage transfers, number of storage accesses, etc., as well as general statistics relative to the number of jobs executed, frequency of utility routine usage, I/O channel and equipment usage, etc. This accounting of component times can be done to any degree desirable providing appropriate hardware is included and the operating system contains the necessary algorithms. Multiprocessing in a multi-access environment proposes some interesting and debatable problems concerning time accounting.

Generation and Maintenance of an Operation's Log

A permanent record of operations in many installations is a requirement. The generation and continuing maintenance of some type of logging procedure is needed for the facility for management reasons, maintenance functions, debugging operations, etc. This operation's log provides a record of past experience needed to predict future loads and requirements, as well as effect facility time accounting.

Maintenance of Separate Time Records for Individual System Components

Continuous updating of appropriate tables of one type or another is required to keep a separate time record of each system component. This must be done in conjunction with the I/O allocation, monitoring and control function, but is mainly a concern of the time accounting function. This function is almost entirely machine dependent.

GENERATING AND UPDATING THE MASTER SYSTEM

In any operating system certain portions are dedicated to assisting the overall installation maintenance programmers in updating the master system, such as adding new routines and deleting or correcting old routines; as well as generating new master system tapes or new master operating system copies. Any minor function which contributes or assists in doing this is included within this major aggregate.

Maintenance and Updating of Compilers, Assemblers, Utility and Other Routines

A basic structure is inherent in an executive system which allows for modifications to be effected to portions of the programming system. New versions or corrections of compilers, assemblers, or any general utility program in this system may be added or deleted by

following the rules associated with this basic structure. Special cases may exist in which this function depends on unique hardware features but in most cases it is machine independent.

Maintenance and Retrieval of Files and/or Systems of Files Directory Tables

The function of keeping a catalog of files for a whole category of user-programmers is one which can consume time in programming and in error detection. The directory table may contain information concerning each file in the system in addition to containing specific formats and data types within the files. This would provide a central data base for various applications within an installation. How this particular implementation is effected relative to available compilers and macro-assemblers within the system is of considerable importance. These files and systems of files are often connected with the master system tape.

OPERATOR AND OFF-LINE COMMUNICATION

A significant portion of the executive system is often directed to on-line messages for the operator or directions to an off-line operating system which is controlled by an operator. Also, there are portions of both user and system programs which are designed to react to operator requests concerning input jobs coming from the off-line equipment. All functions appropriately connected with these communications are described under this major function.

Organized Interface With Off-Line Computing Equipment

Many installations load input jobs or tasks off-line as well as print and/or punch off-line to conserve time on the main computer. The off-line device is very often a small computing system capable of a variety of data processing functions. Some logical interface and agreement in terms must be established to effect the transmission between the computers. An organized and standardized data configuration and command interpretation must be obtained if accurate and efficient job processing is to be realized. There would naturally be a high dependence on features from both the on-line and off-line computing hardware.

Communication With Other Computers or Computer Modules

One function performed by any multiprocessing system is to provide some standard communication between central processors. Whether a master-slave arrangement is implemented with one central processor permanently being master, or whether any central processor could be

master (depending on machine conditions) is immaterial. The concern here is that the communication between the central processors is performed, and the flexibility and open-endedness of that communication capability to provide the requirements for future modifications or additions be available.

Control and Communication With Subsystem Monitors

In some computing systems more than one level of monitor is required. Off-line equipment often has its own monitor requiring certain standardized transmissions and instructions. The executive system in the main computer must effect control by suitable communication in order to produce the requested and appropriate job output. It is possible for the main operating system to decide itself whether a particular request should be done within the main facility or communicated as a request to the off-line computer. This function depends on machine hardware features and on the objectives of the executive system design.

Generation and Console Communication of Output Manual Operator Commands, Input Requests and Associated Messages

One major objective of an operating system is to automate those functions previously required of a computer operator. System requests for activities such as magnetic tape changing, or card deck insertions, etc., are most often done manually. Also, the operator must be able to communicate with the automatic operating system in order to verify and/or specify what has been done. Equipment failures can be reported this way; that is, ones which do not stop machine operation. Listings of which programs are contained on certain output tapes and are to be printed off-line may be communicated to the operator. A flexible and efficient system for operator and off-line communication must be established in any system and is an even more important function in a large and expensive installation.

ERROR RECOGNITION AND RECOVERY

Those functions which detect errors in the central processor, I/O channels or equipment, and the relevant recovery procedures, are discussed under this major function.

Maintenance of Continued Operation During Limited I/O Component Failure

Each operating system must, to some degree, monitor input/output components and channels to continually ascertain whether failures have occurred. This function could be sufficiently flexible so

that a failure of a device not used or needed by the particular program now running would not affect that program. Further, it may be that several of this device type are available, and switching to the use of a different one could allow the program to continue. For example, if several units are available as a scratch tape the operating system could (under certain circumstances) switch to another if one failed. This function is highly machine dependent.

Detection and Recording of Programming Errors

The operating system must continually detect errors made by user programs such as incorrect input/output request specifications as well as hardware system errors. Also, these errors must be recorded appropriately for programmer post analysis, reconstruction and debugging activities. In some cases, the entire system may require restarting depending on the particular error. At any rate, this function depends on both hardware features and software design objectives.

Execution of Restart Procedures After Equipment Failures

In applications requiring program execution ranging from two hours to several days, capability has usually been required of the operating system to record the contents of the machine periodically so that the program may be restarted if equipment failures require complete abortion. The operating system must at each one of these periodic times effect the memory dump and at the appropriate time use the last memory dump to restart the entire program rather than starting from the beginning. This function is essentially machine dependent.

Monitoring Program Violations on Hardware Limitations

Many hardware limitations on specific computing equipments are hardware protected such that violations affect only the user program. The operating system can and does, in many cases, detect this type of error by suitably monitoring the program operation. Some computing systems provide a memory protection feature implemented as a software technique. This type of technique may be needed in lieu of an appropriate hardware feature. In any case, this function depends on unique and particular hardware features.

Parity Checking and Tape Redundancy Calculations

Memory parity checking continuously, as well as calculations for reliability of magnetic tape data transfers are both performed within the operating system. In one multiprocessing system, a second central processor continually checks memory parity at the discretion and under the direction of the operating system. Whether or not a tape

redundancy calculation is performed may depend on how much of the actual machine language I/O is performed by the operating system. This function is indeed machine independent since parity checking and tape checking are very common hardware features. This function may be implemented in conjunction with the execution of restart procedure.

Generation and Updating of Error Frequency Counts for Customer Engineer's Analysis

Parity error recognition during transmission or memory parity errors detected during confidence checking are needed by customer engineers in order to determine and maintain reliability. Any error occurring in the machine such that the operating system may effect a second try should be recorded for any appropriate component. This particular function is designed and implemented with other error recognition and system recovery functions in mind and is highly machine dependent.

Determination and Recovery from Unstable or Nonconverging Iterative Processes

It is a function of the operating system, in many cases with the assistance of the hardware features, to detect tight loops within programs which after a reasonable period of time appear to be nonconvergent. Some operating systems provide a feature whereby detection of this type of error when noticed will effect replacement with a suitable alternative procedure. This is best explained as an executive function since normally the programmer, after having performed post-analysis, would then decide what alternative procedure to use. This function depends on unique hardware features.

DOCUMENTATION

Reference and introductory manuals as well as output listings or print-outs are highly dependent on the particular computing system package proposed. These documentary type items must be evaluated as a complete package in which the specific requirements depend on both the user and on the make-up and content of individual vendor's proposals. The evaluator must determine how adequate each document and print-out is as well as determine what additional documents or additional print-outs are required to provide a completely documented system.

SECTION III

MATRIX OF FEATURES

Executive system features designated as important for comparative analysis are identified in this section. Each feature is presented with the appropriate format for inclusion in the three step procedure described in MTR-197, Volume I. In total, they represent a composite of important features each of which is described in more detail in SECTION FOUR.

The character "X" has been used to show which measures of software capability are affected by the features. It is apparent that any one feature, in some esoteric way, may affect each and every measure of software capability; however, an "X" has been placed under those measures which significantly affect the feature in question. The user-evaluator team must determine what measures are of particular importance to them, in addition to deciding whether each measure will be affected favorably or unfavorably (and to what degree) according to the system requirements. This will establish ground rules such that proposals may be evaluated relative to preestablished specifications appropriately adjusted to reflect characteristics of the application.

REQUIREMENT	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPILATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
		I. JOB AND/OR TASK SCHEDULING						X					X			
		A. framework for Compiling, Assembling, Loading, Execution and Appropriate Combinations														
		B. Provision for Test Execution		X												
		C. Transition from Job to Job				X							X			
		D. Facility for Running Several Jobs Serially											X			
		E. Facility for Running Jobs in Parallel				X					X					
		F. Utilization of Reentrant Coding in Programming System				X				X	X					
		G. Dynamic Scheduling of Program Segments								X						
		H. Inclusion of Externally Programmed Jobs														
		I. Run-to-Run Program Parameter Modifications														
		J. Parameter Modifications for Dynamic Debugging		X												
		K. Inter-Program Task Execution in Parallel				X										

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	I. JOB AND/OR TASK SCHEDULING (Continued)														
	L. Intra-Program Job Execution in Parallel				X				X	X					
	M. Limitations of Parallel Buffering Operations					X				X					
	N. Controlled Use of Common Subroutines				X					X					
	O. Provision for Future CPU Additions								X						
	P. Temporary Storage Availability for Each CPU	X	X					X							
	Q. Control of User Program Timed Requests					X									
	R. I/O Data Transfer Monitoring	X				X									
	S. Dynamic Task Compilation Requests	X							X						
	T. Automatic Library Program Manipulation	X													
	U. Scheduling of User's Own Library Programs	X	X												
	V. Monitoring of User and System Priorities				X										X

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES</u> <u>OF SOFTWARE</u> <u>CAPABILITY</u> <u>OPERATING, EXECUTIVE</u> <u>OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	I. JOB AND/OR TASK SCHEDULING (Continued)		X												
	W. Facility for Low Priority Programs						X								
	X. Automatic Rescheduling for Priority Programs								X						
	Y. System Reorganization Flexibility for High Priority Programs														
	Z. Checking of Program and Data Identification		X				X								
	AA. Complexity of Task Program and/or Job Queue Tables		X								X				
	AB. Dynamic Maintenance of Queue Tables		X		X										
	AC. Priority Scheme Flexibility for Job, Task or Program	X					X								

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	II. I/O ALLOCATION, MONITORING AND CONTROL		X			X									
	A. Initiation of Program Delays on I/O Requests					X									
	B. Continuous Interrupt Monitoring for User's I/O	X				X									
	C. Reduced Complexity of User Programs	X						X							
	D. Future Additions of Standard I/O Equipment							X							
	E. Flexibility for Changes in Channel Assignments					X			X						
	F. Addition of Special I/O Equipment					X			X						
	G. Future Inclusion of External Communication Devices					X			X						
	H. Buffer Area Reassignments for Modifications on Device Speeds or Capacities					X			X						
	I. Increases in Simultaneous Channel Capability					X			X	X					
	J. Decreases in Simultaneous Channel Capability				X										
	K. Efficient Conversion of Symbolic I/O Assignments	X				X									

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPILATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	II. I/O ALLOCATION, MONITORING AND CONTROL (Continued)														
	L. Flexible Symbolic I/O Assignment Scheme	X							X						
	M. Monitoring of Entire Computing System Status	X	X												
	N. Status Updating Scheme Flexibility	X			X										
	O. Technique for Allocating Input & Output Data Areas	X			X			X							
	P. Suitability of Algorithm for Equipment Scheduling	X	X												
	Q. Ease of Controlling Program-to-Program I/O				X							X			
	R. Channel Control and Allocation Flexibility					X				X					
	S. Allocation of Input/Output Devices					X		X							
	T. Remote Console Monitoring and Control				X	X									
	U. Queuing of Remote Console Messages	X			X										
	V. Addition of Remote Console Equipment													X	

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPILATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
		III. USER AND SYSTEM STORAGE ALLOCATION														
		A. Technique for Program Segmentation	X					X	X							
		B. Manipulation and Activation of Overlays	X						X	X						
		C. Scheme Flexibility to Include All Storage Devices	X						X							
		D. Facility for Protecting User's Data or Program	X	X												
		E. Effectiveness of Relocatability Technique	X	X												
		F. Provision for Absolute Address Assignments on Debugging Output		X		X										
		G. Limitations on Reconstructing Program Paths	X	X												
		H. Ease of Monitoring Storage Requests and Returns	X					X	X							
		I. User Flexibility of Requests and Returns	X					X	X							
		J. Availability of Storage Map and Assignment Tables	X	X				X								
		K. Flexibility for Priority Requests						X					X			

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	GENERAL MEASURES OF SOFTWARE CAPABILITY	OPERATING, EXECUTIVE OR MONITOR SYSTEMS	III. USER AND SYSTEM STORAGE ALLOCATION (Continued)	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
			L. Effects of Priority Requests on Segmentation				X		X								
			M. Manipulation of Job Segments to and from Secondary Storage	X			X			X							
			N. Similarity of Job Segment Technique to Library Routine Manipulation	X							X						
			O. Buffer Area Determination for I/O Assignment					X	X								
			P. Algorithm Suitability for Providing Each of Several Jobs with Fast Buffers				X	X									
			Q. Flexibility of User Requests for Buffer Areas								X						
			R. Effects of Changing Buffer Area Requirements on Allocation Scheme	X				X	X	X							
			S. Flexibility of Allocation Scheme to Machine Configuration Modifications	X						X							
			T. Buffer Area Assignment Problems with Increased or Decreased I/O Configuration	X													
			U. Job, Program and Task Initialization	X										X			
			V. Component Initialization Facility (Tape Rewind, Label Checking, etc.)	X										X			

- THREE STEP METHOD FOR SOFTWARE EVALUATION -

CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES</u> <u>OF SOFTWARE</u> <u>CAPABILITY</u> <u>OPERATING, EXECUTIVE</u> <u>OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPILATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	IV LIBRARY MANIPULATION AND MAINTENANCE														
	A. Flexibility of Framework for Library Access										X	X			
	B. Flexibility of Calling on Library Routines during Compilation, Execution, etc.	X									X	X			
	C. Instruction Linkage Between Routines			X	X										
	D. Requirements on User Program for Instruction Linkage	X					X								
	E. Reading Routines Automatically Provided	X					X								
	F. Repeated Requests for Same Library Routine						X								
	G. Library Routine Data Array or Table Needs Provided	X									X				
	H. Monitoring Utility Routine Control	X													
	I. Standard Scheme for Transfer to and from System Routines	X													
	J. Linkage Substitution for Program Segmentation	X			X										
	K. User Requirements for Library Routine Segment Manipulation	X				X									

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>														
	IV. LIBRARY MANIPULATION AND MAINTENANCE (Continued)														
	L. Provision for Updating Library Easily										X				
	M. Nonscheduled Library Maintenance				X						X				
	N. Suitability for Operational Reliability		X								X				
	O. Maintenance Jobs Treated as User Programs										X				
	P. Completeness of Routines for Library Manipulation and Maintenance								X						

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	V. EDITING OF USER & SYSTEM PROGRAMS															
	A. Listings for Program Post Analysis		X	X											X	
	B. Event Oriented Output for Ease of Reconstruction		X	X												
	C. Contiguous Output Message Listing		X	X											X	
	D. Suitability of Message Scheme for Event Recording		X	X											X	
	E. Messages Compatible with Other Software Components		X	X		X										
	F. Provision for On-Line Error Correction		X	X									X			
	G. Flexibility of Machine Language Patching		X	X												
	H. Sufficient Information for Adequate Path Reconstruction		X	X												
	I. Flexible Code Substitution for Debugging Runs		X	X		X										
	J. Flexibility of Formatting Debugging Options		X	X		X										
	K. Contiguous Listing of Recorded Errors											X				

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	VI. FACILITY AND USER TIME ACCOUNTING	X	X		X										
	A. Monitoring Job and Program Timing	X	X												
	B. Maintenance of Components Used Table	X				X									
	C. Provision for Generation of Statistics (Framework)	X													
	1. Number of Secondary Storage Transfers				X						X				
	2. Number of Storage Accesses				X						X				
	3. Number of Jobs Executed										X				
	4. Frequency of Utility Routine Usage				X						X				
	5. I/O Channel Time Used					X			X						
	6. I/O Device Time Used					X			X						
	7. Others														
	D. Execution of Job Abortive Procedures		X												

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u> VI. FACILITY AND USER TIME ACCOUNTING (Continued)	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	E. Computing System Components Status Updating					X									
	F. Recognition of Component Sharing by Jobs				X	X				X					
	G. Maintenance of Operations Log		X								X			X	
	H. Suitability of Log to Management Needs				X				X						
	I. Flexibility of Log for Predicting Future Configuration Changes							X							
	J. Separate Time Records for System Components					X								X	
	K. Sufficient Details for Component Timing								X						

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>														
<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>														
VII. GENERATING AND UPDATING THE MASTER SYSTEM														
A. Ease of Modifying Individual Programs	X									X				
B. Flexibility of Organization Scheme for Making New Masters					X					X	X			
C. Suitability for Updating Compilers, Assemblers, etc.					X					X				
D. Maintenance of File Systems	X	X								X				
E. Updating File Directory Tables	X	X								X				
F. Common Access of Often Used Files	X	X												
G. Compatibility of File System with Compilers, Assemblers, etc.	X	X								X				

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u> <u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u> VIII. OPERATOR AND OFF-LINE COMMUNICATION	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	A. Organized Interface with Off-Line Computers				X							X			
	B. Utilization of Similar Terminology		X									X			
	C. Standardized Data Configuration		X									X			
	D. Use of Standard Commands		X					X				X			
	E. Communication with Other Computer Modules							X							
	F. Open-Endedness of Communication Scheme							X							
	G. Allowance for Reflecting Equipment Configuration Changes							X							
	H. Control & Communication with Subsystem Monitors					X			X			X			
	I. Generation of Manual Operator Commands						X					X			
	J. Console Communication of Requests and Messages					X						X			
	K. Updated Record of I/O Device Assignments					X						X			

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	GENERAL MEASURES OF SOFTWARE CAPABILITY	PROGRAMMING	CHECKOUT	COMPILATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>														
	IX. ERROR RECOGNITION AND RECOVERY				X							X			
	A. Continued Operation During Limited Failures				X							X			
	B. Switching Like Components to Maintain System Operation				X	X						X			
	C. Recognition of Errors Relative to Each Available Component				X	X						X			
	D. Detection of Programming Errors		X											X	
	1. Incorrect I/O Requests		X			X									
	2. Requests for Equipment Already in Use		X			X									
	3. Incorrect Data Formats		X			X									
	4. Incorrect Instructions or Formats		X		X										
	5. Unstable Iterative or Non-Convergent Processes		X		X										
	6. Software System Violations	X	X												
	7. Others														

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>														
	IX. ERROR RECOGNITION AND RECOVERY (Continued)														
	E. Restart Procedures after Equipment Failure				X						X				
	F. Frequency of Periodic Dumping Technique For Restart				X			X							
	G. Amount of Storage Used by Dumping Technique				X	X		X							
	H. Monitoring Violations on Hardware Limitations	X				X									
	I. Suitability of Software Memory Protection Scheme	X						X		X					
	J. Use of Parity Checking Technique				X	X									
	K. Calculations for Tape Redundancy				X	X									
	L. Use of One Processor for Error Detection				X							X			
	M. Maintenance of Error Frequency Counts				X	X									

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>														
	X. DOCUMENTATION													X	
	A. Availability of Descriptive Manuals													X	
	1. Introduction to Operating System													X	X
	2. Operating System User's Manual	X	X											X	X
	3. Error Message Explanations and Recovery Procedures		X								X				
	4. Operating System Maintenance Manual										X			X	
	5. Internal Structure of Operating System		X								X			X	
	6. Programmer's Reference Manual for Operating System	X	X											X	
	7. System Training Manuals													X	X
	8. Available Library and Utility Routines	X	X											X	
	9. Maintenance of Program and Routines Library		X								X			X	

- THREE STEP METHOD FOR SOFTWARE EVALUATION -
 CATEGORY TWO: OPERATING, EXECUTIVE OR MONITOR SYSTEMS

REQUIREMENT	<u>OPERATING, EXECUTIVE OR MONITOR SYSTEMS</u>	<u>GENERAL MEASURES OF SOFTWARE CAPABILITY</u>	PROGRAMMING	CHECKOUT	COMPIATION	EXECUTION	I/O UTILIZATION	PRODUCT ECONOMY	SECONDARY STORAGE	GROWTH FLEXIBILITY	SIMULTANEITY	MAINTENANCE	INTERVENTION	INDEPENDENCE	DOCUMENTATION	TRAINING
	X. DOCUMENTATION (Continued)															
	B. Listing(s) Produced by Operating System		X													
	1. Programming Errors Detected		X			X										
	2. Machine Errors Detected		X									X				
	3. Names of Jobs, Programs or Tasks with Relevant Clock Times		X									X		X		
	4. Time Accounting for Components Relative to User Jobs											X		X		
	5. Manual Operator Instructions												X			X
	6. Memory Maps and Routines Executed		X									X			X	
	7. Events Occurring Relative to Operating System Functions		X									X				
	8. Storage Allocation History		X													
	9. Others															

SECTION IV
DESCRIPTIONS OF EXECUTIVE, OPERATING
OR MONITOR SYSTEM FEATURES

JOB AND/OR TASK SCHEDULING

Framework for Compilation, Assembly, Loading, Execution, and Appropriate Combinations

One of the important features of any operating system is the framework provided for operation of a variety of different types of jobs or tasks. Most installations require the flexibility to compile, assemble, load, or execute any of their specific problem programs. It must provide for a series of jobs of the same type (e.g., several different "compile" requests in a series should not result in reading the compiler into main storage each time, but neither should the compiler remain in main storage continuously thereby idling prime space) as well as provide for a series of jobs of different types in an appropriate variety of combinations (e.g., it may be necessary in some applications to load, compile or assemble tasks as part of a job whose major type is "execute"). The operating system should provide the framework in which the programmer may request any desired sequence of operations simply and with as few steps as possible. A well designed framework will reduce operator intervention as well as contribute to product economy.

Provision for Test Execution

Actual input data may not be available at the time a programmer is checking out his program. Appropriate characteristics of this data may be simulated and the data generated for test executions as an aid in program checkout. Data generators having parameters controlled by the programmer provide a means of producing this data. Provisions should be available for program execution using test data. It may be required by the user to allow for dynamic modification of data generation parameters. Generation routines may be called by the problem program. Definite savings in programmer checkout time may be realized by a well-designed and properly integrated data generation and manipulation scheme.

Transition from Job to Job

A smooth and efficient transition from one program or job to the next is an important function performed by an operating system.

This function is independent of any specific machine and is characterized by the fact that all relevant operating systems provide it and most users require it. In some cases, compilers are read in just before the next job is read in, which is wasted if the job under consideration is not a compilation. This could be an acceptable procedure if a large number of users require compilations. However, in a case where compilation is done rarely, this data transfer is costly and unnecessary. A smooth and efficient transition from one type of job to the next will tend to conserve machine execution time and operator intervention.

Facility for Running Several Jobs Serially

Some operating systems provide a flexibility whereby the input, output and computing requirements of several jobs are assessed at one time and then scheduled such that minimum overall time is utilized. If this is the case, several jobs are looked at before a schedule is defined. The number of jobs may be an optional parameter. Assess the degree of overlap allowed while considering the hardware limits and capabilities. This may affect the computer operator duties. Estimate the savings in compilation and execution time realized.

Facility for Running Jobs in Parallel

To what degree programs, systems or routines may be operated simultaneously depends on the objectives of the designer as well as the capabilities of the particular machine. In any case, a structure must be provided which will allow several programs or a continuous stream of jobs, tasks or programs to be run without interruption. Relatively little general programming development work has been accomplished to exploit parallel program operation and, as a result, paper proposals must be analyzed very closely. It is possible to realize savings in execution time as well as enhance the degree simultaneity of the system.

Utilization of Reentrant Coding in Programming System

Is the reentrant coding technique used within the programming system components (i.e., such that multiple requests will require only one program copy)? The operating system should provide a structure for operating programs within the programming system which are written as common subroutines. It should be realized here that the hardware must have the capability of multiprocessing to some degree, otherwise reentrant coding is of no use. Appropriate implementation of this feature would tend to conserve execution time and contribute to growth flexibility.

Dynamic Scheduling of Program Segments

The decision may be made dynamically as to which program segment will be executed next. Can the decision be made by the user-programmer? Is the use of this feature restricted to the operating system? Suitable addition of this feature to the operating system will tend to conserve programming time as well as contribute to growth flexibility.

Inclusion of Externally Programmed Jobs

Provisions should be available for execution of programmed segments or subroutines originally written for other installations. Facilities within the operating system for inclusion of externally programmed jobs will reduce the programming time, compilation time and tend to increase machine independence.

Run-to-Run Program Parameter Modification

What techniques are available for altering program parameters while maintaining continuous program-to-program transition? Once a program is checked out and considered to be ready for production, the user may require several such runs, each with one or more of its input parameters changed. In fact, it may be convenient to check out a program (after the initial or first phase of checkout) by setting up a series of runs changing one or more parameters at a time, to establish or verify its checked out status. Different operating systems require different rules or techniques for executing these runs. Appropriate addition of this feature tends to reduce checkout time and increase product economy.

Parameter Modifications for Dynamic Debugging

Decisions may be made dynamically as to which debugging component to use as well as allowing modifications to debugging parameters. During the production of large programmed system suitable application of this feature will tend to reduce checkout time and execution time.

Inter-Program Task Execution in Parallel

Can computational tasks, subroutines or segments be executed in parallel if they belong to different jobs? In systems having a multiprocessing hardware capability, a flexibility is sometimes provided the user-programmer to submit jobs in parallel. The function of the operating system is to provide the environment such that jobs may be done in parallel. Important in this area is what limitations have

been imposed and for what reasons. For example, is it possible within the user program to direct the operating system to begin reading in or reading out certain data portions in advance of their actual need. Simultaneity and execution time are the measures of software affected by this feature.

Intra-Program Job Execution in Parallel

Can a single job execute its own internal tasks in parallel (including computation) within the software system? Tasks may be executed within a certain job in parallel which would be considered multiprocessing if the tasks were strictly computation type and multiprogramming if the tasks were both computation and data transfer types. It may be possible to request that a certain subroutine, such as a square root routine, be done using a different central processor at this time even though it would not be needed until later; thus providing the user with a facility that could make his overall program run faster. This particular function has many ramifications and in some cases has been so limited that multiprocessing was impossible even when the hardware capability was present. This feature will affect simultaneity, execution time and growth flexibility.

Limitations of Parallel Buffering Operations

What limitations prevail relevant to buffering input/output operations in parallel within a single job? Among several jobs? Is it possible within the user program to direct the operating system to begin reading in or reading out certain data portions in advance of their actual need, thereby saving overall execution time. Appropriate application of this feature will affect I/O utilization and simultaneity.

Controlled Use of Common Subroutines

Is there a provision for executive control of common subroutines? In a multiprocessing environment more than one central processing unit may require operation of the same subroutine. The operating system must ensure each of the central processors a correct copy of the subroutine as well as providing each processor with its own temporary storage. The framework for controlling this function should be flexible enough to handle future central processing unit additions and associated equipment changes. This feature will affect execution time and simultaneity.

Provision for Future CPU Additions

What software modifications are required when additional central processing units (CPU) are added to the system? How many may

be added with only minor modifications such as changing numbers in a table? This feature will affect growth flexibility.

Temporary Storage Availability for Each CPU

Are provisions available for allocation and control of temporary storage for each CPU? Up to how many? In a multiprocessing environment each central processing unit (CPU) should have either its own protected temporary storage area or if main memory is used, a memory protection feature must be provided for each.

Control of User Program Timed Requests

Does the operating system, monitor and control I/O requests by timed interrupts? For jobs? For programs? For tasks? Continuous monitoring is required, in some cases, to provide the user with the capability of instructing the computing system to perform a particular function in a certain pre-specified time. In other cases a timed interrupt can be programmed to provide this monitoring. The requests could be relative to certain real-time devices, or to a specific standard I/O requirement, or even to a scheduled maintenance type request. Recognition of this feature in the operating system will tend to enhance I/O utilization.

I/O Data Transfer Monitoring

Is control and monitoring of I/O data transfers provided? What is required of the user-programmer? Within the programming system, requests for certain I/O data transfers could be controlled by timed interrupts. In most systems this monitoring and control must be done by a previously defined and agreed upon schedule of priorities. This feature will affect I/O utilization and programming ease.

Dynamic Task Compilation Requests

Can a user request and obtain partial compilations of internal tasks dynamically? How flexible is this feature? Certain on-line applications will find advantage in the capability of compiling on-line queries from remote consoles. Provision for this feature in the operating system will tend to reduce programming time and provide growth flexibility.

Automatic Library Program Manipulation

How much of the selection and manipulation of library program requests are performed automatically by the system? What is the programmer required to do? Automatic manipulation will tend to decrease programming time.

Scheduling of User's Own Library Programs

A programmed request could result in reading in a program from the user's own library of programs. This user may have a variety of different programs which upon his request would be compiled or executed within his current program, job, or task. Is there a provision in the operating system for scheduling user's own library of programs? Suitable application of this feature will affect both programming time and checkout time.

Monitoring of User and System Priorities

An initial request containing a low priority, in some systems, would get this priority raised providing certain prearranged units of time have passed without the program being executed. These priorities must be monitored, then, for all programs, both active and queued. This particular function may depend on other functions of schedule and control if the scheme used for scheduling and priority is complicated. What is the scheme for assigning priorities? Does the system control and monitor jobs and tasks for internal (programming system) and external (user) priorities? What is the difference between the limits of system and user priorities? In an environment requiring establishment of priorities, this feature would directly affect execution time and operator intervention.

Facility for Low Priority Program Upgrading

What facility is available for ensuring that low priority programs get executed (e.g., periodic increases in priority for un-executed jobs)? In an environment where many top priority programs are competing for hardware component use, low priority programs may find they are being completely excluded. Some means must be provided to allow running of low priority programs. This feature will affect time expended in program checkout.

Automatic Rescheduling for Priority Programs

In the event of high priority program requests, the operating system may be required to perform component rescheduling and rearranging of the entire computing system status in order to provide an environment for the execution of the high priority program. In other words, it could happen that with suitable rescheduling and reorganizing of the system status the high priority program may operate with only minor interruption to the program which it interrupted. This tends to make a more efficient or economical product.

System Reorganization Flexibility for High Priority Programs

What flexibility exists for system reorganization under future changes in the priority scheme? This feature will affect growth flexibility.

Checking of Program and Data Identification

In order that the correct priority scheduling of jobs and tasks may be done, the operating system must check the programmer's identification cards for records, and suitably generate any internal communication needs for other central processing units as well as establish logical switches so that appropriate monitoring of the entire computing system may continue. In addition, many installations require limited project numbers and insist on formatted or coded priority requests to ensure the accurate implementation of a particular priority standard. Is there sufficient internal system identification of jobs, programs and tasks, and their priorities to ensure proper completion of priority programs? Suitable implementation of this feature will tend to reduce time spent in checkout as well as make for a more economical product.

Complexity of Task, Program and/or Job Queue Tables

The scheduling of jobs, tasks and programs will require bookkeeping of certain identification data relative to each. This is dependent on the level of computation of the priority scheme used. It is the scheduling function that keeps these tables updated so that the proper sequence of programs can be executed. Are job, program and/or task queuing tables used? Is the structure of the queuing table sufficiently uncomplicated to allow efficient restart or path reconstruction after system failures? This feature will affect checkout as well as software maintenance.

Dynamic Maintenance of Queue Tables

Are these queuing tables maintained dynamically to continuously reflect the true and current status of the system? This will tend to decrease the time spent on checkout in addition to conserving execution time.

Priority Scheme Flexibility for Job, Task or Program

Does the priority scheme provide sufficient flexibility and complexity for the scheduling and manipulation of tasks and programs in addition to the priorities scheme provided for jobs? Can the user-programmer request variable priorities for his programs (within his

job) and/or his specific tasks, whether computation or input/output oriented? It is the flexibility of this particular scheme relative to the application under consideration which must be assessed. This will affect programming time and product economy.

I/O ALLOCATION, MONITORING AND CONTROL

Initiation of Program Delays on I/O Requests

In some cases, machine language I/O instructions are used by the operating system but unknown, for the most part, to the user program. Delaying certain operations until a channel is available or until a rewind is completed on a magnetic tape are functions executed within the operating system. There is a necessary balance here between savings in time and checkout problems for the programmer, on the one hand; and, on the other hand, necessary flexibility in the use of the I/O equipment. Are equipment delays for input/output processing controlled and initiated by the operating system? Describe the steps taken by the user. Appropriate implementation of the feature will decrease time spent in checkout and also increase efficiency in I/O utilization.

Continuous Interrupt Monitoring for User's I/O

Does the operating system provide continuous monitoring of interrupts on user's I/O? Certain real-time equipments in addition to many standard I/O devices may require continuous monitoring in order to provide the appropriate response time needed for a specific application. To what degree will the operating system provide continuous monitoring on any equipment contained in the proposed configuration? This feature may afford savings in programming time as well as effect efficiencies in I/O utilization.

Flexibility of I/O Devices for User Programs

The operating system may provide certain functions as well as a suitable framework so that user-programmers will find execution of data transfers with I/O devices less complicated to specify and check out. Is the complexity of I/O handling considerably reduced (for the user) by the operating system? Does the reduced complexity provide adequate flexibility in I/O device usage for the user? Suitable implementation of this feature may decrease programming time as well as provide for growth flexibility.

Future Additions of Standard I/O Equipment

Flexibility is often needed in the design of those systems having a symbolic I/O addressing capability, so that any particular installation which needs additional tapes, discs, drums, printers, etc., in the future should be allowed this addition with only a small modification to the operating system. It is common to start with minimum I/O and wait until more units are actually needed before they are purchased or rented. Does the framework for I/O manipulation provide for future addition of standard I/O equipment? This feature will affect growth flexibility.

Flexibility for Changes in Channel Assignments

In a system which provides the capability of assigning I/O equipments with symbols, the operating system must effect the conversion to channel and unit numbers. This function is essentially independent of the machine since it is required by any system allowing symbolic assignments of I/O equipments. Is the framework for I/O manipulation sufficiently flexible to allow for future changes in channel assignment? This feature will affect I/O utilization and growth flexibility.

Addition of Special I/O Equipment

Installations which at the beginning have special I/O equipment such as certain real-time devices, or plan in the future to add this type of equipment, do well to have a flexibility built into the operating system such that these additions may be made expeditiously. If the framework of the operating system has limitations not conducive to future predicted requirements, then a very large portion of the operating system may have to be modified when that equipment is added. This equipment may include different types of sensors or various types of long line communication equipment. Does the operating system framework provide for the possibility of future addition of special I/O equipment not initially required? Appropriate implementation of this feature will tend to enhance growth flexibility as well as I/O utilization.

Future Inclusion of External Communication Devices

What provisions within the operating system are available for the future addition of equipment which is necessary for connection to standard telephone circuit lines? Many installations may eventually be interconnected with phone lines. Programming system changes for this future inclusion may be prohibitive. However, the presence of this feature would provide open-endedness and thereby increase growth flexibility as well as contribute to the efficient use of I/O.

Buffer Area Reassignments for Modifications on Device Speeds or Capacities

The operating system should be sufficiently flexible to allow for changes in buffer area assignments required by hardware modifications or configuration changes due to various needs. Is the assignment of buffer areas for I/O transfers flexible enough to take advantage of future hardware changes on standard I/O equipment such as data transfer, speed increases or increased device storage capacities? This feature will affect I/O utilization and growth flexibility.

Increases in Simultaneous Channel Capability

Sufficient flexibility may be required to provide for future increases in the number of simultaneous channels used. Increasing the number of channels available in a system, resulting in more flexibility for allocating and controlling I/O equipment, may provide savings not realizable at the time of installation but rather may be appropriate for future considerations. Will the operating system internal structure allow for future increases of simultaneous I/O channels? This feature will affect growth flexibility, simultaneity and I/O utilization.

Decreases in Simultaneous Channel Capability

Sufficient flexibility may be required to provide for decreases in the number of simultaneous channels used. This capability should be contained in the operating system at the time of hardware selection. Will decreases in I/O channel simultaneity make the operating system overly cumbersome?

Efficient Conversion of Symbolic I/O Assignments

Is I/O symbolically assigned? If so, is the scheme adequate for the user's need? Suitable addition of this feature will afford savings in programming time as well as increase efficiencies in I/O utilization.

Flexible Symbolic I/O Assignment

Is the scheme for symbolic I/O assignment sufficiently flexible to allow for future additions of new and different types of I/O equipment? This feature will affect programming time and growth flexibility.

Monitoring of Entire Computing System Status

The operating system must know, at any time, the status of all components in order to effectively and efficiently monitor programs, jobs, or tasks. Does the operating system provide for continuous monitoring of the entire computing system configuration? How? This feature may afford savings in both programming and checkout time.

Status Updating Scheme Flexibility

Some operating systems provide tables, or a similar technique, so that status updating may be performed on each and every device or component in the system. Others may provide status updating for channels only. Does the scheme implemented allow controlling and updating of each I/O device or is it limited to channels? This feature will affect programming time and I/O utilization.

Technique for Allocating Input and Output Data Areas

One function of the operating system is to honor requests from user programs in the use of scratch tapes, input data tapes, areas of secondary and primary storage for temporary use. Does the operating system provide for allocation and control of scratch tapes, temporary storage areas and input/output data areas? This feature will affect programming time, I/O utilization and secondary storage.

Suitability of Algorithm for Equipment Scheduling

The algorithm for effecting equipment scheduling must provide for smooth and efficient operation. The degree to which it may be changed in order to better suit the user's needs must be assessed. How is the algorithm for scheduling equipment usage suitable for the user's needs? Can it be modified easily? Appropriate implementation of this feature will affect both programming and checkout time for the user-programmer.

Ease of Controlling Program-to-Program I/O

The internal structure must be adequate for effecting the efficient transition of program-to-program or job-to-job control transfers and input/output transfers. The operating system must be suitably organized to minimize unnecessary machine time losses to user programs. This feature will affect execution time and operator intervention.

Channel Control and Allocation Flexibility

Certain operating systems while providing reduced programming and checkout time in the form of already programmed I/O functions

may incur undesired limits or boundaries for users requiring extensive use of I/O. Does the system provide flexibility in allocation and control of I/O channels? This feature will affect I/O utilization and simultaneity.

Allocation of Input/Output Devices

In some installations it is desired to have the allocation scheme of symbolic I/O assignment include all actual input/output devices as well as channel assignments. Does the symbolic I/O assignment scheme allow allocation of the actual input/output device or is it limited to channel assignment? This feature will affect I/O utilization and secondary storage.

Remote Console Monitoring and Control

Remote consoles require immediate attention. Incoming data, messages and programs require the operating system to react in some way so that appropriate storage of this data, message or program may be saved for some time later on when execution may take place. Is continuous monitoring and control of remote console devices provided? This feature, suitably implemented, will allow savings in execution time as well as enhance I/O utilization.

Queuing of Remote Console Messages

Does the operating system store program, data and/or message queues from remote console devices? What functions must be performed by the programmer? How is the programmer provided access to data stored by the operating system? What options or controls may be placed on the operating system relative to these functions? Suitable application of this feature may decrease programming time and machine execution time.

Addition of Remote Console Equipment

How complicated are the changes required for future remote console device additions? Is it possible to add more of the type of device already proposed? Up to how many? What changes must be effected in order to add console devices with more desirable features or other increased capability? Assess the limits and boundaries of the technique within the operating system relative to future changes. In addition, these limits and boundaries must be determined in order to determine their effect on primary and secondary storage assignments. Growth flexibility is the primary measure of software capability affected by this feature.

USER AND SYSTEM STORAGE ALLOCATION

Technique for Program Segmentation

Many computing installations sooner or later face the need for execution of a large program. By large is meant that the size of the program instructions or statements is larger than available memory. Because of this, the programmer must segment his program into portions of program and data which will fit into main working memory. In those installations which do not provide a segmentation scheme, the programmer must himself decide what divisions he can make within the problem and write pieces of program accordingly. Of course, some programs cannot be segmented in this way and a more sophisticated segmentation scheme must be provided. For instance, in FORTRAN, some computing systems have defined and implemented a chain in which each segment of the program is a link and any link can call any other link. The technique of this program segmentation scheme implemented must be assessed as to its suitability to the application being considered. A well-designed scheme could provide more efficient utilization of secondary storage as well as provide a more economical product.

Manipulation and Activation of Overlays

A much more flexible and efficient computing system will be realized if the control and activation of overlaid segments is performed within the operating system. In some systems it is only the compiler system which will allow manipulation of segments. In case this manipulation and activation is provided within the operating system, all of the software components may then be provided the flexibility of segmentation. Does the operating system monitor and control the manipulation and activation of overlays? Appropriate implementation of this feature will affect secondary storage and growth flexibility.

Scheme Flexibility to Include All Storage Devices

What computing system storage devices in the proposed configuration are included in the segmentation scheme? Which ones are not? Inclusion of the scheme for segmentation with all proposed storage devices will tend to conserve programming time as well as provide efficient usage of secondary storage.

Facility for Protecting User's Data or Program

Some systems may provide a hardware feature for protecting each program segment whether it be data or program. In other systems a software scheme is implemented. This is necessary since a checked out and running program must be protected against programs in the

debugging stage. What facilities are available for protecting the user's data segments and/or program segments? These must be described for both primary and secondary storage. Implementation of this feature will affect programming and checkout.

Effectiveness of Relocatability Technique

Different relocatability techniques require implementation at different levels. For example, some techniques require calculation of symbolic relocatability at compilation or assembly time. Others are handled strictly by the loader program. Some program segmentation techniques require implementation through the compiler or assembler since insufficient hardware features are available in the configuration. What portions of the relocatability technique are performed at assembly time? At program loading time? By the operating system? Appropriate hardware features together with effective software scheme for relocatability will tend to reduce programming time as well as contribute to smooth and efficient checkout.

Provision for Absolute Address Assignment on Debugging Output

Some relocatability schemes make it almost impossible to determine exactly where segments of program or data resided in core during an already executed program run. In order to save execution time and contribute to efficiency during checkout a provision should be implemented which will allow programmers to determine exactly where segments of program or data resided during execution. Are there provisions for furnishing debugging output with the actual memory locations used by each instruction during execution?

Limitations on Reconstructing Program Paths

In some programming systems it is required to insert steps in the program in a very well-specified way so that path reconstruction may be performed. What pre-execution steps must be taken to ensure complete reconstruction of program paths during debugging operations? The actual steps which must be taken by the programmer must be assessed, since it may be apparent that these steps are an unnecessary burden relative to other computing systems.

Ease of Monitoring Storage Requests and Returns

Most storage allocation schemes allow user-programmers to request blocks or units of storage and in some similar way allow for these same units to be returned. It must be assessed how much of the allocation processing requirements must be performed by the user-

programmer? Efficiency in this area will tend to reduce programming time as well as provide efficient utilization of secondary storage.

User Flexibility of Requests and Returns

Does the storage allocation scheme extend to secondary storage? Are similar requests and returns useful for these secondary storage areas? What steps are needed for requesting and returning of primary and secondary storage areas? A well-organized and flexible scheme allowing the user use of the entire machine configuration will tend to make the programming task easier and therefore take less time. Also, efficient utilization of secondary storage may be effected by appropriate implementation of an organized technique.

Availability of Storage Map and Assignment Tables

A well-organized scheme for storage allocation will include a map of primary and secondary storage probably in the form of some type of table where the programming system controller may continually update and maintain status for user assignments and post analysis. In some systems this table, or whatever, is available to the user-programmer and as such, allows him greater flexibility. Is the map of primary and secondary storage and table of user assignments readily available to the user-programmer? This will afford him efficiencies which will tend to decrease his programming and checkout time.

Data Arrays and Table Relocation Flexibility for Priority Requests

Can existing tables and arrays of data be suitably relocated for high priority programs? How does each user-programmer know where his data is during execution? During debugging? Suitable implementation of this feature will tend to ensure the compilation or execution of high priority programs which would otherwise require operator intervention.

Effects of Priority Requests on Segmentation

It must be assessed here whether the operating system does indeed save the appropriate machine conditions, under high priority interrupt, such that the interrupted program will not have to be completely redone. What effects do priority programs have on program segmentation? Appropriate saving of completed work will effect savings in execution time.

Manipulation of Job Segments To and From Secondary Storage

Does the operating system manipulate and control the transfer of job segments to and from secondary storage? What is required of the user during execution of his program in order to effect this manipulation and control if not done by the operating system? What internal communication switches or tables are affected by priority requests during program segmentation operation? Assess the flexibilities afforded the user of the segmentation technique in this area. This feature will affect programming time, execution time and secondary storage.

Similarity of Job Segment Technique to Library Routine Manipulation

Is the segmentation technique similar to and compatible with the technique used for library routine manipulation? Appropriate implementation of this scheme will tend to decrease programming time as well as contribute to growth flexibility.

Buffer Area Determination for I/O Assignment

Does I/O assignment by the operating system include buffer area determination? That is, does the operating system provide determination and allocation of buffer areas for each I/O request? How is it done? It must be assessed whether this allocation is done well enough to effect efficient data transfers. This feature will affect I/O utilization as well as product economy.

Algorithm Suitability for Providing Each of Several Jobs With Fast Buffers

Are buffer areas reassigned when the number of programs increases? That is, when the number of programs using I/O increases, the operating system may cut down on each program's block size for buffer area assignment. Also, is the buffer area assignment reapportioned when the number of programs becomes smaller? Appropriate implementation of this feature will affect the efficiency of I/O utilization as well as provide savings in execution time.

Flexibility of User Requests for Buffer Areas

Can the user effect buffer area assignment if needed? That is, in the case that the operating system fails to provide sufficient flexibility in buffer area assignment, how difficult is it for the programmer to effect changes in buffer area assignment? This feature will provide assistance in growth flexibility.

Effects of Changing Buffer Area Requirements on Allocation Scheme

What effect does the changing of buffer area requirements have on the allocation scheme? Is less space allowed the user-programmer? Can the allocation scheme still be used when the programmer is effecting his own buffer area assignments? This feature will affect programming time and the use of secondary storage in addition to affecting product economy.

Flexibility of Allocation Scheme to Machine Configuration Modification

Is the allocation scheme sufficiently flexible to take advantage of hardware configuration changes such as the addition or deletion of memory modules? More secondary storage? This feature will affect programming time and secondary storage?

Buffer Area Association Problems with Increased or Decreased I/O Configuration

Can the buffer area assignment scheme be easily modified to take advantage of any increased or decreased I/O equipment changes made to the computing system configuration? Describe the steps required for modification in cases where configuration changes may be required. In cases where configuration changes are required but suitable flexibility in buffer area assignment is not forthcoming, additional programming and checkout time will be used to effect the required change.

Job, Program and Task Initialization

Does the operating system provide initial system status setup for jobs, programs and/or tasks? Initialization not done by the operating system must be done either by the programmer or by the computer operator. Required initialization must be assessed by looking at the requirements of each hardware component. After that, it must be determined who should perform the initialization - the programmer or the operating system. This feature will affect programming time, operator intervention.

Component Initialization Facility (Tape Rewind, Label Checking, etc.)

Are specific I/O components initialized or put on ready status by the operating system (e.g. tape label checking, deleting identification records, selecting proper operating modes, etc.)? This feature will affect programming time and operator intervention.

Flexibility of Running Combinations of Foreground and Background Jobs

Can a mix of foreground and background jobs be executed? Describe the limitations. (Note: In a multiple-user computing system, foreground programs are those real-time programs requested by on-line users. They usually require small pieces of time. Background programs are those programs requiring much more time, possibly for long compilations, which are run in the background as lower priority type programs.) Available time on the computing system will be more efficiently used by appropriate mixing of foreground and background programs thereby enhancing the product economy.

Flexibility in Selection of Combinations of Debugging Operations

Can appropriate combinations of debugging operations be selected to handle predicted checkout needs? It may be required by certain applications to trace small sections of a program and effect program dumps of portions of the program in suitable combinations. Suitable implementation of this type of request will tend to reduce checkout time as well as afford savings in execution time.

Suitability of Commands for Debugging Operations

Are the debugging operation commands adequate to provide requests for snapshots, partial dumps, segment dumps, traces and intermediate results? Suitable application of this feature will tend to reduce checkout and execution time.

Allocation of Storage for Output of Snaps, Dumps, Traces and Intermediate Results

Does the operating system monitor and allocate storage for snapshots, partial dumps, segment dumps, traces and intermediate results? What is required of the programmer when utilizing these debugging operations? What is provided by the operating system? Suitable implementation of techniques to assist the programmer will provide savings in programming and checkout time as well as allow more efficient use of secondary storage.

LIBRARY MANIPULATION AND MAINTENANCE

Flexibility of Framework for Library Access

The framework within the operating system for the manipulation and maintenance of library routines must be flexible. It

should be sufficiently flexible to allow the addition of new subroutines, subprograms or portions of the programming system, modification to those routines or even complete deletion. Some commonly used routines require more input parameters than others, also some programs require a separation of the actual instruction code from the data. Provisions within the structure or framework of the operating will tend to decrease maintenance and operator intervention.

Flexibility of Calling on Library Routines During Compilation, Execution, Etc.

Is the structure sufficiently flexible to provide for calling library routine during compilation? Execution? Both? What specific flexibilities are provided in the framework of the operating system which allow for easy access to library routines? Does the operating system communicate with library routines using similar linkages as are used with compilers and other large software packages within the system. This feature will affect operator intervention, maintenance and ease of programming.

Instruction Linkage Between Routines

The method used for linking various types of routines, subroutines, and library programs should be sufficiently open ended to include the possibility of future additions. What is the method used for linking user programs with library routines? With compilers or assemblers? Flexible and yet uncomplicated linkages will tend to reduce compilation and execution time.

Requirements on User-Program for Instruction Linkage

Identify the steps required by the user in order to construct linkages to the various types of library routines. This feature will affect programming and product economy.

Reading Routines Automatically Provided

Are the library routines automatically read in and incorporated into the user's program? If not, what must the user do? If they are provided automatically, what options are available to the user? Suitable application of this feature will affect programming and product economy.

Repeated Requests for Same Library Routine

Are appropriate linkages set up by the system so that more than one request for the same library routine will produce only one copy in the requested program? What flexibilities are available in this area? This feature will affect product economy.

Library Routine Data Array or Table Needs Provided

Many library subroutines must be given certain input parameters or data in order to operate. This data must be provided by the system or by the user. What library routines need separate data arrays or tables? For each, describe how these items must be provided. By whom? Suitable implementation of this feature will tend to decrease programming time and maintenance.

Monitoring Utility Routine Control

Some library routines require I/O channel or device control in monitoring. It must be assessed where this control or monitoring is done. That is, within the operating system or by the user. Are there any requirements for control or monitoring on any of the library routines? Who performs the monitoring in each case? What affect does this have on the user program? This feature will affect programming and I/O utilization.

Standard Scheme for Transfer To and From System Routines

Is there a standard scheme for control transfer between the library routines and the operating system? Describe it briefly. Must the user know how to transfer control or is this handled for him? This will affect programming and I/O utilization.

Linkage Substitution for Program Segmentation

Linkages between the segments must be generated as well as manipulated by the programming system. What requirements are placed on the user for affecting program segmentation of library routines? This will affect programming and execution time.

User Requirements for Library Routine Segment Manipulation

What steps must be taken by the user in order to affect library routine segment manipulation? Does the system provide linkage substitution in case library routines require program segmentation? Clearly distinguish between functions performed by the programming system and those performed by the user programmer.

Provision for Updating Library Easily

Once a library has been generated and its manipulative routines have been checked out, it should be fairly easy to use. Also, it should be fairly easy to maintain; that is, to modify existing routines, add new routines, or delete old ones. What provisions are included for allowing the maintenance and updating of library routines? Can user programs do it easily? This will affect maintenance.

Non-scheduled Library Maintenance

In case a new library routine fails in some way, what procedure must be performed to replace it with the old one or effect a suitable modification? Can this be done on-line? In either case, how is continuous system operation maintained? This will affect maintenance and execution time.

Suitability for Operational Reliability

Are there any special features available for maintaining operational reliability of library routines? Describe each briefly along with a reason and the extent of the affect on reliability. Consideration should be given library routines which require input data within certain ranges or parameters of specific values. What happens to each of the routines in case the range of input data is exceeded or the wrong parameter is entered? This will affect check-out and maintenance.

Maintenance Jobs Treated as User Programs

Can library maintenance jobs, programs or tasks be submitted to the operating system as any normal user program? If so, what scheme or technique is used to insure a program obtaining the desired library routine version? If not, what special action or special operating systems are needed? Appropriate application of this feature will affect maintenance.

Completeness of Routines for Library Manipulation and Maintenance

Does the operating system include the utility routines needed to both manipulate and maintain the system library? If not, what is needed? Will these be supplied by the vendor or by the user? This will affect growth flexibility and maintenance.

EDITING OF USER AND SYSTEM PROGRAMS

Listings for Program Post Analysis

Facilities must be provided programmers for analyzing results of trial runs. Many of these facilities should be incorporated in the operating system so that the programmer may have sufficient information to reconstruct the events and paths occurring during the trial run. Notes are provided to programming system users to communicate between the various programming system components such as compilers and other large software components. Naturally, this depends

on the design and implementation of the operating system. What listings are provided for post analysis by user programmers? This particular feature will tend to decrease checkout time, if suitably implemented, as well as decrease execution time in the form of re-runs in case additional information is needed which is not available on a printout.

Event Oriented Output for Ease of Reconstruction

Events occurring during trial runs may indeed assist the programmer if they are clear and completely defined. Are the messages produced by the operating system oriented toward distinct and well defined events so that accurate reconstruction may be performed? Adequate and flexible implementation of this feature will affect time spent in checkout.

Contiguous Output Message Listing

Messages may be produced by several pieces of software within the programming system. That is, the compiler may produce messages relative to the compilation process, inappropriate use of assembly language may produce output messages, as may each of the different software components. Does the operating system provide a contiguous and unambiguous output message listing so that the programmer may easily reconstruct what happened during a trial run? This will affect programmer checkout time.

Suitability of Message Scheme (Descriptive or Numbers only) for Event Recording

In some message output schemes numbers only are produced and used as reference to a document which describes the intention and the meaning of that message. Other systems provide the complete message on the output listing. Advantages of the number only system are in savings of computer time and space for the message during processing as well as output printing time. However, the complete description contained within the output listing is much more convenient for the programmer to use during checkout. Which scheme is more advantageous will depend on the particular application being considered. Does the output message consist of a description? Is it complete within itself or is a further description needed from an associated document? Does the output message consist of a number which refers to a description in some associated document? If so, is the document available? The completeness of each message must be assessed also in order to determine the effectiveness of the overall scheme. This feature will affect programmer checkout time and machine compilation time.

Messages Compatible With Other Software Components

Is there an organized scheme within the software programming system concerning the production of the message descriptions so that each term has a distinct meaning? Each message must be assessed as to whether there is a conflict of meaning. Some programming systems, during production, have been separated such that separate groups define output messages relative to compilers, others specify output messages relative to report generators, etc., and very little is done to coordinate message compatibility across the range of software components. This will affect programmer checkout time in attempting to assess just what the meaning is of each message and also will affect machine time since re-runs may be required in order to determine what the meaning was.

Provision for On-Line Error Correction

Does the list of output messages include a facility for notifying the operator immediately when an on-line correction may be made instead of aborting a program? On some computing equipments a ready status must be received from some components before data transfers may be effected. If the ready status is not received by the programming system two things can occur: 1) the job can be aborted and 2) the programming system could provide a message on-line to the operator such that he may then ready the status of the component in question and the job continued normally. Each component must be assessed separately relative to its needs for on-line message output. This will affect operator intervention as well as programmer checkout time.

Flexibility of Machine Language Patching

Patching machine language object code can save compilations. Trial runs may be made with patches rather than with recom compilations thus saving machine time. What provisions are available for making machine language patches? Can they be made easily to the object program? This feature will affect programming time and tend to reduce compilation time.

Sufficient Information for Adequate Path Reconstruction

Does the output message listing contain sufficient information within itself to adequately reconstruct the program path? If not, what else is needed? What is required of the programmer in order to obtain this information? This will affect both programming time and checkout time.

Flexible Code Substitution for Debugging Runs

The operating system should provide the code substitution for each debugging software package available. This allows for centralized control which makes for efficient checkout. Does the operating system provide efficient code substitution suitable for available debugging schemes? This feature must be assessed relative to each debugging scheme provided in the programming system. This feature will affect checkout time and execution time.

Flexibility of Formatting Debugging Options

Are provisions available for making compilations with or without debugging aids? What modifications are required by the user to delete debugging aids from a program? That is, if a program has been run and structured with debugging aids is it a simple process to delete use of these debugging aids? Describe relative to each debugging routine. This will affect checkout and execution time.

Contiguous Listing of Recorded Errors

Machine errors recorded by the operating system should be listed separately such that hardware maintenance personnel may be able to perform their function without searching through long listings of programs. A separate listing should be provided but with the machine errors in a contiguous and time ordered occurrence. This will affect maintenance.

Monitoring of Out-of-Range Quantities

Out-of-range quantities such as data calculation overflows or arithmetic operation violations are sometimes monitored by software within the operating system to assess their effect on the program being checked out. Are there any provisions for monitoring and editing of out-of-range quantities during debugging? This will affect program checkout.

Recording of Out-of-Range Quantities

Data calculation overflows or arithmetic operation violations must be recorded for programmer reconstruction. Are arithmetic violations and data calculation overflows detected and recorded during program execution? During debugging? Are they also recorded during compilation in those cases where compilation calculations may occur? This will affect programmer checkout time and execution time.

FACILITY AND USER TIME ACCOUNTING

Monitoring Job and Program Timing

Some installations require each job run in the facility to be accounted for separately under separate projects. Others require, within a job, that certain programs be timed separately with control afforded both to the operating system and to the job specifier. Does the operating system perform monitoring of job timing for both user and facility? In installations requiring the accounting of job and program training, absence of this feature will require the user to implement it himself. This will affect programming and checkout time as well as machine execution time since without it the installation must stop operation of any job, account for the time and then start the next one.

Maintenance of Components Used Table

Is a table maintained of timing information of each hardware component on the computing system? If not, is one maintained for any of the components? Which ones? Identify each and describe specific accounting of time. This will affect programming time and product economy.

Provision for Generation of Statistics

Is there a framework in the operating system which will allow any specific installation to specify what statistics they feel are necessary to generate? Is it open ended or are there only certain specific statistics generated? Statistics are useful to a variety of people within an installation and can effect the amount of time expended on overhead functions. If the framework exists then programming time may be saved in those installations requiring it. Some of the statistics which may be generated are listed below. Savings in overhead functions may be realized knowing how often certain functions are used.

1. Number of Secondary Storage Transfers--Any method of assessing the number of secondary storage transfers made will allow maintenance personnel to assess the overhead time spent by segmenting programs or data in large, medium or small pieces. This will allow the maintenance personnel to suitably adjust the size of programs or data.

2. Number of Storage Accesses--Also affecting 1. above may be the number of actual accesses made as opposed to the number of transfers made; that is, if a large number of programs access the secondary storage for only a small number of distinct data items transferred then this will affect the way the operating system should make these data transfers.
3. Number of Jobs, Programs or Tasks Executed--The number of times certain tasks or programs within a job are executed may be required in some real-time installations. Gathering of this type of statistic controlled by the job specifier would be especially useful in post analysis.
4. Frequency of Utility Routine Usage--Placing library routines on a magnetic tape in an order which will tend to save execution time waiting for tape library routine transfers is a well known problem. This statistic would allow the installation to assess the number of times each library routine is used and therefore allow maintenance personnel to place these routines appropriately on a library tape to minimize execution time in waiting for library routines to be read in.
5. I/O Channel Time Used--In installations where several channels are available and any one may be used to connect the central processor with certain I/O devices, it would be desirable to know the time that each of those channels were connected and performing transfers. This statistic may be generated by software and allow future additions or deletions of channels to increase I/O utilization efficiency.
6. I/O Device Timing Used--Appropriate recording and maintenance of statistics relative to I/O device usage would allow for assessing the need for additional I/O devices both in increasing the number of each type of device and in reducing the number of each type of I/O device.
7. Others--Are there any other statistics that may be generated? Describe each briefly and relate what specific purpose may be served by its inclusion.

Execution of Job Abortive Procedures

Once the decision has been made to abort any specific job, it will be wise to include all the reasons used for aborting it as well as the conditions which prevailed at the time of the abortion procedure. Does execution of job abortive procedures include sufficient data to determine exactly what happened? Describe what information is included in the job abortive procedure. This will affect programmer checkout time and execution time.

Computing System Components Status Updating

Is an internal status kept relative to each computing system component? Is there means for continuously maintaining the status of each computing system component? Is this status accessible by the user program? Can the user program utilize this data to effect his own scheduling of components? Must this be done through the operating system?

Recognition of Component Sharing by Jobs

Does the scheme for maintaining components status and timing recognize and reflect the sharing of components by jobs, programs or tasks? That is, does the status include the possibility of this component being used alternately by one program, two programs, etc.? How does one program know when a component is being used by another program? This will affect I/O utilization.

Maintenance of Operations Log

An operations log is useful for maintenance personnel in order to assess what may have happened to either hardware or software. Also, it is useful during checkout for programming personnel. Is an operations log maintained? Describe the items which may appear on it. Is each component represented?

Suitability of Log to Management Needs

Does the operations log contain sufficient information to assess what jobs used what components, and for how long? In a multi-processing environment it would be necessary to know to what degree the processors were being shared and what percentage of the time were the processors used in parallel. The capability to assess the degree of overlap by each of the components will tend to increase the efficiency of I/O utilization as well as providing a method for assessing the degree of simultaneity used.

Flexibility of Log for Predicting Future Configuration Changes

Does the log contain sufficient information to facilitate predicting future estimates of configuration changes such as increases or decreases in numbers of I/O devices or channels? What additional features are provided?

Separate Time Records for System Components

Is a separate time record provided for each major component of the hardware system? Describe each different method briefly. Is this time record part of the operations log or is it a separate record?

Sufficient Details for Component Timing

If no individual component timing is performed, are there sufficient details maintained in the operating system for future addition of this capability? Is the structure within the operating system flexible and sufficient such that future requirements of component timing may be added? This feature will affect growth flexibility.

GENERATING AND UPDATING THE MASTER SYSTEM

Ease of Modifying Individual Programs

Individual components of the programming system must, periodically, be altered or deleted. Also, appropriate additions may be made to the overall programming system of newly developed or acquired software components. In any installation, where development of these software components is a continuing activity, it must be easy to make these types of modifications. What features are available so that individual utility programs may be modified or added easily? Programming time expended as well as maintenance may be affected by this feature.

Flexibility of Organization Scheme for Making New Masters

In most installations reorganization of the component programs, subroutines and software components must be made periodically to maintain efficient use of the programming system. Library routines must be reorganized or placed in different locations such that the most common used routines are closest to the beginning of the tape. Is the organization of the master system sufficiently flexible so that new master tapes may be made up with individual programs contained in

a completely different sequence? Describe the flexibilities available. This will affect product economy and maintenance.

Suitability for Updating Compilers, Assemblers, Etc.

Does the system provide for easy updating and maintenance of compilers and assemblers by system programmers?

Maintenance of File Systems

Some installations require generation and maintenance of many large file systems. Manipulation of these file systems may best be done in the operating system. Programmer time may be saved in addition to checkout time by the appropriate manipulative functions being resident in the operating system. Are provisions available for maintaining and updating users files or file systems?

Updating File Directory Tables

File systems are often organized around directories which make the alteration of items in the file easy to effect. Are directories of files or file systems maintained by the operating system? Describe the flexibilities available relative to this area. Programming is saved in addition to checkout time by suitable implementation of this feature.

Common Access of Often Used Files

Does the framework of the files or file system maintenance allow for easy access to these files during execution? During compilation? By other appropriate major software components? In some systems files must be accessed during compilation in order to save programming and checkout time. What steps are needed by the programmer in order to make these files available during execution and during compilation? Can the programmer use files generated for some other program without modification?

Compatibility of File System with Compilers, Assemblers, Etc.

Is the file system structured such that the programming system itself may utilize the advantages of information storage and retrieval through the file system? If the structure is sufficiently standardized then any component (compilers, assemblers, etc.) may access the same file thus saving regeneration of the file for each software component. Suitable implementation of this feature will tend to decrease programming and checkout time.

OPERATOR AND OFF-LINE COMMUNICATION

Organized Interface with Off-Line Computers

Many installations load input jobs or tasks off-line as well as print or punch off-line to conserve time in the main computer. The off-line device is a small computing system capable of a variety of data processing functions. Some logical interface and agreement in terms must be established to effect a transmission between the computers. An organized and standardized data configuration in command interpretation must be obtained if accurate and efficient job processing is to be realized. At the interface between the operating system and the off-line computers, are the commands organized and suitable for efficient data transfers? This feature will affect operator intervention and I/O utilization.

Utilization of Similar Terminology

Terminology used within the design of the main computer's operating system should be the same as that used within the peripheral computing system. Use of dissimilar terminology may require unnecessary data conversions or other instructional coding conversions in order to relate proper meaning. Is similar terminology used in communication by both the on-line operating system and the off-line computing system or systems? Appropriate implementation of this feature will tend to decrease execution time as well as operator intervention.

Standardized Data Configuration

It must be determined what data conversions are made at the interface and also the exact nature of these conversions in order to determine their affect on either system. There should be a standardized data format designed with the two systems in mind. Does either system require unnecessary data conversions? Is there a standard data format? This feature will affect execution time as well as operator intervention.

Use of Standard Commands

Commands used by either system for their own internal purposes or for purposes of communication with the other system should be designed in an organized manner. Are the commands used between systems standardized in this manner? Each command should be described and an explanation of its contribution to the overall installation should be included. An organized design of the command structure will tend to decrease execution time and operator intervention, as well as increase the degree of growth flexibility.

Communication with Other Computer Modules

In cases where computing system hardware provides flexibilities for addition of other central processing computer modules, flexibility in the operating system should be provided. In some cases future installation requirements will specify connection to other complete computer systems. The operating system should provide flexibility for communication to any of these types of system for which there exists a corresponding hardware flexibility. Is there a provision for communicating with other on-line computers or other computer modules in this system? This feature will affect growth flexibility.

Open Endedness of Communication Scheme

Is the communication scheme used sufficiently open ended to provide for future additions of computer modules, on-line computers or off-line computers. In some cases, a system may provide the communication but does not allow a large enough number of the different types of devices to accommodate expected future additions. This number must be decided upon by a user evaluator team and the operating system must be assessed in order to determine whether the system will accommodate this number. This feature will affect growth flexibility.

Allowance for Reflecting Equipment Configuration Changes

Expected future changes in any installation may require additions or deletions of equipment components in either the on-line computing system or in the off-line computing system or systems. Is there sufficient flexibility to allow for changes in the configuration of either computing system? Determine the actual upper and lower bounds for each type of equipment proposed in the configuration as well as describing upper and lower bounds for any other types of equipment available but not proposed. This feature will affect growth flexibility.

Controlling Communication With Subsystem Monitors

Has a master-slave relationship for controlling communication been established between the on-line operating system and any sub-system monitors? Does this relationship include the possibility of future additions of more off-line operating systems? This feature will affect growth flexibility and operator intervention.

Generation of Manual Operator Commands

The operating system should have the provision for generation of commands to the computer operator in order to relate specific manual operation instructions. Does the operating system generate commands for the manual operation functions? This feature will affect operator intervention.

Console Communication of Requests and Messages

Appropriate communication of requests and messages between the operating system and the computer operator should be established. Are the commands to the operator communicated to a central console easily accessed by the computer operator? Can the operator request or provide answers to the operating system through that same central console? Is sufficient information provided to the operator for each message? Does the operator have sufficient answer-requests to keep the operating system aware of the hardware status? This feature will affect operator intervention.

Updated Record of I/O Device Assignments

The operator must have sufficient information available to him so that he may do his job quickly and accurately. I/O device assignments are one of his primary responsibilities and he must know what the operating systems assignments are. Does the operator have access to an updated record of I/O device assignments? This will affect operator intervention.

ERROR RECOGNITION AND SYSTEM RECOVERY

Continued Operation During Limited Failures

In some installations, the operating system ceases operation when a component becomes inactivated. This will occur no matter whether the component in question is being used or not. It is important to some installations to continue operation even though certain components become inactive. Operating systems must monitor input/output components in channels to continually ascertain whether failures have occurred. This function could be sufficiently flexible so that a failure of a device not used or needed by the particular program now running would not affect that program. Further it may be that several of this device type are available, and switching to the use of a different one could allow the program to continue. For example, if several units are available as scratch tape the operating system could (under certain circumstances) switch to another if one fails. Does the operating system provide for continued operation when limited failures occur on components not currently being used by this job? To what extent can this occur? Describe the extent relative to each component proposed. This feature will affect operator intervention as well as execution time.

Switching Like Components to Maintain System Operation

Does the operating system automatically substitute a similar unused I/O device for one which has failed? Was system operation interrupted during this switch? This feature will affect execution time, I/O utilization and operator intervention.

Recognition of Errors Relative to Each Available Component

In the case where individual components do fail and are being used by the particular job program or task being processed, errors should be completely described relative to computing system status. Are errors identified relative to the individual component or device that failed? This will affect execution time, I/O utilization and operator intervention.

Detection of Programming Errors

The operating system must continually detect errors made by user programs such as incorrect input/output request specifications as well as hardware system errors. Also these errors must be recorded appropriately for programmer post analysis, reconstruction and debugging activities. Are programming errors detected during program execution? If so, identify each as to one of the following types:

1. Incorrect I/O Requests--This feature will affect checkout and I/O utilization.
2. Request for Equipment Already In Use-- This will affect execution time and I/O utilization.
3. Incorrect Data Formats--Detection of this type of error will tend to reduce checkout time as well as enhance the efficiency of I/O utilization.
4. Incorrect Instruction or Formats--This feature will affect checkout and execution time.
5. Unstable, Iterative or Non-convergent Processes--This will affect checkout time and execution time.
6. Software System Violations--The operating system may indeed detect certain violations defined and required by other software component packages such as compilers. A structure should exist within the operating system to allow for detection of these types of errors. Provision for detection of this type of error will affect programming and checkout.
7. Other Types--

Restart Procedures After Equipment Failure

Some organized and appropriately designed procedures should exist for restarting after various types of equipment failures. One way of course is to re-run the entire tape, or whatever, of jobs. However, this may be very inefficient in machine usage. Does the operating system have a capability of restarting a system after an equipment failure has occurred, without completely re-running a batch of jobs? Describe briefly. This feature will affect execution time.

Frequency of Periodic Dumping Technique for Restart

If the system provides periodic storage dumps as a method of easy recovery, what is the period? What happens when failure occurs during data transfer of this storage dump? Periodic saving of entire primary and secondary memory status on magnetic tape is a rather commonly used technique in this area. At any rate, storage on some secondary storage device is usually required in cases where restarting is desired. This feature will affect execution time and secondary storage.

Amount of Storage Used by Dumping Technique

What amount of primary and secondary storage is required to utilize this dumping procedure? That is, there is usually a program resident in primary storage. What is its size? Also, secondary storage media must be provided in order to store the memory map or maps. Determine the limits, etc. This will affect execution and secondary storage as well as I/O utilization.

Monitoring Violations on Hardware Limitations

Specific violations, such as magnetic tape running off the reel, are sometimes monitored by operating systems to provide additional assistance to the programmer. What specific violations of hardware limits are monitored by the operating system? Identify each and describe its effect on the system as well as requirements by the user-programmer. This will affect programming time and I/O utilization.

Suitability of Software Memory Protection Scheme

There must be some recognition of memory protection features within the computing system proposed. Many systems provide software schemes with a minimum of hardware features which together provide an adequate memory protection scheme. Is there a software implemented scheme for memory protection? Does it provide for both

program and data protection separately? How many distinct areas of each type can be protected at one time? This feature will affect programming and secondary storage as well as simultaneity.

Use of Parity Checking Technique

Is there a parity checking technique used for error recognition? On what components? This will affect execution time and I/O utilization.

Calculations for Tape Redundancy

Does the operating system execute tape redundancy calculations? Automatically or by user requests? Describe steps required by the user-programmer in order to utilize these calculations. This will affect execution time and I/O utilization.

Use of One Processor for Error Detection

Is one computing module used strictly for error detection? Is that program part of the operating system? Is communication provided by the operating system between this program and the operator? This will affect execution time and operator intervention.

Maintenance of Error Frequency Counts

Does the operating system maintain a list or table of counts of error frequencies? Is this table continually assessed by the operating system to determine certain threshold limits beyond which component usage must be aborted? This will affect execution and I/O utilization.

DOCUMENTATION

Availability of Descriptive Manuals

Different vendors provide manuals concerning their computing systems in different ways. One vendor may organize all of his manuals such that each software item has a section oriented to training as well as one oriented to reference material. Others may have written separate manuals for training, for reference, for internal structure and for system maintenance. The following documentary material may be required by the user and if so, it may be found in a manual with the same title or in a manual oriented differently and therefore have some related but different title.

1. Introduction to Operating System

2. Operating System User's Manual
3. Error Message Explanations and Recovery Procedures
4. Operating System Maintenance Manual
5. Internal Structure of Operating System
6. Programmer's Reference Manual for Operating System
7. System Training Manuals
8. Library and Utility Routines
9. Maintenance of Program and Routines Library
10. Others?

Listing Produced by the Operating System

Most all of the listings produced by executive or operating systems provide a measureable degree of documentation for the installation. These listings assist programmers and analysts in checking out programs, and maintaining programs as well as providing a record of the results of runs made on the computing system. The following items, if needed by the user, may be found in one complete listing or in one of several different listings but should be presented in a consistent manner such that reconstruction of events occurring during program execution are easy for the programmer or analyst.

1. Programming Errors Detected
2. Machine Errors Detected
3. Events Occurring Relative to Operating System Functions or Events
4. Names of Jobs, Programs or Tasks which were Compiled or Executed with Relevant Clock Times
5. Time Accounting for Hardware Components Relative to User Jobs
6. Manual Operator Instructions
7. Table of I/O Devices and What Jobs are Using Them
8. Others?

SECTION V

QUESTIONNAIRE

INTRODUCTION

The questionnaire contained in this report has been designed to assist evaluators. The totality of items constitutes a composite of questions concerning characteristics or functions which are of special interest when performing comparative analysis of software systems designed and built for a variety of computing systems. In any particular EDP application, an evaluator would select questions which are oriented toward features of particular interest to the application or installation being considered. This subset of questions would then be included in the specifications or requests for proposal sent to computer manufacturers. Part of each proposal would contain a suitable set of appropriate answers, explanations or further descriptions concerning only those features of particular interest. Next, evaluators would compare each vendor's answers with the known requirements in order to place a specific value judgement on each feature, each grouping of features or on the general category of operating, executive or monitor systems.

RESPONDENT'S NOTE

In some cases, a simple "yes" or "no" will be sufficient for an answer, but explanation or further description should be provided whenever necessary for an accurate and complete understanding. It is expected that any explanations supplied would require less than a few hundred words although a few special cases may need more.

Appropriate references should be indicated with answers, whenever necessary, and a copy of the referenced document (or its complete description) should be submitted.

FUNCTIONAL DIVISIONS

Job and/or Task Scheduling

1. Does the framework provide for Compilation, Assembling, Loading, and Execution options as well as appropriate combinations?

2. What provisions have been made for program execution using test data?
3. Has a test data generator been provided? What steps must be taken by the programmer to use it? Can it be called during problem program execution?
4. What features are available to insure a fast and smooth transition from job to job?
5. Is there a facility for executing several jobs serially without requiring stops in between jobs? Can the computing system be set up for future serial type jobs by the operator while the system is in operation?
6. Is there a facility for operating and executing several jobs in parallel including both I/O and computation? Can operator setup for parallel operation of several jobs be performed simply? How is continuous operation maintained?
7. Is the reentrant coding technique used within the programming system components (i.e. such that multiple requests will require only one program copy in primary storage)?
8. Can the decision be made dynamically as to which program segment will be executed next?
9. What provisions are available for execution of programs, segments or subroutines originally written for other installations?
10. What techniques are available for altering program parameters while maintaining continuous program-to-program transition?
11. Can decisions be made dynamically as to which debugging component to use as well as allowing modifications to debugging parameters?
12. Can computational tasks, subroutines or segments be executed in parallel if they belong to different jobs?
13. Can a single job execute tasks in parallel (including computation) within the software system?

14. What limitations prevail relevant to buffering input/output operations in parallel within a single job? Among several jobs?
15. Is there a provision for executive control of common subroutines?
16. What software modifications are required when additional central processing units (CPU) are added to the system? How many may be added with only minor modifications such as changing numbers in a table?
17. Are provisions available for allocation and control of temporary storage for each CPU? Up to how many?
18. Does the operating system monitor and control I/O requests by timed interrupts? For jobs? For programs? For tasks?
19. Is control and monitoring of I/O data transfers provided? What is required of the user-programmer?
20. Can the user request and obtain partial compilations of internal tasks dynamically? How flexible is this feature?
21. How much of the selection and manipulation of library program requests are performed automatically by the system? What is the programmer required to do?
22. Is there a provision in the operating system for scheduling the user's own library of programs?
23. What is the scheme for assigning priorities? Does the system control and monitor jobs and tasks for internal (programming system) and external (user) priorities? What is the difference between the limits of system and user priorities?
24. What facility is available for insuring that low priority programs get executed (e.g. periodic increases in priority for unexecuted jobs)?
25. Is rescheduling performed for high priority jobs? Automatically? Can a low priority job delay a high priority one? For how long?

26. What flexibility exists for system reorganization under future changes in the high priority scheme?
27. Does the operating system check for appropriate identification of programs? Of associated data and data segments?
28. Is there sufficient internal system identification of jobs, programs and tasks, and their priorities to insure proper completion of priority programs?
29. Are job, program and/or task queuing tables used?
30. Is the structure of the queuing tables sufficiently uncomplicated to allow efficient restart or path reconstruction after system failures?
31. Are these queuing tables maintained dynamically to continuously reflect the true system status?
32. Does the priority scheme provide sufficient flexibility and complexity for the user's needs? Describe.

I/O Allocation, Monitoring and Control

1. Are equipment delays for input/output processing controlled and initiated by the operating system. Describe the steps taken by the user.
2. Does the operating system provide continuous monitoring of interrupts on user's I/O?
3. Is the complexity of I/O handling considerably reduced (for the user) by the operating system?
4. Does the reduced complexity provide adequate flexibility in I/O device usage for the user?
5. Does the framework for I/O manipulation provide for future addition of standard I/O equipment?
6. Is the framework for I/O manipulation sufficiently flexible to allow for future changes in channel assignments?

7. Does the operating system framework provide for the possibility of future addition of special I/O equipment not initially required?
8. What provisions within the operating system are available for the future addition of external communication equipment?
9. Is the assignment of buffer areas for I/O transfers flexible enough to take advantage of future hardware changes on standard I/O equipment such as data transfer speed increases or increased device storage capacities?
10. Will the operating system internal structure allow for future increases of simultaneous I/O channels?
11. Will decreases in I/O channel simultaneity make the operating system overly cumbersome?
12. Is I/O symbolically assigned? If so, is the scheme adequate for the user's need?
13. Is the scheme for symbolic I/O assignments sufficiently flexible to allow for future additions of new and different types of I/O equipment?
14. Does the operating system provide for continuous monitoring of the entire computing system configuration? How?
15. Does this scheme allow controlling and updating of each I/O device or is it limited to channels?
16. Does the operating system provide for allocation and control of scratch tapes, temporary storage areas and input/output data areas?
17. How is the algorithm for scheduling equipment usage suitable for the user's needs? Can it be modified easily?
18. Is the internal structure adequate for effecting the efficient transition of program-to-program I/O?
19. Does the system provide flexibility in allocation and control of I/O channels?

20. Does the symbolic I/O assignment scheme allow allocation of the actual input/output device or is it limited to channel assignment?
21. Is continuous monitoring and control of remote console devices provided?
22. Does the operating system store program, data and/or message queues from remote console devices?
23. How complicated are the changes required for future remote console device additions? Up to what limits?

User and System Storage Allocation

1. Is there a technique for segmenting programs? Describe it briefly.
2. Does the operating system monitor and control the manipulation and activation of overlays?
3. What computing system storage devices in the proposed configuration are included in the segmentation scheme? Which ones are not?
4. What facilities are available for protecting the user's data segments and/or program segments? Describe these for both primary and secondary storage.
5. What portions of the relocatability technique are performed at assembly time? At program loading time? By the operating system?
6. Are there provisions for furnishing debugging output with the actual memory locations used by each instruction during execution?
7. What pre-execution steps must be taken to insure complete reconstruction of program paths during debugging operations?
8. Is the requesting and returning of storage areas performed easily by the user program?
9. What steps are needed for requesting and returning of storage areas?

10. Is the map of primary storage and table of user assignments readily available to the user-programmer?
11. Can existing tables and arrays of data be suitably re-located for high priority programs? How does each user program know where his data is during execution? During debugging?
12. What effects do priority programs have on program segmentation?
13. What internal communication switches or tables are effected by priority requests?
14. Does the operating system manipulate and control the transfer of job segments to and from secondary storage?
15. What flexibilities are afforded the user of the segmentation technique? Describe briefly.
16. Is the segmentation technique similar to and compatible with the technique used for library routine manipulation?
17. Does I/O assignment by the operating system include buffer area determination?
18. Are buffer areas reassigned when the number of programs increases? When the load gets lighter?
19. Can the user effect buffer area assignment if needed?
20. What effect does the changing of buffer area requirements have on the allocation scheme?
21. Is the allocation scheme flexible enough to take advantage of hardware configuration changes such as the addition or deletion of memory modules? More secondary storage?
22. Can the buffer area assignment scheme be easily modified to take advantage of increased I/O equipment? Decreased?
23. Does the operating system provide initial system status setup for jobs, programs and/or tasks?

24. Are specific I/O components initialized or put on ready status by the operating system (e.g. tape label checking, deleting identification records, selecting proper operating modes, etc.)?
25. Can a mix of foreground and background programs be executed? Describe the limitations briefly.
26. Can appropriate combinations of debugging operations be selected to handle predicted checkout needs?
27. Are the debugging operation commands adequate to provide requests for snapshots, partial dumps, segment dumps, traces and intermediate results?
28. Does the operating system monitor and allocate storage for snapshots, partial dumps, segment dumps, traces and intermediate results?

Library Manipulation and Maintenance

1. Are the library routines stored and retrieved the same way the compilers or assemblers are? Describe the differences briefly and what effects they have on the user.
2. What specific flexibilities are provided in the framework of the operating system which allow for easy access to library routines?
3. Is the structure sufficiently flexible to provide for calling library routines during compilation? Execution? Both?
4. What is the method used for linking user programs with library routines? With compilers or assemblers? Both?
5. What is required of the user programs to effect this linkage?
6. Are the library routines automatically read in and incorporated into the user's program? If not, what must the user do?

7. Are appropriate linkages set up by the system so that more than one request for the same library routine will produce only one copy in the requested program? If not, why not?
8. What library routines need separate data arrays or tables? For each, describe how these items must be provided. By whom?
9. Are there any requirements for control or monitoring on any of the library routines? Who performs the monitoring in each case? What effect does this have on the user program?
10. Is there a standard scheme for control transfer between the library routines and the operating system? Describe it briefly.
11. What requirements are placed on the user for effecting program segmentation of library routines?
12. Does the system provide linkage substitution in case library routines require program segmentation?
13. What provisions are included for allowing the maintenance and updating of library routines? Can user programs do it easily?
14. In case a new library routine fails in some way, what procedure must be performed to replace it with the old one or effect a suitable modification? Can this be done on-line? In either case, how is continuous system operation affected?
15. Are there any special features available for maintaining operational reliability of library routines?
16. Can library maintenance jobs, programs or tasks be submitted to the operating system as any normal user program? If so, what scheme or technique is used to insure a program obtaining the desired library routine version? If not, what special actions or special operating systems are needed? Why?
17. Does the operating system include the utility routines needed to both manipulate and maintain the system library? If not, what is needed?

Editing of User and System Programs

1. What listings are provided for post analysis by user programmers?
2. Are the messages produced by the operating system oriented toward distinct and well defined events so that accurate reconstruction may be performed?
3. Are the operating system output messages saved and combined so that the programmer may see what happened without having compilation, library routine or user program output interspersed?
4. Does the output message consist of a description? Is it complete within itself or is further description needed from an associated document?
5. Does the output message consist of a number which refers to a description in some associated document?
6. Is there an organized scheme within the software system concerning the production of the message descriptions so that each term has a distinct meaning?
7. Does the list of output messages include a facility for notifying the operator immediately when an on-line correction may be made instead of aborting a program?
8. What provisions are available for making machine language patches?
9. Does the output message listing contain sufficient information within itself to adequately reconstruct the program path? If not, what else is needed?
10. Does the operating system provide an effect code substitution suitable for all available debugging schemes?
11. Are provisions available for making compilations with or without debugging aids? What modifications are required by the user to delete debugging aids from a program?
12. Is a contiguous listing of machine errors provided?

13. Are there any provisions for monitoring and editing of out-of-range quantities during debugging?
14. Are arithmetic violations detected and recorded during program execution? During debugging?

Facility and User Time Accounting

1. Does the operating system perform monitoring of job timing for both user and facility?
2. Is a table maintained of timing information for each hardware component on the computing system? If not, is one maintained for any of the components? Which ones?
3. Is there a framework in the operating system which allows for generation and accumulation of statistics concerning system bookkeeping?
4. Can any of the following bookkeeping statistics be accumulated? Others? Describe each briefly.
 - a) Number of secondary storage transfers made.
 - b) Number of storage accesses for each secondary storage device.
 - c) Number of jobs, programs and/or tasks executed.
 - d) Frequency of usage of library and/or utility routines.
 - e) I/O channel time usage.
 - f) Timed usage of each I/O device.
5. Does execution of job abortive procedures include sufficient data to determine exactly what happened? Describe briefly.
6. Is there a means for continuously maintaining the status of each computing system component? Accessible by the user program?
7. Does the scheme for maintaining component status and timing recognize and reflect the sharing of components by jobs, programs or tasks?
8. Is an operations log maintained?

9. If so, are there sufficient items contained in it to fulfill management requirements such as contained in question 4. above?
10. Does the log contain sufficient information to facilitate predicting future estimates of configuration changes such as increases or decreases in numbers of I/O devices or channels?
11. Is a separate timed record provided for each major component of the component system? Describe each different method briefly.
12. If no individual component timing is performed, are there sufficient details maintained in the operating system for future addition of this capability?

Generating and Updating the Master System

1. What features are available so that individual utility programs may be modified or added easily?
2. Is the organization of the master system sufficiently flexible so that new master tapes may be made up with individual programs contained in a completely different sequence?
3. Does the system provide for easy updating and maintenance of compilers and assemblers by system programmers?
4. Are provisions available for maintaining and updating user's files or file systems?
5. Are directories of files or file systems maintained by the operating system?
6. Does the framework of the files or file system maintenance allow for easy access to these files during execution? During compilation? By other appropriate major software components?

Operator and Off-Line Communication

1. At the interface between the operating system and the off-line computer(s), are the commands organized and suitable for efficient data transfers?

2. Is similar terminology used in communication by both the on-line operating system and the off-line computing system(s)?
3. Does either system require unnecessary data conversions? Is there a standard data format?
4. Are the commands used between systems standard ones? Describe briefly.
5. Is there a provision for communicating with other on-line computers or other computer modules in this system?
6. Is the communication scheme used sufficiently open-ended to provide for future additions of computer modules, on-line computers or off-line computers?
7. Is there sufficient flexibility to allow for changes in the configuration of either computing system?
8. Has a master-slave relationship for control and communication been established between the on-line operating system and any subsystem monitors?
9. Does the operating system generate commands for the manual operation functions?
10. Are these commands communicated to a central console?
11. Can the operator request or provide answers to the operating system through a central console?
12. Does the operator have access to an updated record of I/O device assignments?

Error Recognition and Recovery

1. Does the operating system provide for continued operation when limited failures occur on components not currently being used by this job? To what extent can this occur?
2. Does the operating system automatically substitute a similar unused I/O device for one which has failed? Is system operation interrupted during this switch?

3. Are errors identified relative to the individual component or device that failed?
4. Are programming errors detected during program execution? If so, identify each as to one of the following types:
 - a) Incorrect instructions or formats.
 - b) Unstable iterative or nonconvergent processes.
 - c) Incorrect I/O requests.
 - d) Requests for equipment already in use.
 - e) Arithmetic violations or data overflows.
 - f) Incorrect data formats.
 - g) Software system violations.
 - h) Others?
5. Does the operating system have a capability of re-starting the system after an equipment failure has occurred, without completely rerunning the last batch of jobs? Describe briefly.
6. If the system provides periodic storage dumps as a method of easy recovery, what is the period? What happens when failure occurs during data transfer of this storage dump?
7. What amount of primary and secondary storage is required to utilize this dumping procedure?
8. What specific violations of hardware limits (such as magnetic tape running off the reel) are monitored by the operating system?
9. Is there a software implemented scheme for memory protection? Does it provide for both program and data protection separately? How many distinct areas of each type can be protected at one time?
10. Is there a parity checking technique used for error recognition? On what components?
11. Does the operating system execute tape redundancy calculations? Automatically or by user request?
12. Is one computing module used strictly for error detection? Is that program part of the operating system? Is communication provided by the operating system between this program and the operator?

13. Does the operating system maintain counts of error frequencies?

Documentation

1. Which of the following descriptive manuals are currently available:
 - a) Introduction to Operating System
 - b) Operating System User's Manual
 - c) Error Message Explanations and Recovery Procedures
 - d) Operating System Maintenance Manual
 - e) Internal Structure of Operating System
 - f) Programmer's Reference Manual for Operating System
 - g) System Training Manuals
 - h) Library and Utility Routines
 - i) Maintenance of Program and Routines Library
 - j) Others?

2. Which of the following computer output listings are currently produced by the operating systems:
 - a) Programming Errors Detected
 - b) Machine Errors Detected
 - c) Events Occurring Relative to Operating System Functions or Events
 - d) Names of Jobs, Programs or Tasks which were Compiled or Executed with Relevant Clock Times
 - e) Time Accounting for Hardware Components Relative to User Jobs
 - f) Manual Operator Instructions
 - g) Table of I/O Devices and What Jobs are Using Them
 - h) Others?

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) The MITRE Corporation Bedford, Massachusetts		2a. REPORT SECURITY CLASSIFICATION Unclassified	
		2b. GROUP	
3. REPORT TITLE A Method for the Evaluation of Software: Executive, Operating or Monitor Systems			
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) N/A			
5. AUTHOR(S) (First name, middle initial, last name) Budd, Arthur E.			
6. REPORT DATE September 1967		7a. TOTAL NO. OF PAGES 100	7b. NO. OF REFS 0
8a. CONTRACT OR GRANT NO. AF 19(628)-5165		9a. ORIGINATOR'S REPORT NUMBER(S) ESD-TR-66-113, Vol. 3	
b. PROJECT NO. 8510		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) MTR-197, Vol. 3	
c.			
d.			
10. DISTRIBUTION STATEMENT This document has been approved for public release and sale; its distribution is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY EDP Equipment Office, Electronic Systems Division, L. G. Hanscom Field, Bedford, Massachusetts	
13. ABSTRACT <p>This report contains features of executive, operating or monitor systems considered important for evaluation and comparative analysis. These features are identified in a form expressly for inclusion in the Three Step Method for Software Evaluation (Volume 1 of this series) under Category TWO: Executive, Operating or Monitor Systems. Included in this volume is a composite list of functions contained in current executive systems. These functions provide the basis for a standard approach to the software category of executive systems particularly needed for evaluation and comparative analysis.</p>			

14.

KEY WORDS

LINK A

LINK B

LINK C

ROLE

WT

ROLE

WT

ROLE

WT

Electronic Data Processing
Executive Systems
Monitor Systems
Operating Systems
Software Evaluation