AN EXECUTION BATCH MONITOR FOR PROCESSING STUDENT JOBS

By

SHARON JOYCE CLARKE

Bachelor of Science

Oklahoma State University

Stillwater, Oklahoma

1974

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1979

AN EXECUTION BATCH MONITOR FOR PROCESSING STUDENT JOBS

Thesis Approved:

[illegible]
Thesis Adviser

J P Chandler

[illegible]

Norman N Durham
Dean of Graduate College

1042920

PREFACE

This study deals with the investigation and implementation of a method of reducing operating system overhead for the short-running student jobs at Oklahoma State University. The method chosen is that of an execution batch monitor which eliminates much of the job overhead in processing these student jobs. Sound operating system principles and techniques are studied and incorporated into the monitor, as it assumes some of the operating system functions for the jobs which it processes.

I wish to express deep appreciation to my major adviser, Dr. Donald Fisher, for his guidance and assistance throughout my graduate study.

I would also like to thank Dr. John Chandler and Dr. Eugene Bailey for serving on my graduate committee and for their continuing support throughout my college career. Thanks also to Dr. George Hedrick for all his help and guidance.

A very special thanks goes to Dr. Verlin Drinen whose technical expertise made this study possible. I wish to thank all my friends in the Computing and Infomation Sciences Department and the University Computer Center for their friendship and encouragement. I am especially indebted to John Sherblom who pleasantly and patiently answered all my questions about SCRIPT, the text formatting program used to produce this thesis.

Special thanks go to the students enrolled in COMSC 2113 during the spring semester, 1979, for contributing to my test data, along with my sincere hope that they will be using my system as students in more advanced computer science classes in the future.

Finally I wish to express deep love and gratitude to my parents and family for their unfailing love and support of my endeavors.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The combination of faster and more powerful computer hardware and the increased diversity of computer applications have contributed to the trend towards large and complex general purpose operating systems. As the complexity of operating systems has grown, the overhead associated with accomplishing a given task on a computer has increased. This overhead, or work which the operating system must perform to control a task, can sometimes exceed the work which the task actually performs.

An example of this can be seen in a typical university computer center which utilizes a general purpose operating system to service both academic and administrative functions. A large number of the jobs processed by such a center are short-running student jobs which require minimal computer resources. The computer time necessary to initiate and terminate these student jobs is often greater than the time spent in actual processing.

Such a situation exists at the Oklahoma State University Computer Center which utilizes the IBM OS/VS2 MVS operating system on an IBM 370/168 computer to process both academic and administrative jobs. Almost half of the jobs processed at this center invoke one of the four available fast compilers developed for student use. Many of these student jobs spend less central processor time in actual execution than they do in initiation and termination.

The purpose of this study is to investigate and implement a method of reducing the system overhead for this special class of jobs at Oklahoma State University. This method involves the use of a monitor which assumes some of the tasks which the operating system normally performs. For this reason it is necessary to study the concepts and selected techniques of operating systems.

Chapter II of this study presents a brief overview of operating system evolution. The characteristics of batch monitor systems are presented in greater detail.

Chapter III discusses the various operating system techniques which have applicability to batch monitors.

Chapter IV discusses the implementation of the monitor system at Oklahoma State University. Approaches taken to avoid the historic pitfalls of monitor systems are also presented.

Chapter V discusses the methods used in testing to determine the amount of system overhead saved by the monitor system. Future work and possible improvements are also presented.

The appendices include a user's guide, a programmer's guide, and an error message manual for the monitor implementation. A logic description of the monitor is given in the programmer's guide. Also included is a list of common acronyms used in these documents.

CHAPTER II

OPERATING SYSTEM OVERVIEW

Definitions of the term "operating system" are as plentiful as the number of people who write about them (1,12,15,16). The underlying theme of these definitions is that an operating system is a set of programs which manages computer resources for the user of the machine. Madnick and Donovan (15) classify these computer resources as memory, processors, I/O devices, and information to include programs and data. They further define the tasks of the operating system in managing these resources as keeping track of the resource; enforcing policy that determines which job gets what, when, and how much of each resource; allocating the resource; and reclaiming the resource.

In order to understand the underlying concepts, a brief overview of operating system evolution is in order. The material in the next section is based on the historical work of Robert F. Rosin (16). The approximate dates of each development are those of Madnick and Donovan (15).

Operating System Evolution

Before any type of operating system existed, programmers wrote in machine code and operated the computer personally. Batch monitor systems, first developed around 1956, allowed users to "batch" their jobs together, eliminating the need for each programmer to load his or her program manually. The bottleneck which still existed was that of the relatively slow speeds of the input and output processors compared to the faster speed of the central processor.

The advent of the data channel and I/O interrupt processing ushered in the first executive systems in the late 1950's. These systems permanently resided in memory and provided the user with input and output management routines. Although some overlap of I/O and computational processing was provided, the real advantages of interrupt processing were not realized until the development of multiprogramming operating systems in the early 1960's.

Multiprogramming is the ability to have more than one program in main storage at the same time. The processes of the various programs are interleaved rather than simultaneously executed as is done in a multiprocessing environment. This implies that one process may use the central processor while one or more other processes wait for the completion of an I/O operation.

Multiprogramming greatly enhanced system throughput and led to the increased use of computers for a wider range of applications. These included sophisticated data processing and interactive systems. Data management routines and timesharing techniques were incorporated into operating systems in the middle 1960's to handle these tasks.

This brief sketch of operating system development illustrates the improvements which have been made since the early days of computing. At the same time it illustrates the inherent complexity of modern operating systems. Systems have become so elaborate that the amount of computer time used in actual processing of a job can be exceeded by the amount of time required by the system to control that job.

Rosin (16) concludes that more work must be done in the area of providing systems which result in as little overhead as possible for jobs which do not require the full resources of a system, while providing extended functions for those jobs which require them. The incorporation of the batch monitor concept into modern general purpose operating systems provides a solution to this problem.

Batch Monitor Characteristics

As discussed, the concept of batch monitors is a primitive one. They were used in the early days of computing to run whole sequences of jobs without human intervention in order to save the time lost waiting for an operator to respond to a request or initiate a new job.

Colin (2) presents the basic form of a batch monitor as shown in Figure 1.

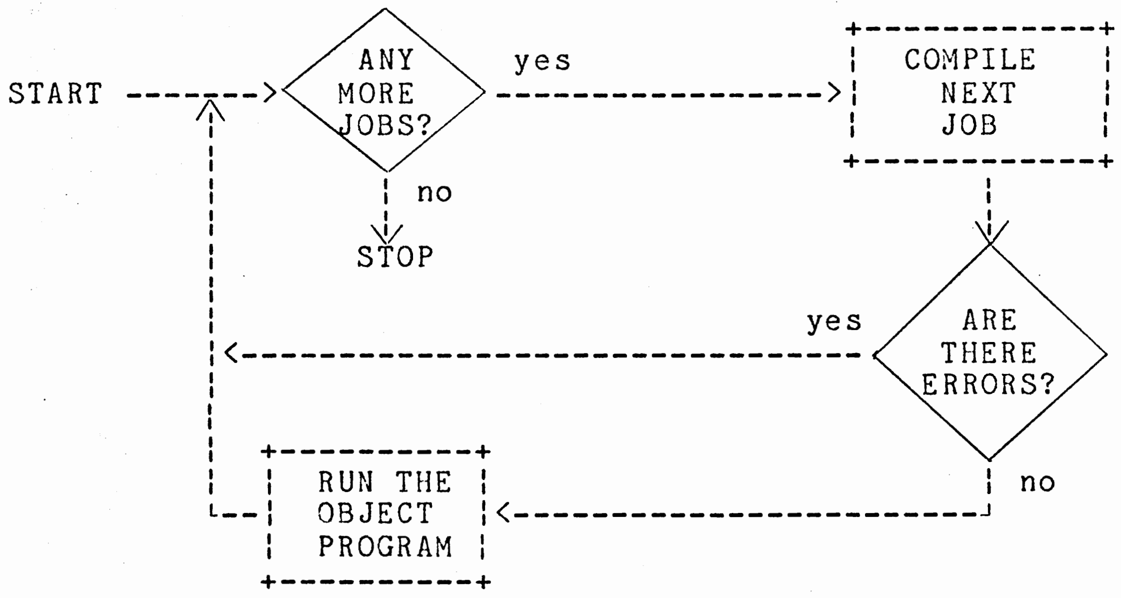

Figure 1. Basic Form of a Batch Monitor

It simply automates the job cycle and maintains control over the compiler and the object program produced by the compiler by calling them as subroutines. Each job is preceded by some type of job description and terminated by a special record to enable the monitor to distinguish individual jobs.

This simple system has several drawbacks which limit its use. The compilers available to a job are only those which have been written into the system as subroutines. The addition or modification of a compiler implies that the monitor itself must be modified. Another problem is that an error in the compiler or object program can cause the entire system to terminate or loop indefinitely. Since no protection of the area of memory in which the monitor resides is provided, the object program might also replace part of the monitor causing unpredictable results for other jobs in the stream.

In spite of these drawbacks the batch monitor is used in this simple form, particularly at universities where the job load consists of many short simple jobs.

A Batch Monitor Superimposed on the Operating System

Another type of batch monitor, sometimes called a "pseudo" batch monitor, is one which is superimposed on another more general operating system (2). In IBM's MVS operating system this facility is called an execution batch monitor and is actually provided by the Job Entry Subsystem (JES) which is primarily responsible for the input and output of jobs. JES controls the monitor which appears to the operating system as a single job. In actuality multiple jobs of a pre-specified "batch" class are passed to the monitor by JES. The monitor is then responsible for processing each of these jobs. The monitor itself is not provided by JES as it must be tailored to the unique needs of the particular installation.

The possibility of reducing system overhead for the jobs processed by such a monitor is substantial because the operating system itself does not control the jobs which are processed by the monitor. The monitor, on the other hand, introduces some overhead because it must provide some of the functions of an operating system to the jobs which it is processing. The amount of overhead introduced depends on the needs of the installation and how efficiently the monitor implements the operating system functions which it assumes.

For this reason it is necessary to investigate selected operating system techniques and to explore methods of eliminating or reducing the problems associated with early batch monitors. These areas are presented in the following chapters.

CHAPTER III

OPERATING SYSTEM TECHNIQUES

Much of the theoretical work currently being done in operating systems is directed toward the complex interactions found in multiprogramming systems. Although a batch monitor is by definition a serial processor, its design utilizes basic operating system principles. Among these are the selection of appropriate design objectives, I/O buffering techniques, job accounting considerations, and general operating system characteristics.

Selection of Design Objectives and Methodologies

Brinch Hansen (1, p. 18) states that "the key to success in programming is to have a realistic, clearly-defined goal and use the simplest possible methods to achieve it." This is especially true for operating systems and yet can be most difficult to achieve.

The objective of many general purpose operating systems is to provide a large variety of services in accommodating an environment of diverse applications on a range of hardware configurations. Systems constructed on this premise normally use a horizontal or functional approach. In this type of design the system is a collection of tools, techniques, and functions which can be assembled in a variety of ways to meet the requirements of a particular user. MVS is a good example of this type of construction. Although it is commercially successful, the combination of such a broad objective and the functional design method chosen are directly responsible for its lack of efficient performance (14).

In contrast a special purpose operating system can take advantage of such critical components as available resources and predictable workloads to optimize performance. Lynch (14, p. 582) proposes that "the success of the design process is strongly influenced by how well the load for the system can be characterized." He cites airline reservation systems as examples of systems with statistically predictable workloads. Brinch Hansen (1) suggests that the success of the EXECII spooling system is due to the designers' knowledge of the expected workload and of the characteristics of the I/O device used for spooling.

Brinch Hansen also suggests that productive sharing of a large installation requires a range of operating systems, each providing a particular service in the most efficient and simplest manner. The designers of the FAMOS system (6) disagree, citing the development cost of independent systems as being prohibitive. The major goal of the FAMOS system is to show the feasibility of a system family based on the design methodology introduced by Dijkstra (4) in the T.H.E. system. This design methodology, referred to as vertical design, defines the system as a hierarchy of levels of abstraction. Each level is in effect a virtual machine to be used by the higher levels. It is the contention of the FAMOS designers that an efficient general purpose system family can be constructed using this methodology.

Regardless of the objectives or methodology employed, the importance of clearly defining the objectives and selecting an appropriate methodology is emphasized by the designers of each of the systems discussed.

I/O Buffering Techniques

The concept of data transfer is a basic one as this function can be found in the simplest of operating systems. Buffering is a technique in which a block of data is input into or output from an area of memory called a buffer so that the central processor can access it. Once the input or output operation is begun, the actual data movement is performed by the data channel. This leaves the central processor free to perform other processing as long as the buffer area is not accessed until the I/O operation is complete. This capability of overlapping computational time and I/O time has led to the development of algorithms for buffer management which attempt to utilize this overlap.

Two classes of algorithms exist: synchronous and asynchronous. A synchronous technique is one which requires that the program periodically check for the completion of the I/O operation. The more commonly used asynchronous technique is one which depends on an interrupt from the I/O device to signal completion.

Knuth (11) describes some synchronous algorithms, although the concepts presented can be readily adapted to an asynchronous environment. The simplest technique which Knuth describes is that of buffer swapping. In this technique two buffers are used so that while one is involved in an input or output operation, the other may be accessed by the program. The algorithm includes a simple method of detecting which buffer may be accessed.

A more general algorithm involving any number of buffers arranged in a circularly linked list is also described. Knuth depicts a buffer as being in one of the following three states:

1. The buffer is ready to be assigned; that is, it is filled with information in the case of input or is a free area in the case of output.

2. The buffer is the current one with which the program is communicating.

3. The buffer is released; that is, it is a free area in the case of input or it is filled with information in an output situation.

In the algorithm pointers are kept to the next buffer in each of the three states and proceed around the circularly linked list of buffers in an orderly fashion. Checks must be made to ensure that the pointers do not pass each other.

Another important consideration in buffer management is the size of the buffer. The time which it takes to move a block of data between auxiliary and main storage can be broken into two parts: access and flow. Access consists of some form of mechanical positioning such as disk arm movement and rotational delay or tape start time. A single access is required regardless of the amount of data being transferred. The time for flow, or actual transmittal of the data, on the other hand, is directly proportional to the amount of data transmitted. This implies that the total time to transmit a given amount of information can be reduced if each access is to as much data as possible. The trade-off is obviously in the amount of main storage required for the buffer.

The concepts presented here are used by Hellerman and Smith (7) in an analysis of the effect on performance of some idealized overlap configurations. Their results indicate that some overlap of computation and I/O operation is usually better. However, they do point out that in an I/O bound system with a single channel a nonoverlapped operation with a larger buffer is better.

Accounting for System Resource Usage

One of the important aspects of an operating system is keeping records of system resource usage in order to charge the users of the system equitably. Sayers (3) describes the information typically recorded by a system for each job as job identification and termination status, number of records added to or deleted from each permanent file, central processor time utilized, time used by each channel and each device, number of lines printed, number of cards punched, and number of records written on system output units. In most systems the actual routines needed to analyze and subsequently charge the user account are left for the installation to code.

IBM's OS/VS2 MVS operating system provides this function through a facility called System Management Facilities or SMF (9). SMF records a variety of both system-wide and job related information. It provides exits that allow installations to add routines to the system to perform additional processing or create their own records. SMF records are written on special system data sets which may subsequently be read by installation routines to perform the actual job billing.

Operating System Characteristics

Kurzban (12) presents the following list of desirable qualities of an operating system.

1. Usability - The system's interfaces are designed with its users' convenience in mind.

2. Generality - The system does exactly those things which its users want it to do -- no more, no fewer.

3. Efficiency - The system makes optimum use of the resources at its disposal.

4. Visibility - The system permits its users to learn those things about itself which might be of value to them.

5. Flexibility - The system can be modified (tuned) in response to the behavior of its users.

6. Opacity - The system permits its users to remain ignorant of those things which are beneath the interface it provides.

7. Security - The system protects those things which its users entrust to it.

8. Integrity - The system protects itself from damage which might be caused by its users' errors or malice. Conversely, its users can be sure that the errors they see are their own and not the system's or anyone else's.

9. Capacity - The system presents as large an interface as possible within its physical constraints.

10. Reliability - The system fails as rarely as possible with as little impact upon its users as possible.

11. Availability - The system continues to function in the presence of as many errors as possible, albeit with restricted capability or efficiency.

12. Serviceability - The system does as much as possible to facilitate and expedite repair.

13. Extensibility - The system facilitates the addition to it of functions which its users might desire.

Many of these qualities have application to the batch monitor described in the next chapter.

CHAPTER IV

BATCH MONITOR IMPLEMENTATION

The execution batch monitor facility of JES provides a method of reducing system overhead for the numerous student jobs processed at the Oklahoma State University Computer Center. As described in Chapter II, this facility passes jobs of a pre-specified class to a user-written monitor for processing. The monitor appears to the MVS operating system as a single job. Thus dataset allocation and job initiation and termination are performed one time instead of once for each of the student jobs processed by the monitor.

The monitor is responsible for processing each of the jobs passed to it in an efficient and reliable manner. How this is accomplished is discussed in the remainder of this chapter.

Design Objectives and Methodology

As stated, the primary objective of the monitor is to reduce system overhead. An important secondary objective is to provide the functions required by the student jobs in such a way as to make the change in processing invisible to the user. At Oklahoma State University a "student" job is classified as a job which:

1. Uses ten seconds or less of central processor time,

2. Does not require disk or tape mounts,

3. Requires no additional DD cards, and

4. Executes one of the fast student compilers, WATFIV, PLC, ASSIST, or WATBOL, or the OSU-written programs LIST or ROUTE.

In the text which follows, a "processor" refers to one of these six programs elgible for student job processing.

A top-down structured methodology is used in the design of the monitor. This methodology is more applicable than either the functional or vertical approaches described earlier because of the sequential nature of the monitor. The processes which it must perform for each job can be easily broken down into functions which can be coded as subroutines. Each function or subroutine can then be coded and tested separately before inclusion in the monitor. The logic descriptions given in the Programmer's Guide in Appendix B enumerate these functions.

Buffering Techniques

Each job processed by the monitor includes an input stream. If the job invokes one of the student compilers, this stream consists of the program to be compiled and any associated data. The monitor must create a file from this stream for use by the processor.

The Basic Access Method (BSAM), which requires that the program perform buffering, is used to create the file. A simple one-buffer technique is employed for two reasons. First of all it is possible to predict the average size of the file. By selecting a buffer size large enough to contain all the input for one job, only one output operation is needed. Second, it is important to realize that although the monitor is a serial processor, it is operating in a multiprogramming environment. Any time which it spends waiting for the completion of an I/O operation is used by other jobs in the system. This implies that the expense in terms of programming effort and additional overhead outweighs any gains in implementing a more sophisticated buffering algorithm.

The monitor does achieve a degree of overlap between computational and output operations. When it detects the end of a job it immediately begins the output operation on the buffer it is building. It then proceeds to perform other tasks for the job such as final authorization checking. The wait for the completion of the output operation is done just before invocation of the required processor.

Resource Accounting

Another important function of the monitor is that of resource accounting. The resources available to each job running under the monitor are limited to ten seconds of central processor time, access to certain system datasets pre-allocated to the monitor (for example, WATFIV subroutine libraries), and the capability of producing an output file of up to 1500 lines. The monitor is responsible for ensuring that these limitations are not exceeded and charging the user for resources expended.

In order to accomplish this, the monitor utilizes the control block structure of MVS. Because the monitor is a single job to MVS, resource statistics are maintained for it as they are for any other job in the system that is not running under the monitor. These statistics are kept in control blocks available to the monitor and are checked before and after a batch job is processed. Details of the control blocks used are described in the Programmer's Guide in Appendix B.

The problem which remains is that of ensuring that a batch job does not use more resources than it is allowed, that is, more than ten seconds of central processor time or more than 1500 lines of output. In order to explain how this problem is resolved, definitions of the terms "task" and "subtask" are required.

A task is a program which resides in storage and has been scheduled to use the central processor. It has the ability to create another task, known as a subtask, which competes for system resources in the same manner as any other task in the system.

For reasons to be discussed later, the actual processing of the student job by the compiler takes place as an independent subtask of the monitor. This means that the timer facility of MVS cannot be used simply to interrupt processing of the subtask after it has used ten seconds. Instead the monitor must periodically check the time used by the subtask. It accomplishes this by activating itself at given intervals of time which decrease to a set minimum as the time used by the subtask nears ten seconds or the number of lines output approaches its maximum. This algorithm can by no means produce an exact cutoff, but it works quite well if careful attention is given to selection of the activation intervals.

The monitor not only collects resource usage data, but it also charges the user for the job. The method for charging the user is essentially the same as that used for other jobs in the system. A five-digit project number is assigned to each user or project and must be included in the accounting information on the JOB card of each job submitted. The current balance for each project number is kept in a permanent file and is updated by the monitor after a job is processed. The monitor optionally prints the accounting information for the user. In addition the monitor outputs an SMF record detailing the resource usage for each job. SMF is the system-wide collector of resource usage data and is described in Chapter III. These records are read by a daily accounting job which ensures that each project number is charged correctly for its usage.

The monitor contains an accounting feature which is not available for other jobs. At Oklahoma State University it is often the case that a course is assigned one project number for all the students enrolled in that course to use. This is necessary because of the volume of paperwork involved if each of 800 students, say, were assigned an individual number. The professor often wishes to be able to maintain better control over the funds available to a particular student. In order to do this a user exit is provided which allows the professor (or anyone who can be persuaded) to code his or her own program to perform whatever functions are desired. The exit has the capability of denying access to any job running under its project number.

Batch Monitor Limitations

As discussed in Chapter II, monitor systems have three major drawbacks. In this implementation two of these are eliminated and the third reduced in severity.

The limitation which remains is the problem of adding functions. The monitor is designed to invoke one of six pre-defined processors, all of which have unique parameter and calling requirements which must be coded in the monitor. Although it is true that the monitor must be modified and re-assembled to provide additional processors, this modification is limited to four of the seventeen modules which comprise the monitor, and consists mainly of adding appropriate entries to existing tables. Instructions for doing this are well-documented in the program, which further adds to the ease of modification.

The two remaining drawbacks of monitor systems are that errors in the user program can cause the system to terminate or loop indefinitely and that the system itself may be overlain by a user program because of the lack of protection. These drawbacks are eliminated by attaching the required processor as an independent subtask instead of calling it as a subroutine. The problem of the indefinite loop is nullified by the previously described method of periodically checking the central processor time used by the subtask.

The ability to create a subtask is provided by the MVS operating system and does include drawbacks of its own. More overhead is introduced than would be if the processor were called as a subroutine. Tne inability to time the subtask directly has already been mentioned. These drawbacks are minor compared to the advantages which are provided. The abnormal termination of a subtask does not terminate the creating task, but rather simply returns control to it with an appropriate condition code. Memory protection is provided for both tasks, in that neither one can overlay part of the other. Communication between tasks is also provided so that the monitor can regain control when the processor is completed. The monitor also has the ability to abort the subtask if it exceeds the resources allotted to it. Thus the advantages gained by attaching the processor greatly enhance the reliability of the monitor.

CHAPTER V

MONITOR EVALUATION AND ENHANCEMENTS

In order to evaluate the performance of the batch monitor, a discussion of the testing methods and results is in order.

Testing Methods and Results

In order to test the resource savings of the batch monitor system, a job stream consisting of 101 jobs was run on a stand-alone system first under the monitor and then under the control of MVS. The jobs were run on the IBM 370/158 in use at the university at the time of the tests. Approximately half of these jobs were collected from students taking a beginning course in programming using WATFIV at Oklahoma State University. The remaining jobs were generated by the author and other members of the University Computer Center Systems staff in order to test certain aspects of the system. The jobs used anywhere from less than a second to the full ten seconds of central processor time allotted and were representative of the job mix normally run at the installation. The Resource Management Facility (RMF) of MVS was used to collect the statistics shown in Tables I and II.

TABLE I

CENTRAL PROCESSOR UTILIZATION

	Batch Monitor	MVS OS/VS2
Elapsed Real Time In Seconds	337.357	606.746
CPU Utilization In Seconds	162.442	306.714
Average CPU Utilization Per Job in Seconds	1.608	3.037
% Cpu Idle	51.84	49.44

As Table I shows, the batch monitor used 47% less central processor time than the conventional operating system. This savings can be directly attributed to the reduced overhead of the monitor system. It is interesting to note that the monitor has little effect on the percentage of time which the central processor spent idle, or waiting for work to do.

Table II depicts the count of device accesses for each of four disk devices in use at the installation. The overall decrease of 46% occurs for several reasons. The savings in activity on DASD50 are due to the absence of scratch dataset creation under the batch monitor. The monitor user VIO (Virtual I/O) for its scratch space if needed. Under MVS one or more scratch datasets are allocated for each job whether they are used or not.

TABLE II

DEVICE ACCESS COUNT

Volume Serial	Batch Monitor	MVS OS/VS2
DASD00	3533	6068
DASD10	4071	8801
SYSTSO	3268	2128
DASD50	0	3279
	-----	-----
TOTAL	10872	20276

The device activity on DASD00 is primarily due to accesses to the processor load modules. Each retrieval of a processor by the MVS operating system actually requires two accesses because the processors are stored as members of a partitioned library. One access is needed to the directory of the library to obtain the location of the member and a second access is required to retrieve the processor load module. The batch monitor introduces a savings of nearly 50% because it performs one access to the partitioned library directory during its initialization process and saves the location of each processor which it needs. It then must perform only one access per job to retrieve the applicable load module.

A reduction in accesses to the accounting file explains the savings found on DASD10. The monitor performs one read and one write to this file for each job. Under MVS two reads and one write are required because the authorization and updating are performed by independent modules.

The increase in activity on SYSTSO under the batch monitor is due to the need for the intermediate file for input to the attached processor. Overall, however, the test results indicate a significant savings in central processor utilization and device activity.

Monitor Evaluation

It is now useful to evaluate the performance of the batch monitor using Kurzban's thirteen qualities of a good operating system given in Chapter III.

Usability

The system is designed with the users' interface convenience in mind as the job control language remains unchanged from that used before its implementation. Although it is possible to have used fewer than the three control cards necessary to run a job, this implementation requires that the user learn only one set of control cards to run a job in any class.

Generality

According to Kurzban, generality in an operating system is the quality of doing exactly those things which the users want it to do. This is rather hard to judge because the desires of the users are not well-defined. The monitor does in fact perform exactly those things which are required of it to process the special class of jobs known as student jobs at Oklahoma State University, and in this respect it fulfils the requirement.

Efficiency

Efficiency is a quality which evades unbiased evaluation. The test results show that the monitor processes its jobs with greater efficiency than the MVS operating system, but according to Lynch (14) that is a rather weak claim. Efficient methods are utilized in the monitor as presented in this study, but improvements are nearly always possible and are discussed in a later section of this chapter.

Visibility

The quality of allowing users of the system to learn those things about the system which might be of value is not present in the batch monitor. This is not a serious drawback because the majority of users are students who have limited interest in how their jobs are processed.

Flexibility

The area of tuning the monitor in response to user behavior is one of the areas which needs improvement. This is discussed in a later section.

Opacity

Opacity, which is allowing users to remain ignorant of the system's internal processing, is probably one of the best features of the monitor system. The monitor has been operating in a production environment at Oklahoma State University for over five months and students who have not been informed of the change are totally unaware that their jobs are being processed differently.

Security and Integrity

These qualities are presented together as they are actually provided by the MVS operating system as discussed in Chapter IV. The monitor itself possesses these qualities by the way in which it makes use of the facilities of MVS available to it.

Capacity

Presenting as large an interface as possible within physical constraints is another quality which has little impact on the evaluation of the batch monitor. The monitor is designed to be a special purpose system with restricted capabilities and so presents a limited capacity to its users.

Reliability and Availability

Reliability is another strong feature of the batch monitor. After an initial period of two weeks in which two errors were found and corrected, the system has not failed in a five month period of use. This contributes strongly to its availability, although the monitor does not have the ability to function in the presence of errors. It does report certain errors to the operators of the machine so that they can remedy the situation and restart the monitor in many cases.

Serviceability

The quality of expediting repair has not been adequately tested since only two errors have been discovered. The top-down methodology used in the design of the monitor should contribute to isolating problems when they occur.

Extensibility

The ease of adding processors to the monitor system has been discussed earlier. The addition of other functions, for example the ability to allocate datasets dynamically, could be accomplished but probably not as easily. Here again the top-down methodology lends itself to the addition of functions.

Improvements and Future Work

Overall the test results and evaluation indicate that the batch monitor provides an efficient method of processing student jobs. Improvements, however, are an important part of any functioning system.

One way to improve the monitor is to add the capability to collect statistics about itself. Useful information could include the number of jobs run, the amount of resources used, the number of times each processor was invoked, the number of records written to the output file, and so forth. This information could then be used to tune the system. For example, if it were discovered that a large percentage of the jobs write only fifty records to the output file which the monitor is responsible for buffering, the size of the buffer could be reduced resulting in a significant storage savings.

The problem which exists in collecting data about the jobs it is processing is that the monitor never terminates normally. It is actually an infinite loop, continually processing the jobs sent to it and waiting in an inactive state when there are no jobs to process. The monitor is abnormally terminated by JES when its initiator is drained. This implies that a simple collection of data and some type of report or record written just before termination is not possible. Writing a record to a file for each job seems excessive, particularly since the load on the system fluctuates according to the due date of the next assignment and careful attention would have to be taken to ensure that the file would not overflow. A better alternative would be to collect statistics and generate a record for a given interval of real time, such as every half hour. These statistics could then be printed as part of a daily job and used to tune the system.

Another area in which the monitor could be improved is in storage utilization. Currently the monitor requests the maximum amount of storage required by any of its six processors at initialization time and performs no dynamic storage requests. Because the processors require varying amounts of storage, the monitor could make more efficient use of storage by dynamically requesting the amount needed depending on the processor invoked. This method introduces more overhead, so a careful analysis of the benefits versus the disadvantages is required. The aforementioned enhancement of collecting statistics would greatly benefit this analysis.

A third possibility for improvement is for the monitor to decipher and print the return code from the attached processor for the user's information. This was not done in the original version because a non-zero return code from WATFIV, the processor used most often, is usually meaningless. Other processors, in particular WATBOL, have useful return codes and thus give this enhancement some merit.

Summary

This study deals with the investigation and implementation of a method of reducing system overhead for the short-running student jobs processed at Oklahoma State University. This study is important because of the large percentage of overhead introduced by the general purpose operating system for this special class of jobs.

The method chosen was that of the execution batch monitor facility of JES. Sound operating system principles and techniques were studied and incorporated into the monitor. Testing showed that an overall savings of nearly 50% was introduced by the use of the monitor over conventional job processing.

REFERENCES

(1) Brinch Hansen, P. _Operating System Principles_. Englewood Cliffs, N.J.: Prentice-Hall, 1973.

(2) Colin, A. J. T. _Introduction to Operating Systems_. London, Great Britain: Macdonald & Co., 1971.

(3) Comtre Corporation. _Operating Systems Survey_. Ed. Anthony P. Sayers. New York: Auerbach, 1971.

(4) Dijkstra, Edsger W. "The Structure of the T.H.E. Multiprogramming System." _Communications of the ACM_, 11, 5 (May, 1968), 341-346.

(5) Gibson, O. L. "AUTOBATCHER: The Virginia Tech Student Job Processor." _Proceedings of Share XLIII_, Vol. 2, (August, 1974), 750-754.

(6) Habermann, A. N., P. Feiler, L. Flon, L. Guarino, L. Cooprider, B. Schwanke. _Modularization and Hierarchy in a Family of Operating System_ Pittsburgh, Pennsylvania: Department of Computer Science, Carnegie-Mellon University, 1978.

(7) Hellerman, H., H. J. Smith, Jr. "Throughput Analysis of Some Idealized Input, Output, and Compute Overlap Configurations." _Computing Surveys_, 2, 2 (June, 1970), 111-117.

(8) IBM, _OS/VS2 MVS System Programming Library_: _JES2_. Poughkeepsie, New York: IBM, 1979.

(9) IBM, _OS/VS2 MVS System Programming Library_: _System Management Facilities_. Poughkeepsie, New York: IBM, 1977.

(10) Keedy, J. L., "On Structuring Operating Systems With Monitors." _Operating Systems Review_, 13, 1 (January, 1979), 5-9.

(11) Knuth, Donald E. _The Art of Computer Programming_. Vol. 1. 2nd Ed. Reading, Mass.: Addison-Wesley, 1973.

(12) Kurzban, Stanley A., Thomas S. Heines, Anthony P. Sayers. Operating Systems Principles. New York: Petrocelli/Charter, 1975.

(13) Liskov, Barbara H. "The Design of the Venus Operating System." Communications of the ACM, 15, 3 (March, 1972), 144-149.

(14) Lynch, W. C. "Operating System Performance." Communications of the ACM, 15, 7 (July, 1972), 579-583.

(15) Madnick, Stuart E. and John J. Donovan. Operating Systems. New York: McGraw-Hill, 1973.

(16) Rosin, Robert F. "Supervisory and Monitor Systems." Computing Surveys, 1, 1 (March 1969), 37-53.

(17) Warwick, Martin. "Introduction to Operating System Concepts." Executive Programs and Operating Systems. Ed. G. Cuttle and P. B. Robinson. New York: American Elsevier, 1970, 1-10.

APPENDIX A

SUPERMON USER'S GUIDE

General Description

SUPERMON is an execution batch monitor program responsible for the execution of Class Z jobs. Execution batching is a facility of JES whereby quick-running jobs using similar resources are "batched" together as a single job thereby eliminating certain job management overhead.

At OSU Class Z jobs fall into this category because they are jobs which:

1. Use ten seconds or less of CPU time,
2. Do not require disk or tape mounts,
3. Require no additional DD cards, and
4. Execute one of the fast "student" compilers WATFIV, PLC, ASSIST, or WATBOL, or the OSU-written programs LIST or ROUTE.

Very simply, execution batching works in the following way at OSU. When a job is read into the system with 'CLASS=Z' on the JOB card, JES passes the entire job to the execution batch monitor, SUPERMON. SUPERMON is then responsible for reading the job, performing validity checking, invoking the processor (WATFIV, PLC, etc.) which the job requires, and performing OSU accounting for the job.

SUPERMON processing is essentially transparent to the user and for this reason the details of processing are not given here. Users who wish more detailed information may refer to the Programmer's Guide in this document.

Job Submission

Job submission for Class Z is essentially the same as other classes at OSU with a few minor differences. These are:

1. MSGCLASS=Z is the default message class for Class Z jobs. This job card parameter suppresses the listing of input JCL and the OSU accounting box. Users who wish to have these listed must specify MSGCLASS=A on the first (or only) JOB card. MSGCLASS should not be specified on a JOB continuation card for Class Z.

2. Allocation messages, step condition codes, and EXCP counts are not listed for Class Z jobs.

3. Error messages are printed for Class Z users if the password check fails, the account is out of funds, or a JCL error occurs, regardless of the value of MSGCLASS.

4. The TIME parameter on the JOB card has no meaning, but a 10 second time limit is in effect. Users who wish to lower this time limit may do so by using the $JOB card for WATFIV, WATBOL, and ASSIST, and the *PL/C card for PL/C.

5. The MSGLEVEL parameter on the JOB card does not affect the JCL listing because the class Z job does not execute a procedure.

6. The number of cards input is limited to 1500.

7. The number of lines output is limited to 1500.

8. A '//jobname JOB' card or a null JCL card ('// ') cannot be listed using LIST under Class Z. A JOB card, even though part of a DD DATA input stream, is still recognized as a JOB card and unpredictable results will occur. A null JCL card terminates the listing.

Other OSU standards for the JOB card and /*PASSWORD card remain in effect. A single '// EXEC ' card must be included and specify one of the processors WATFIV, ASSIST, PLC, WATBOL, LIST, or ROUTE. The only DD card which may be included is a '//SYSIN DD' card which must immediately precede the input.

SUPERMON prints the following user messages under certain conditions.

OSU001I	OSU002I	OSU006I	OSU007I
OSU008I	OSU009I	OSU011I	OSU014I
OSU015I	OSU016I	OSU019I	OSU032I
OSU101I	OSU102I	OSU103I	OSU104I
OSU105I	OSU106I	OSU115I	

These messages are listed in detail elsewhere in this document.

User Exit Description

Purpose

A user exit facility has been implemented which provides the project director of an OSU account more control over Class Z usage. This facility consists of a program, written by a user with the project director's authorization, which performs additional validity checking of the JOB card used to submit a job under that OSU project number. For example, the user exit could contain a list of social security numbers which would be valid and refuse access to a job which did not have a valid social security number. The criteria for validation is solely at the discretion of the project director as long as OSU standards are first met.

User Exit Requirements

The user exit must accept two parameters, an 80 byte character string containing the job card and a full-word (length 4) binary return code. This latter parameter is used to indicate to SUPERMON how the user exit wishes the job in question to be handled. A code of zero indicates that the job is to be processed. Any other value indicates that the job is not to be processed. In this latter case SUPERMON issues a message to the user indicating that the job has been cancelled by a user exit.

The user exit may be written in any language although Assembler is recommended for performance reasons. The user should be aware that resources used by the user exit (CPU time, disk accesses, and so forth) will be charged accordingly.

Anyone wishing to implement a user exit should check with the Systems Group no later than the completion of the early design phase of such a project. File requirements must be approved, and ample time must be provided for testing before a user exit will be permitted to be placed in production.

APPENDIX B

SUPERMON PROGRAMMER'S GUIDE

System Description

Abstract

SUPERMON is an execution batch program responsible for processing Class Z jobs at Oklahoma State University. Execution batching is a JES facility which allows certain job management overhead to be eliminated. A description of execution batch processing is contained in *OS/VS2 MVS System Programming Library: JES2* in the section "JES2 Processing".

Programs

SUPERMON is actually one Assembler program incorporating several subprograms. Because of its large size, each subprogram is treated as an individual program for purposes of this documentation. The source for each of the subprograms listed below may be found in the library 'SYS1.SUPERMON.SOURCE'.

1. PARSPRM - Parses the parameter which is passed to SUPERMON on its EXEC card.
2. INITBTCH - Initializes pointers which remain constant throughout the life of a SUPERMON batch run.

3. INITSDB - Initializes pointers to the SDBs for all of the SYSOUT files which the attached processors use.

4. BUILDL - Retrieves directory information for the processor and user exit load modules which SUPERMON attaches.

5. EXCPKNT - Retrieves the EXCP counts for disk files used by SUPERMON and its attached processors.

6. JOBSCAN - Verifies the JOB card for a SUPERMON job.

7. WRTCRD - Writes a card to an external file for use by the attached processor.

8. EXECSCN - Verifies the EXEC card for a SUPERMON batch job.

9. USERINF - Calls the user exit if one exists for the job's project number.

10. LINEKNT - Totals the number of lines output to the SYSOUT files used by the attached processor.

11. FNDTIME - Converts CPU time to 100ths of a second.

12. UPDACCT - Updates the OSU active file and prints the accounting box.

13. SMFREC - Writes the SMF record.

14. CONVDATE - Converts a Julian date to MMDDYY format.

15. ACCTVER - Performs project number verification and authorization checks.

16. TIMEXT - Timing exit used to post the timer ECB.

The relationship between modules is illustrated in Figure 2.

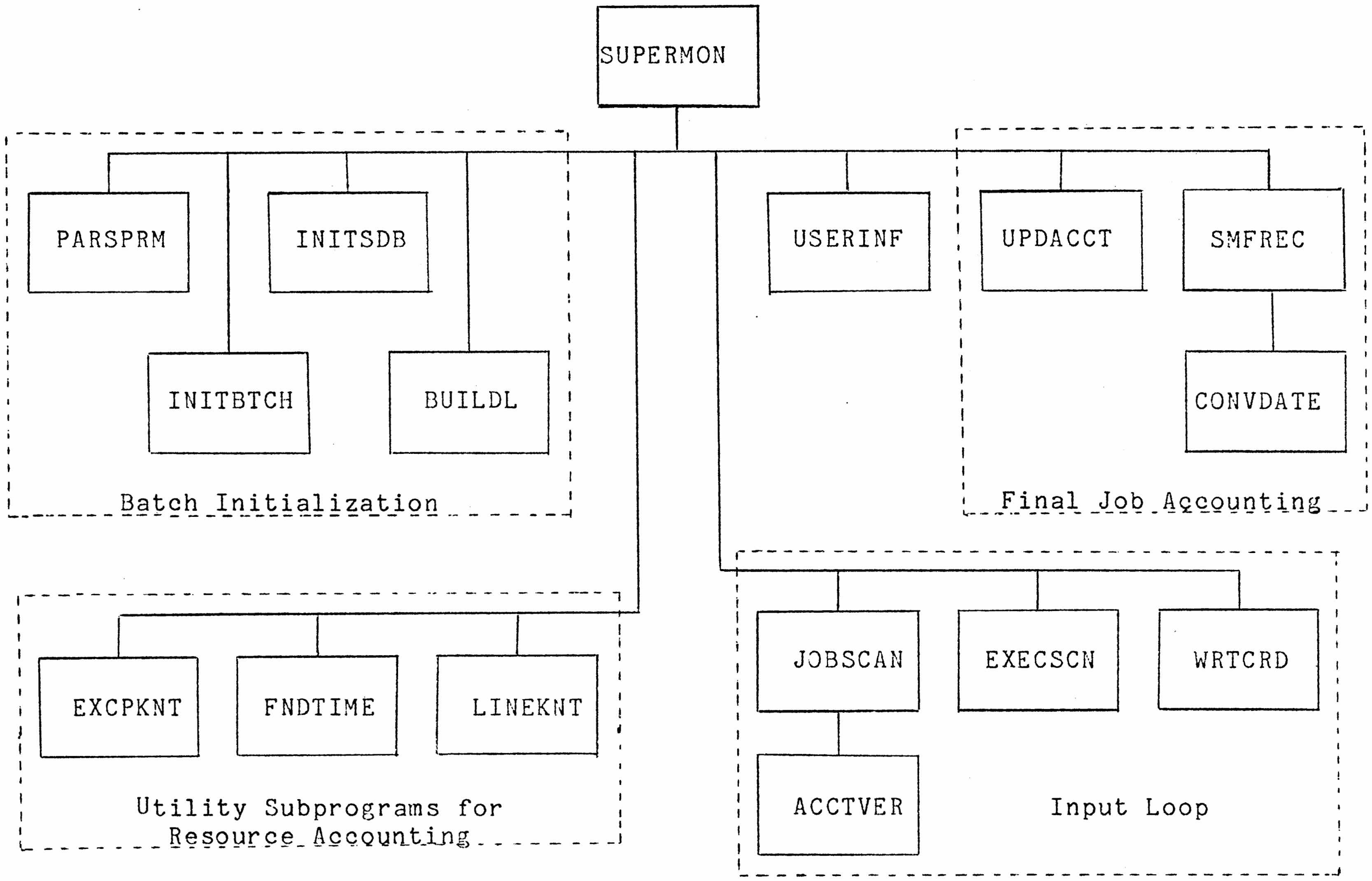

Figure 2. SUPERMON Hierarchical Module Chart

Files

SUPERMON utilizes the following four files:

1. SYSIN - The input file created by JES from which SUPERMON reads the Class Z jobs to be processed.

2. SYSIN$$$ - The output file which SUPERMON creates for the processor which it attaches.

3. TUOREPUS - The output file for messages to the user job.

4. ACCTFILE - The OSU active file which SUPERMON must read and update for accounting purposes.

SUPERMON also must open and close the SYSOUT files used by the attached processors in order to save pointers to the line count field in the SDB. This is necessary to control the number of lines output by a Class Z job. These files are FT06F001, SYSOUT, SYSPRINT, and COBPRINT.

Other files are required by the attached processors but are not directly accessed by SUPERMON. These files may be found in the procedure used to start SUPERMON processing, B$$$$Z1.

System Flow

A very brief overall description of SUPERMON processing is shown in Figure 3. Details may be found in each of the program descriptions elsewhere in this document.

```
INITIALIZATION FOR THE ENTIRE BATCH RUN;
DO WHILE THERE ARE MORE JOBS TO PROCESS;
   DO UNTIL THE END OF A CLASS Z JOB IS FOUND OR
    AN ERROR OCCURS;
      READ THE NEXT CARD FROM INPUT;
      IF IT IS A JOB CARD THEN
         PERFORM JOB CARD VALIDATION;
      IF IT IS AN EXEC CARD THEN
         PERFORM EXEC CARD VALIDATION;
      IF IT IS A DATA CARD THEN
         WRITE THE CARD TO SYSIN$$$ FILE;
   END;
   IF THE JOB HAS PASSED ALL VALIDATION CHECKS THEN
      DO;
         ATTACH THE REQUESTED PROCESSOR;
         WAIT FOR ITS COMPLETION;
      END;
   UPDATE THE OSU ACTIVE FILE;
   WRITE AN SMF RECORD;
END;
```

Figure 3. SUPERMON Logic PDL

Program Description - SUPERMON

Abstract

SUPERMON is the main csect of this system. As such, its overall flow is given in Figure 3. A more detailed description is given in "Program Logic" below.

Link Edit Attributes

Link edit attributes are only given for the main program SUPERMON since all programs are linked together as one module as previously explained. The only attribute required is 'AC=1'. SUPERMON must be an authorized program in order to place itself into supervisor state. This is necessary to allow SUPERMON to perform such authorized tasks as writing SMF records while attaching unauthorized subtasks.

Subroutines

The subroutines which SUPERMON calls are listed above. Two other external subroutines utilized by SUPERMON but not written by this author are:

1. ACCTRW - Access routines to the OSU active file. SUPERMON uses the entry points OPEN, READ and WRITE.

2. NQDQ - Enqueue routines for the OSU active file. SUPERMON uses the entry points ENQ and DEQ.

Documentation for these subroutines may be found in the UCC Systems Documentation Manual.

Macros

SUPERMON uses the following local macros which are all contained in 'SYS1.SYS.MACLIB'.

1. SMCOMMON - Layout of the common storage area. This area contains information which is used by many of SUPERMON's subroutines.

2. SMACTREC - Layout of the OSU active file record.

3. SMFRC254 - Layout of the SMF record which is generated by SUPERMON for each job it processes.

Inputs and Outputs

1. Parameter - SUPERMON is passed a parameter string via the EXEC statement in the procedure which JES uses to begin SUPERMON processing. This parameter string consists of project numbers which have user exits and the entry point names for the user exits. The exact form is '99999,eeeeeeee,99999,eeeeeee...' where '99999' is a valid OSU project number and 'eeeeeeee' is the one to eight-character entry point name. The project numbers and corresponding entry point names must be in ascending alphabetical order by entry point name. At present a limit of ten user exits may be specified, although this number could be increased with minor modifications to SUPERMON.

2. Files - File requirements for SUPERMON are discussed in the 'System Description' above.

3. Messages - SUPERMON outputs messages to the user job via the file TUOREPUS. These include:

OSU001I	OSU002I	OSU006I	OSU007I
OSU008I	OSU009I	OSU011I	OSU014I
OSU015I	OSU016I	OSU019I	OSU032I
OSU101I	OSU102I	OSU103I	OSU104I
OSU105I	OSU106I	OSU115I	

   SUPERMON outputs the following messages to the operator console:

OSU024I	OSU025I	OSU107I	OSU108I
OSU109I	OSU110I	OSU111I	OSU112I
OSU113I	OSU114A	OSU116I.	OSU117I
OSU118I	OSU119I	OSU120I	

   These messages are listed elsewhere in this document. If MSGCLASS=A is specified on the JOB card, SUPERMON also prints the input JCL and the accounting box on file TUOREPUS.

Program Logic

SUPERMON logic may be broken into two main parts. The first consists of initialization for the batch run. This part is executed only when JES invokes SUPERMON for the first time after its initiator has been drained and re-started. Processing which is performed includes:

1. Writing a message to the operator console that SUPERMON has started,
2. Parsing the input parameter string containing user exit information,
3. Opening the required files,
4. Initializing pointers to certain control block information,
5. Initializing pointers to the SDBs of the SYSOUT files used by the attached processors,
6. Retrieving directory information for the processor and user exit load modules,
7. Placing itself into supervisor state.

The second part of SUPERMON is in reality an infinite loop which is executed once for each Class Z job which it processes. The processing within this loop is comprised of the following steps:

1. Read the input file until a JOB card is found.
2. Verify the accounting information and other keywords on the JOB card. If there is an error, print the message and go to Step 1.
3. Read each card of the input file SYSIN until a null JCL card is found. SUPERMON utilizes the internal subroutine CREAD to determine

the type of card read. Depending on the type, do one of the following things:

a. For an EXEC card, verify that the processor name is valid.

b. For a /*PASSWORD card, save the password for later validation. (The last password card found is used in password checking.)

c. For a delimiter card (/* or user-specified), a JCL comment card, or a //SYSIN DD * card, print the card if desired (MSGCLASS=A).

d. For a JOB continuation card, continue the JOB card validation.

e. For any other type of JCL card, print an error message.

f. For any other type of non-JCL card, write the card to the external file SYSIN$$$ to be used by the attached processor.

4. Write out the last block of data for the file SYSIN$$$.

5. Perform miscellaneous validation checks such as final input card count and password checking.

6. If an error has occurred, go to Step 8.

7. Call the user exit routine if one exists for the project number of the current job.

8. Perform a 'dummy' close on the file SYSIN$$$. (This allows SUPERMON to begin writing at the beginning of the file without opening it again.)

9. If the user exit requested cancellation of the job, go to Step 14. If some other error occurred, go to Step 1.

10. Attach the requested processor (WATFIV, WATBOL, ASSIST, PL/C, ROUTE, or LIST).

11. Set a timer to activate SUPERMON after a given period of real time.

12. Wait for one of two events: the attached processor finishing or the timer expiring. If the attached processor is finished, go on to the next step, otherwise check the CPU time used and the lines output for the Class Z limits. If they exceed the limits, go to the next step, otherwise reset the timer and start this step again.

13. Detach the processor subtask.

14. Calculate the CPU time used and the disk EXCPs performed by the job.

15. Update the active file and print the accounting box if requested.

16. Issue the SMF record type 254 for the job.

17. Go back to Step 1 of the infinite loop to process the next job.

When SUPERMON attempts to read the next job, JES detects this condition and temporarily suspends SUPERMON processing while it performs termination tasks for the job which just completed. These include spooling the output files, queuing the job for printing and so forth. If another Class Z job is waiting to be executed, JES reactivates SUPERMON immediately at this point. Otherwise, SUPERMON remains in a wait state until another Class Z job is read by JES.

There are only two methods of terminating SUPERMON's infinite loop. The first is by draining or cancelling the initiator. The second is if SUPERMON abnormally terminates.

Program Description - PARSPRM

Abstract

PARSPRM is the csect which parses the input parameter string to SUPERMON and builds the user exit project number table and the BLDL list for the user exit entry points.

Inputs and Outputs

1. Parameters - PARSPRM is passed three parameters:

   a. The address of the parameter string to be parsed. (See the section on parameters in the program description of SUPERMON for a description of the format.)

   b. A table which on output contains UCC project numbers with user exits.

   c. The BLDL list for the user exits. This list must be initialized to blanks except for the field containing the length of each BLDL entry which must contain the appropriate length.

2. Return codes - PARSPRM returns a condition code of 4 if it encounters an error in the parameter string, otherwise it returns a condition code of zero.

Program Logic

PARSPRM simply scans the parameter string alternately picking off a UCC project number and its corresponding user exit entry point name. It expects the project number to be exactly 5 digits in length. It uses the TRT (translate and test) instruction to locate the comma following a user exit entry point name so that the names can be of any valid length (1 to 8 characters). It places the entry point names in the BLDL list for use by the subroutine BUILDL. Use of the BLDL facility is explained in the program description of BUILDL.

Program Description - INITPTRS

Abstract

INITPTRS is the csect which initializes certain pointers which remain constant throughout the life of a SUPERMON batch run. These include pointers to the SDB (Subsystem Data Set Block), the JCT (Job Control Table), the TCT (Timing Control Table), and the CPU elapsed time field from the ASCB (Address Space Control Block). Also returned is the CPU System Identifier from the SMCA (System Management Control Area).

Parameters

INITPTRS expects the following six parameters, the first of which is input and the rest output.

1. Address of the DCB for the file which SUPERMON uses for its user messages. (Needed to locate the SDB and JCT.)
2. Address of the SDB.
3. Address of the JCT.
4. Address of the CPU elapsed time field from the ASCB.

5. System Identifier from the SMCA.

6. Address of the TCT.

<u>Program Logic</u>

INITPTRS simply traces through the appropriate control blocks for the required information. This operation is well documented in the internal program documentation.

Program Description - INITSDB

<u>Abstract</u>

INITSDB is the csect which finds and saves pointers to the SDB for each of the SYSOUT files used by the processors which SUPERMON attaches. These pointers are needed to locate the current line count for each of the files. The line count is used in determining if a job processed by SUPERMON has exceeded its output line limit.

<u>Inputs and Outputs</u>

1. Parameters - INITSDB expects the following two parameters, both of which are output:

   a. SDB address table

   b. Failing DDname

   The latter parameter is used only if an OPEN fails for one of the DDnames.

2. Return codes - INITSDB returns a condition code of 4 if any of the OPENs for the SYSOUT files returns a non-zero condition code. Normal processing is indicated by a return code of zero.

Program Logic

INITSDB progresses through a table of DDnames in its own local storage performing the following for each DDname:

1. Opens the file.
2. Locates the SDB.
3. Saves the pointer to the SDB in the next location of the table passed to it.
4. Closes the file.

One DCB is used for all of the SYSOUT files.

Program Description - BUILDL

Abstract

BUILDL is the csect which performs BLDLs for the processors and user exits which SUPERMON attaches. Use of the BLDL facility allows SUPERMON to reduce the access time in retrieving the load modules to be attached. The BLDL list contains the directory information of a partitioned library for these members. This allows SUPERMON to skip the directory access when attaching the processors and user exits.

Inputs and Outputs

1. Parameters - BUILDL expects the following three parameters to be passed to it:
   a. The BLDL list for the processors which SUPERMON attaches.
   b. The BLDL list for the user exits which SUPERMON attaches.

c. A table of user exit OSU project numbers in the same order as the user exit BLDL list.

Both of the BLDL lists must be completed prior to calling this procedure. The format for the BLDL list may be found in MVS Data Management Macro Instructions.

2. Return codes - This procedure returns a code of 4 if it receives a non-zero return code from the BLDL for the processors. Normal processing is indicated by a return code of zero.

Program Logic

BUILDL performs two BLDLs, one for the processors and one for the user exits. As noted above, both BLDL lists must be completed before entry to this procedure.

If an error is returned from the BLDL for the processors, this procedure determines the failing entry point, issues a message to the operator, and returns a code of 4. This causes SUPERMON to shut itself down, as this error must be corrected before processing can continue.

If an error is returned from the BLDL for the user exits, this procedure determines the failing entry point, deletes the corresponding OSU project number from the table, and issues a warning message to the operator. This has the effect of cancelling the failing user exit without impacting SUPERMON processing.

Program Description - JOBSCAN

Abstract

JOBSCAN is the csect which performs the verification of a JOB card for a job processed by SUPERMON.

Subroutines Called

JOBSCAN calls ACCTVER to verify the accounting information.

Macros

JOBSCAN uses the macro SMCOMMON from 'SYS1.SYS.MACLIB' to define the layout of SUPERMON's common storage area.

Inputs and Outputs

1. Parameters - Although JOBSCAN receives no parameters, it expects the following registers to contain the indicated information:

   a. R5 - Points to 'JOB' on the card to be scanned.

   b. R6 - Points to the end of the card to be scanned.

   c. R11 - Points to SUPERMON's common storage area.

2. Return codes - JOBSCAN returns one of the following condition codes:

   a. 0 - Normal completion, no errors encountered.

   b. 4 - Error found on JOB card or in account verification routine, applicable error address and length stored in CSA.

c. 8 - ACCTVER returned condition code of 8 which indicates a bad read of the active file.

Program Logic

JOBSCAN performs two main functions. It verifies the accounting information and searches for certain keywords on the JOB card. Once it validates the format of OSU accounting information, it calls ACCTVER to read the active file record and perform authorization checks for the project number. Next, it loops searching for keywords on the JOB card. If the card being scanned is a continuation card, only the keyword search is performed.

If JOBSCAN encounters an error, it sets the error message address and length fields in the CSA so that its caller (SUPERMON) can issue the appropriate message to the user. It also sets the return code as shown above.

Program Description - ACCTVER

Abstract

ACCTVER is the csect which performs account verification and authorization for a job which SUPERMON is processing.

Macros

ACCTVER uses the following macros from 'SYS1.SYS.MACLIB'.

1. SMACTREC - Layout for the OSU active file

record.

2. SMCOMMON - Layout for SUPERMON's common storage area.

Inputs and Outputs

1. Parameters - ACCTVER expects no parameters, but does require that the address of SUPERMON's common storage area be in register 11.

2. Return codes - ACCTVER returns one of the following codes:

   a. 0 - Normal processing, no errors encountered.

   b. 4 - Project number in question failed one of the authorization checks.

   c. 8 - I/O error on the active file.

Program Logic

ACCTVER retrieves the project number for verification and authorization checks from the common storage area. The checks which it makes are:

1. Numeric project number

2. Open account

3. Positive account balance or unlimited funds

4. Current shift authorization

It also saves the byte of flags which provide processor authorization in the common storage area for later use by EXECSCN.

Program Description - EXECSCN

Abstract

EXECSCN is the csect which scans the EXEC card of a job, looking for a valid procedure name.

Macros

EXECSCN uses the macro SMCOMMON from 'SYS1.SYS.MACLIB' to define the layout of SUPERMON's common storage area.

Inputs and Outputs

1. Parameters - Although EXECSCN receives no parameters, it expects the following registers to contain the indicated information:

   a. R5 - Points to 'EXEC' on the card to be scanned.

   b. R6 - Points to the end of the card to be scanned.

   c. R11 - Points to SUPERMON's common storage area.

2. Return codes - EXECSCN returns a condition code of 4 if it encounters an error on the EXEC card, otherwise it returns zero.

Program Logic

EXECSCN first attempts to locate a procedure name following 'EXEC' on the card to be scanned. Then it compares the name found to a table of valid procedure names contained in its local storage. If a valid one is found, it checks for authorization from the flags from the active file which have been saved in the CSA. Lastly, it moves the appropriate parameter to be passed to the attached processor to a field in the CSA.

If an error is encountered, it uses the error message address and length fields of the CSA to allow its caller (SUPERMON) to output the appropriate message to the user.

This routine must be modified if additional processors are to be added to SUPERMON's capabilities. Instructions for doing this may be found in the internal program documentation.

Program Description - WRTCRD

Abstract

WRTCRD is the csect which is responsible for writing records to the file which the attached processor accesses for its input. It uses the BSAM access method and is responsible for blocking the data.

Macros

WRTCRD uses the macro SMCOMMON from 'SYS1.SYS.MACLIB' to define the layout of SUPERMON's common storage area.

Parameters

WRTCRD expects one input parameter, the address of the DCB to which it is to write records. The address of the common storage area must also be in register 11.

Program Logic

WRTCRD moves the card from its input buffer in the common storage area to the next location in the output buffer which it has in its local storage. When the buffer becomes full, it issues a write and does not attempt to place more records into the buffer until it issues a check which ensures that the I/O operation is complete.

If WRTCRD is called and the FINALWRT flag in the common storage area is on, the routine simply writes the block which it has already built, provided there are records to write.

Program Description - USERINF

Abstract

USERINF is the interface between SUPERMON and its user exits.

Macros

USERINF uses the macro SMCOMMON from 'SYS1.SYS.MACLIB' to define the layout of SUPERMON's common storage area.

Inputs and Outputs

1. Parameters - USERINF expects one input parameter, the address of the BLDL list for the user exits. It also requires that the address of the common storage area be passed in register 11.

2. Return codes - USERINF returns a condition code of 4 if the user exit requested

cancellation of the job, otherwise it returns zero.

Program Logic

USERINF first determines if a user exit exists for the current project number by checking the project number table in the common storage area. If not, it simply returns to its caller. If an exit does exist, it sets up the required parameters and attaches the appropriate routine. In the remote possibility that the user exit gets in an infinite loop, USERINF sets a timer to reactivate itself in order to check the CPU time used by the exit. If it is forced to detach the user exit before its completion, it issues a message to the console so that the situation does not go unnoticed.

If the user exit returns a non-zero completion code, USERINF formats a message to the user that the job has been cancelled because of the user exit. The error message address and length are then saved in the common storage area so that the caller (SUPERMON) can issue the message.

Program Description - TIMEXT

Abstract

TIMEXT is the timer exit for SUPERMON. It receives control when the timer expires. The timer is used to re-activate SUPERMON out of the wait state it enters after it has attached a subtask.

Program Description - FNDTIME

Abstract

FNDTIME is used by SUPERMON to convert CPU time from its internal format to hundredths of seconds.

Parameters

FNDTIME expects two parameters; the first is the address of the CPU time field, and the second is the area to return the result.

Program Logic

The input CPU time field is an unsigned 64-bit fixed point number where bit 51 is equivalent to one microsecond. The result is returned in a full word binary number representing the equivalent time in 100ths of a second. The conversion process is explained in the internal program documentation.

Program Description - LINEKNT

Abstract

LINEKNT is the csect which totals the number of lines output to the SYSOUT files used by the attached processors of SUPERMON.

Parameters

LINEKNT expects two parameters, a fullword area to return the count and a table of SDB addresses, one address for each SYSOUT file to be included in the count.

Program Logic

LINEKNT progresses through the table of SDB addresses sent to it, locating the line count field and adding it to a total.

Program Description - EXCPKNT

Abstract

EXCPKNT is the csect which totals the EXCPs performed on the disk files used by SUPERMON.

Parameters

EXCPKNT expects the following three parameters, the first of which is input:

1. Address of the TCT (Timing Control Table)
2. Fullword count of the disk EXCPs
3. Fullword count of the VIO EXCPs

Program Logic

This procedure uses the information in the I/O table of the TCT to locate and sum the disk and VIO (Virtual I/O)

EXCPs. This operation is well defined in the internal program documentation.

Program Description - SMFREC

<u>Abstract</u>

SMFREC is the csect which formats and writes the SMF record issued by SUPERMON for each job it processes.

<u>Macros</u>

SMFREC uses the following macros from 'SYS1.SYS.MACLIB':

1. SMFRC254 - Layout of the SMF record type 254
2. SMCOMMON - Layout of SUPERMON's common storage area.

<u>Parameters</u>

SMFREC expects one input parameter, the address of the SMF record it is to format and write. It also expects the address of SUPERMON's common storage area to be passed in register 11. Much of the information which it uses to fill in SMF record fields comes from the common storage area.

<u>Program Logic</u>

SMFREC obtains and formats the following information for the SMF record:

1. Job name and number,
2. Date and time job started,

3. Accounting information to include the UCC project number and the social security number,

4. EXCP and input card counts,

5. Amount charged for the job,

6. Date and time job ended.

The procedure uses the MVS macro SMFWTM to perform the actual write of the record and outputs a message to the operator if an error occurs.

Program Description - UPDACCT

Abstract

UPDACCT is the csect which updates the active file and prints the accounting box if requested (MSGCLASS=A).

Macros

UPDACCT uses the macro SMCOMMON from 'SYS1.SYS.MACLIB' to define the layout of SUPERMON's common storage area.

Parameters

UPDACCT expects one input parameter, the address of the DCB to use in writing the accounting box. It also expects the address of SUPERMON's common storage area to be passed in register 11.

Program Logic

UPDACCT calculates and formats the following accounting information:

1. Processor time charge,
2. Processor storage charge,
3. Disk EXCP charge,
4. Discount for shifts other than the prime shift,
5. Total charge for the job,
6. Amount of funds remaining in the project account.

It also updates the active file with this information. The accounting box is printed only if the user specifies MSGCLASS=A on the JOB card.

APPENDIX C

SUPERMON ERROR MESSAGES

OSU001I JCL ERROR - PERFORM= PARAMETER IS NOT PERMITTED

Explanation: The PERFORM= parameter has been detected on the JOB card. This is not in accordance with OSU JCL standards. This message is issued by the subroutine JOBSCAN.

System Action: The job is terminated without execution.

Operator Response: None.

Programmer Response: Remove the PERFORM= parameter from the job and resubmit.

OSU002I JCL ERROR - INVALID SYNTAX IN ACCOUNTING SUBFIELDS

Explanation: The accounting subfields on the job card must be specified in the following format:

```
//jobname JOB (nnnnn,sss-ss-ssss,......
```

where 'nnnnn' is a valid UCC project number and 'sss-ss-ssss' is the social security number of the user. This message is issued by the subprogram ACCTVER.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Resubmit the job specifying UCC project number and social security number in the format shown above.

OSU006I JCL ERROR - NON-NUMERIC CHARACTER ENCOUNTERED IN PROJECT NUMBER

Explanation: The project number in the accounting subfields of the JOB card contains non-numeric data. This message is issued by the subprogram JOBSCAN.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Resubmit the job specifying a valid project number on the JOB card. The project number format is described in the message OSU002I.

OSU007I JCL ERROR - NON-EXISTENT OR CLOSED PROJECT NUMBER

Explanation: The project number used in the accounting subfields of the JOB card is not active. This message is issued by the subprogram ACCTVER.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Check the project number for correctness. If no error is found, contact the project director or Accounting Services of the University Computer Center.

OSU008I JCL ERROR - INSUFFICIENT FUNDS

Explanation: The account specified by the project number in the accounting subfield of the JOB statement is out of funds. This message is issued by the subprogram ACCTVER.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Contact the project director. Further questions can be forwarded to the Accounting Services Section of the University Computer Center.

OSU009I PROJECT NUMBER NOT AUTHORIZED FOR THIS SHIFT

Explanation: The project number used in the accounting subfield of the JOB card has not been authorized to be used on the shift (1, 2 or 3) that has been attempted. This message is issued by the subprogram ACCTVER.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: See the project director to determine for which shifts the project number is authorized. Any further questions may be forwarded to the Accounting Services Section of the University Computer Center.

OSU011I JCL ERROR - PASSWORD CHECK FAILED

Explanation: The password contained on the /*PASSWORD card is not the valid password for the account, or the /*PASSWORD card has been omitted. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Include a /*PASSWORD card with the proper password in the job stream. If the password is invalid, contact the project director.

OSU014I PROCEDURE OR PROGRAM NOT AUTHORIZED FOR CLASS Z

Explanation: The job is attempting to use a procedure or program which is not designed for use in the CLASS=Z processor. This message is issued by the subprogram EXECSCN.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Resubmit the job using any valid class other than Z. Consult the UCC User's Manual to determine those procedures and programs authorized for CLASS=Z.

OSU015I JCL ERROR - MULTIPLE JOB STEPS NOT AUTHORIZED FOR CLASS Z

Explanation: The job contains more than one job step. This is not allowed for class Z jobs. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Resubmit the job using any valid class other than CLASS=Z.

OSU016I JCL ERROR - ACCOUNT NOT AUTHORIZED FOR CLASS Z PROCESSOR SPECIFIED

Explanation: A job with CLASS=Z specified on the JOB statement has requested the use of a processor (WATFIV, for example) that was not authorized for use by the project number specified in the accounting subfields of the JOB statement. This message is issued by the subprogram EXECSCN.

System Action: The job is purged from the system without execution.

Operator Response: None

Programmer Response: Consult the project director to determine which processors are authorized for the account. Further information can be supplied by Accounting Services of the University Computer Center.

OSU019I JCL ERROR - INVALID DD CARD FOR CLASS Z JOB

Explanation: The class Z processors allow only one DD statement: SYSIN. Any other DD statements contained in the job stream are invalid for the class Z processors. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Change the CLASS=Z parameter on the JOB card to any valid class other than Z, or delete the invalid DD card in the job.

OSU024I ACTIVE FILE OPEN FAILED, CONTACT UCC SYSTEMS

Explanation: The program received a non-zero return code from the OPEN subroutine which opens the active file. This message is issued by SUPERMON.

System Action: SUPERMON is terminated.

Operator Response: Contact UCC Systems as soon as possible.

Programmer Response: None.

OSU025I I/O ERROR ON ACTIVE FILE, CONTACT UCC SYSTEMS

Explanation: The program which issued the message has received a non-zero return code from the subroutine NQDQ or ACCTRW which process the active file. This message is issued by the subprograms ACCTVER and UPDACCT.

System Action: SUPERMON is terminated.

Operator Response: Contact UCC Systems as soon as possible.

Programmer Response: None.

OSU032I JCL ERROR - ADDRSPC=REAL NOT AUTHORIZED

Explanation: The parameter ADDRSPC=REAL has been found on the JOB card. Use of this parameter is not allowed in Class Z. This message is issued by the subprogram JOBSCAN.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Remove the request ADDRSPC=REAL from the JOB card.

OSU101I CLASS Z LINE LIMIT EXCEEDED. JOB CANCELLED.

Explanation: The line limit for a Class Z job has been exceeded. This message is issued by SUPERMON.

System Action: The job is terminated before execution is complete.

Operator Response: None.

Programmer Response: Resubmit the job using any valid class other than Z. Consult the UCC User's Manual for the current line limit of Class Z jobs.

OSU102I JCL ERROR - MISPLACED EXEC CARD.

Explanation: An EXEC card has been found out of order. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Examine the order of the input JCL. Be sure the EXEC card comes before any SYSIN cards. This problem may occur if an unrecognizable card is included before the EXEC card such as a misspelled PASSWORD card. After correcting the error, resubmit the job.

OSU103I JCL ERROR - MISSING EXEC CARD

Explanation: A Class Z job did not include an EXEC card in the input JCL. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Inlude a valid EXEC card specifying one of the Class Z procedures. Consult the UCC User's Manual for the JCL required to correctly submit a Class Z job.

OSU104I JCL ERROR - CLASS Z CARD LIMIT EXCEEDED.

Explanation: A Class Z job has exceeded the maximum number of cards allowed on input. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Resubmit the job using any valid class other than Z. Consult the UCC User's Manual for the current input card limit for Class Z.

OSU105I CLASS Z TIME LIMIT EXCEEDED. JOB CANCELLED

Explanation: A Class Z job has exceeded its time limit. This message is issued by SUPERMON.

System Action: The job is terminated.

Operator Response: None.

Programmer Response: Resubmit the job using any valid class other than Z. Consult the UCC User's Manual for the current time limit for Class Z.

OSU106I JCL ERROR - $$ CARD ENCOUNTERED

Explanation: A Class Z job has included a '$$' card somewhere in its input. This card may not be used in Class Z. This message is issued by SUPERMON.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: Remove the '$$' card from the input job stream or resubmit the job using any valid class other than Z.

OSU107I SUPERMON PROCESSOR ATTACH FAILED, CONTACT UCC SYSTEMS

Explanation: The attach for one of the Class Z processors has failed. This message is issued by SUPERMON.

System Action: SUPERMON will abnormally terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU108I SUPERMON SYSIN$$$ FILE CLOSE FAILED, CONTACT UCC SYSTEMS

Explanation: The close of SYSIN$$$ has failed. This message is issued by SUPERMON.

System Action: SUPERMON will abnormally terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU109I SUPERMON xxxxxxxx FILE OPEN FAILED, CONTACT UCC SYSTEMS

Explanation: The OPEN for the file designated by xxxxxxxx has failed. This message is issued by SUPERMON.

System Action: SUPERMON will abnormally terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU110I SUPERMON RECEIVED INVALID PARM. CONTACT UCC SYSTEMS

Explanation: The parameter specified on the EXEC card of the procedure for SUPERMON is invalid. This message is issued by the subprogram PARSPRM.

System Action: SUPERMON will abnormally terminate.

Operator Response: Leave SUPERMON's initiator drained until the Systems Group has corrected the problem. Message OSU114A will also be issued.

Programmer Response: The systems programmer must correct the invalid parameter in SUPERMON's procedure.

OSU111I SUPERMON USER EXIT xxxxxxxx EXCEEDED MAXIMUM TIME. NOTIFY UCC SYSTEMS.

Explanation: The designated user exit has exceeded its allotted time and is probably looping. This message is issued by the subprogram USERINF.

System Action: None.

Operator Response: Notify UCC Systems. SUPERMON processing will continue.

Programmer Response: The Systems programmer must notify the appropriate project director of the probable user exit problem.

OSU112I SUPERMON RECEIVED ERROR CODE FROM WRITE OF SMF RECORD RC=xx, CONTACT UCC SYSTEMS

Explanation: The write of the SMF record has failed. This message is issued by the subprogram SMFREC.

System Action: SUPERMON will abnormally terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU113I SUPERMON VERSION x.x IS NOW PROCESSING CLASS Z JOBS

Explanation: This message is issued when SUPERMON begins processing. The denotation x.x indicates the current version in use. This message is issued by SUPERMON.

System Action: None.

Operator Response: None.

Programmer Response: None.

OSU114A SUPERMON IS ABNORMALLY TERMINATING. DRAIN ITS INITIATOR AND CONTACT UCC SYSTEMS BEFORE REPLYING Y

Explanation: SUPERMON has encountered an error from which it is unable to recover. This message will be preceded by another message defining the exact problem. This message is issued by SUPERMON.

System Action: SUPERMON will terminate.

Operator Response: Drain SUPERMON's initiator, contact UCC Systems and then reply 'Y' to this message. Unless otherwise instructed, attempt to restart SUPERMON by re-starting its initiator. If the problem recurs, leave the initiator drained.

Programmer Response: The System programmer must correct the applicable problem.

OSU115I JOB CANCELLED BY USER EXIT. RC=xxxx. SEE YOUR INSTRUCTOR OR PROJECT DIRECTOR FOR INFORMATION.

Explanation: A user exit has requested cancellation of a Class Z job. This message is issued by the subprogram USERINF.

System Action: The job is purged from the system without execution.

Operator Response: None.

Programmer Response: See the instructor or project director concerning the reason the job was cancelled.

OSU116I SUPERMON USER EXIT xxxxxxxx ATTACH FAILED. CONTACT UCC SYSTEMS.

Explanation: The attach of the designated user exit has failed. This message is issued by the subprogram USERINF.

System Action: None.

Operator Response: Notify the Systems Group as soon as possible.

Programmer Response: The Systems programmer must correct the problem and notify the applicable project director responsible for the failing user exit.

OSU117I SUPERMON BLDL FAILED BECAUSE OF INSUFFICIENT STORAGE, CONTACT UCC SYSTEMS

Explanation: The BLDL for SUPERMON's processors failed because of insufficient storage. This message is issued by the subprogram BUILDL.

System Action: SUPERMON will terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU118I SUPERMON BLDL I/O ERROR, CONTACT UCC SYSTEMS

Explanation: An I/O error has occurred while SUPERMON was performing a BLDL for its processors. This indicates a problem with the directory of SUPERMON's step library. This message is issued by the subprogram BUILDL.

System Action: SUPERMON will terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU119I SUPERMON CONTAINS INVALID BLDL LIST, CONTACT UCC SYSTEMS

Explanation: BLDL has returned a condition code of 4, but the program was unable to detect which processor was not located. This message is issued by the subprogram BUILDL.

System Action: SUPERMON will terminate.

Operator Response: Message OSU114A will also be issued. Refer to it for the proper action.

Programmer Response: None.

OSU120I SUPERMON BLDL FAILED, MISSING ENTRY POINT NAME xxxxxxxx, CONTACT UCC SYSTEMS

Explanation: The entry point indicated by 'xxxxxxxx' is missing. This message is issued by the subprogram BUILDL.

System Action: If the missing entry point is one of the Class Z processors, SUPERMON will terminate. If it is a user exit, SUPERMON will continue processing without it.

Operator Response: If message OSU114A is also issued, refer to it for the proper action. Otherwise, contact UCC systems as time allows.

Programmer Response: None.

APPENDIX D

ACRONYMS

ASCB	- Address Space Control Block
BLDL	- Macro used in retrieving directory information from a partitioned data set.
BSAM	- Basic Access Method
CPU	- Central Processing Unit
CSA	- Common Storage Area
DCB	- Data Set Control Block
EXCP	- EXecute Channel Program
JCL	- Job Control Language
JCT	- Job Control Table
JES	- Job Entry Subsystem
MVS	- Multiple Virtual Storage
RMF	- Resource Management Facility
SDB	- Subsystem Data Set Block
SMCA	- System Management Control Area
SMF	- System Management Facilities
TCT	- Timing Control Table
VIO	- Virtual Input/Output

VITA 2

Sharon Joyce Clarke

Candidate for the Degree of

Master of Science

Thesis: AN EXECUTION BATCH MONITOR FOR PROCESSING STUDENT JOBS

Major Field: Computing and Information Sciences

Biographical:

Personal data: Born in Ponca City, Oklahoma, on October 17, 1949, the daughter of John and Marcelle Clarke.

Education: Graduated from Ponca City High School, Ponca City, Oklahoma, in June, 1967; received Bachelor of Science degree in Mathematics from Oklahoma State University in December, 1974; completed requirements for Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in December, 1979.

Professional Experience: Associate Programmer/Analyst, Standard Oil Company (Indiana), 1975-1977; graduate teaching assistant, Oklahoma State University, Computing and Information Sciences Department, 1977-1978; systems programmer, University Computer Center, Oklahoma State University, 1978-1979.