A MODEL OF A MULTIPROGRAMMED DEMAND PAGING COMPUTER SYSTEM

by

Judith Lynn Piggins

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREES OF

BACHELOR OF SCIENCE

and

MASTER OF SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September, 1973

Signature of Author ______________________________

Department of Electrical Engineering, August 13, 1973

Certified by ______________________________

Thesis Supervisor

Accepted by ______________________________

Chairman, Departmental Committee on Graduate Students

A MODEL OF A MULTIPROGRAMMED

DEMAND PAGING COMPUTER SYSTEM

by

Judith Lynn Piggins

Submitted to the Department of Electrical Engineering on August 13, 1973 in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science.

ABSTRACT

This thesis describes a model of a demand paging computer system and provides samples of results obtained from experiments made with the model. The model is implemented in PL/1 in a highly modularized and parameterized form. This form facilitates adjustments to the model to enable it to be used in simulating systems of many different types. The model is well suited to comparative studies of systems where one or more of the parameters of modules is systematically varied to yield a spectrum of results. The output from sample runs is discussed in some detail as an illustration of the use of the model. Finally, some of the limitations of the model are discussed along with suggestions for extensions and improvements to it and possibilities for further experiments.

THESIS SUPERVISOR: Stuart E. Madnick
TITLE: Assistant Professor of Management

ACKNOWLEDGEMENTS

A note of thanks is due first to Professor Stuart Madnick who advised this thesis for his suggestions, criticisms and support of the work. Appreciation is due also to Professor F.J. Corbató for his interest and understanding during the course of the work described here. Mr. Joseph R. Steinberg and Mr. Leo Ryan of the MIT Information Processing Center were very helpful in providing the calibration data used in the experiments described in this document. Finally, the author is grateful to Mr. Harry Forsdick and Mr. Charles Lynn for their suggestions and encouragement throughout the development of the model described here and their comments on the written results.

J.L.P.

Cambridge, Massachusetts

August, 1973

TABLE OF CONTENTS

	Page
ABSTRACT	2
ACKNOWLEDGEMENT	3
LIST OF FIGURES	5
LIST OF TABLES	6
CHAPTER 1 - INTRODUCTION	7
Motivation for the Research	8
Overview of the Scheduling Process	9
Summary of Related Literature	16
Summary of the Thesis	18
CHAPTER 2 - DISCUSSION OF THE MODEL	20
Overview	20
Description of the Model	27
Data Bases	32
Detailed Discussion of the Modules Comprising the Model	38
CHAPTER 3 - SOME EXPERIMENTAL RESULTS OBTAINED WITH THE MODEL	82
Parameter Values Describing the Simulated System	83
The Schedulers	92
A Word of Caution	101
Summary of Results from the Test Runs	104
Resource Usage	105
Job Behavior	115
Overall System Performance	125
Conclusions	133
CHAPTER 4 - LIMITATIONS OF THE MODEL AND SUGGESTIONS FOR FURTHER STUDY	137
Limitations and Inaccuracies	139
Additions and Improvements	147
Some Further Experiments	157
APPENDIX A - HOW TO USE THE MODEL	163
APPENDIX B - LISTINGS OF THE TEST SCHEDULERS	177
APPENDIX C - SAMPLE OUTPUT PRODUCED BY THE MODEL	195
APPENDIX D - SAMPLE STUDENT ASSIGNMENT	213
BIBLIOGRAPHY	238

LIST OF FIGURES

		Page
1-1	Basic Scheduling Scheme	11
1-2	Alternative Scheduling Scheme	14
2-1	General Form of Simulated System	25
2-2	Structure of the Model	28
2-3	Job Stream List Entry	33
2-4	System Event List	37
2-5	Main Loop of Supervisory Routine	41
2-6	General Form of Scheduler	45
2-7	Scheduler Data Structures	52
2-8	Curve Governing Time Between Page Faults	58
2-9	Example of Generation of Time Between Page Faults	60
4-1	Simple I/O Network	151
4-2	More Complex I/O Network	154

LIST OF TABLES

		Page
3-1	Average Number of Jobs in Main Memory	106
3-2	Average Amount of Main Memory Assigned to Jobs	107
3-3	Average Number of Pages in Main Memory	107
3-4	Percentage of Maximum Possible Number of Pages in Main Memory (Actual Main Memory Usage)	108
3-5	Total Number of Page Faults Incurred	109
3-6	Average Page Wait Time	110
3-7	Total Number of Peripheral I/O Requests Incurred	112
3-8	Average Peripheral I/O Wait Time	113
3-9	Average Time Between Page Faults	116
3-10	Average Time Between Peripheral I/O Requests	117
3-11	Average Percentage of Active Time in each Traffic Control State	119
3-12	CPU Idle Time	125
3-13	Average Throughput	127
3-14	Average Turnaround Time for the Aggregate Job Stream	129
3-15	Average Number of Jobs in System	130
3-16	Turnaround Time by Priority Level (Preemptive Scheduler only)	133

CHAPTER 1

INTRODUCTION

This thesis describes a model which represents a single processor, demand paged multiprogramming computer system and generates jobs to be scheduled for processing by a scheduling algorithm. The model is designed to facilitate the running and evaluation of different scheduling algorithms for the purpose of determining their relative effectiveness. The model has a number of parameters which may be modified to reflect the characteristics of different systems and different aggregate behavior in the set of jobs to be processed. These parameters may be adjusted to yield a picture of the behavior of any algorithm under a variety of conditions. Clearly, a model such as this one provides only an approximation of the behavior of a real system. It can, however, give a relatively clear idea of the comparative performances of different schedulers in a given environment and of a single scheduler in a spectrum of environments. The model is constructed with fairly extensive error checking. Diagnostic printouts are provided in response to scheduler commands which are invalid. This makes the model well suited for pedagogical use in courses and seminars studying operating systems. In particular, assignments may be made to write different scheduling algorithms, and these programs may then be tested out in the modelled environment.

Motivation for the Research

Scheduling is an important task in contemporary computer systems, both batch-processing and time-sharing systems. It will become increasingly so in the future as the growing size and complexity of computer systems require better algorithms for coordinating the processing of jobs. The only reliable way to determine for certain how well a given scheduling algorithm will perform on a given system is to try it out on that system. Furthermore, this approach is required in order to debug and validate an implementation of a given scheduler. This experimental approach is not always feasible, however, due to difficulties in changing supervisory programs on systems which are already running, or to not having the system available, as in the case of systems which are still in the design stage. Developing a model to reflect the relevant aspects of the system and running various scheduling algorithms in the environment of the model is a logical alternative in such a case. The shortcomings of such a model in terms of inaccuracies of representation of the system under study must be borne in mind when interpreting the results of such studies. However, a model provides the capability of testing scheduling algorithms

over a wide range of conditions, and such experiments lead to some general hypotheses which may prove useful both in the study of scheduling in theory and its implementation in specific cases.

Overview of the Scheduling Process

Scheduling in a multiprogramming system has been viewed by Hansen (1) and Browne et al (2) as being comprised of two basic processes or tasks which interact with one another. First, the scheduler is responsible for choosing the job to be run by the central processor at all times. This is termed short-term scheduling or CPU scheduling. It involves selecting a job from among the set of eligible jobs in the system to be run whenever the processor becomes free. In a preemptive scheduling system the scheduler may also interrupt the running of one job to allow another job to be processed first. Each job assigned to the processor is assigned a timeslice by the scheduler which limits the time for which it may be processed without being interrupted. Secondly, the scheduler is responsible for managing the set of jobs which are assigned space in main memory. This task is referred to as medium-term scheduling or job scheduling. In the context of the model described here jobs having at least one page in core are known as

active jobs; those which do not are termed *inactive*. In order to keep as many runnable jobs in main memory as possible the scheduler can activate and deactivate jobs. *Activating* a job involves allocating a set number of blocks of main memory to that job. This amount of memory, referred to as the job's partition size, represents the maximum number of pages of that job which may be in primary memory at any one time. When a job is activated its first page is brought into main memory. Other pages are brought in on demand as the job runs. *Deactivating* a job causes the job's assigned core space to be freed and any of its pages in core to be written out to secondary storage. A good scheduling algorithm in general is one that manages these interrelated processes so as to maintain a high level of system throughout and keep turnaround times as short as possible. In addition, various other objectives may be important in scheduling for specific systems, such as maintaining a good level of response to terminal users in a time-shared system, or giving special consideration to achieving fast turnaround for jobs of high priority in a batch system.

The basic process involved in scheduling in a multiprogramming system is shown in figure 1-1. This diagram and the accompanying discussion are based on the control framework described by Saltzer (3). The scheduler selects jobs to be processed from among those in the *ready* state;

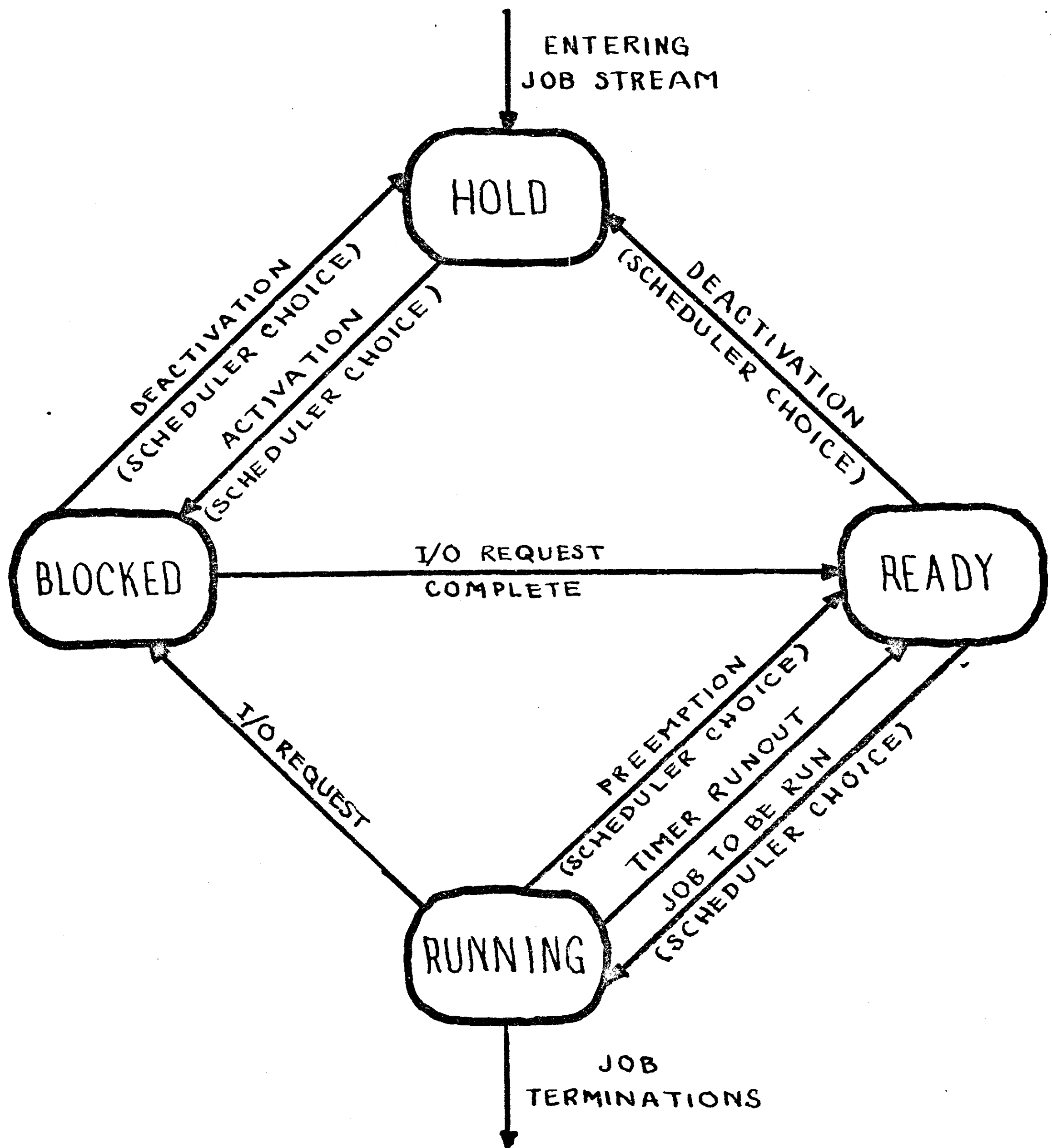


FIGURE 1-1
BASIC SCHEDULING SCHEME

this choice constitutes a transfer to the running state. A job leaves the running state for one of four reasons. First, it may terminate and leave the system. Or it may generate an I/O request, causing it to be transferred to the blocked state. An I/O request here may be either a page fault or an explicit request for disk or tape I/O. Third, it may run out its time allotment, in which case it is returned directly back to the ready state to await its turn to be processed again. Lastly, it may be pre-empted in favor of some other job, and here again it is returned to the ready state to await another chance at the processor. Jobs leave the blocked state and enter the ready state when an interrupt occurs indicating that the request they issued has been satisfied.

Jobs arriving at the system for processing are placed in the hold state. When a job is activated by the scheduler it is promoted to the blocked state. It is placed in blocked rather than ready because it is not eligible to be processed until its first page has been brought into core. This page swap is effectively the same as that performed for a running job which incurs a page fault, and thus it is treated in the same manner. A job chosen to be deactivated may be in either the ready or blocked state; in either case the chosen job is moved back to the hold state.

A variety of scheduling algorithms may be used to per-

form these tasks, depending upon the aims of the particular system in question and the amount of overhead it can tolerate in terms of time spent executing the scheduler. For instance, in a simple batch-processing system where low system overhead is desired a scheme such as first-come-first-served, or round-robin, might be used. Under this scheme, jobs entering the ready state are placed at the end of a single queue and each time the processor becomes free the job at the head of the queue is chosen for processing. Activations are usually performed in the order in which jobs arrive at the system. Often no deactivations are ordered; jobs which are brought into core remain resident until they terminate. This sort of a scheduling algorithm might operate under the basic pattern shown in figure 1-1.

Alternatively, in a system with more diverse requirements such as a time-sharing system which must maintain good response to terminal users as well as performing computational tasks, a more complex framework will probably provide better results. One scheme which might be used in such a case is diagrammed in figure 1-2. In this scheme a job leaving the running state is handled differently depending on the reason for which its processing was suspended. If a job incurs a page fault it is sent to the blocked state, and when the required page has been brought into core the job is placed in the high-priority ready state.

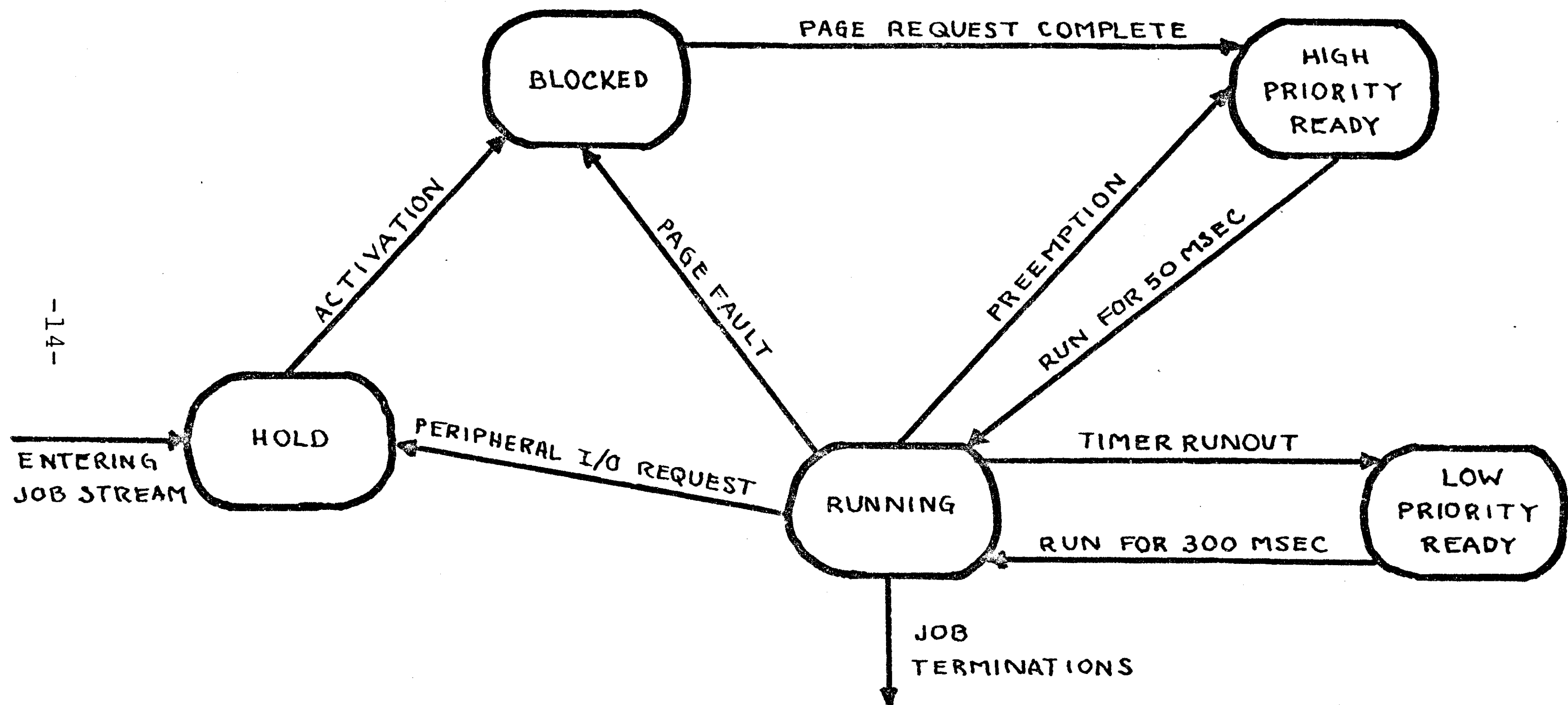


FIGURE 1-2

ALTERNATIVE SCHEDULING SCHEME

If the job runs out its timeslice it is placed in the low-priority _ready_ state. In choosing a ready job to be processed the scheduler first checks the high-priority _ready_ state. If there are jobs in this state one of them is chosen to be run and is assigned a timeslice of fifty milliseconds. Only if this state is empty does the scheduler select a job from the low-priority _ready_ state. Once chosen, however, a job from this state is allowed to run for three hundred milliseconds. The rationale behind this procedure is the assumption that a job which has just incurred a page fault is likely to do so again relatively quickly. Such a job should be run as soon as possible in order to keep the paging devices busy, overlapping I/O and processing as much as possible. Similarly, a job which ran out its timeslice when it was last processed is assumed to be likely to repeat this behavior. Such a job is forced to wait until all jobs which are more likely to require the use of the paging devices in the immediate future have had a chance to run. When it is allowed to run, however, such a job is given a much longer timeslice. This avoids incurring the overhead of stopping it again, and it does not reduce the utilization of the I/O devices since there are no other more urgent jobs waiting to be run. When a job issues a request for disk or tape I/O it is deactivated. The motivation here is that a disk or tape I/O request

generally takes much longer to service than a page request. Rather than allowing main memory to be tied up by a job which is waiting for peripheral I/O and is not eligible to be processed, the job is removed from main memory, making room for other jobs which may be able to do useful work during this period. When its I/O is complete, the job is ready to be reactivated.

Summary of Related Literature

A number of articles dealing with the subjects of scheduling methods and computer system modelling have been published in recent years. These articles are highly varied. Those pertaining to scheduling methods range from discussions of different scheduling strategies to studies of specific schedulers used in actual systems. Those related to the modelling of computer systems include descriptions of both theoretical mathematical models and more practical simulation models. Some of this literature is discussed briefly below. It is hoped that this discussion will serve as a guide to the reader who may wish to further explore some aspect of these areas.

On the subject of scheduling methods, a theoretical discussion of the scheduling process is presented in Hansen (1). A general discussion of different scheduling schemes

may be found in Coffman and Kleinrock (4). Oppenheimer and Weizer (5) give a description of the relative performance of different scheduling algorithms in a time-sharing environment. Studies of the operation of actual systems of the type mirrored by the model described in this thesis under specific scheduling schemes are described by Browne and Lan (2), Bryan and Shemer (6), Arden and Boettner (7), Sherman et al (8), DeMeis and Weizer (9) and Losapio and Bulgren (10).

There are a number of articles which provide a discussion of the reasons for constructing models and the benefits which may be gained from models of different types. These include papers by Calingaert (11), Estrin et al (12) and Lucas (13). In the area of mathematical modelling, McKinney (14), Adiri (15), Gaver (16), and Kleinrock (17) provide general discussions of mathematical models of computer systems. DeCegama (18) and Kimbleton (19) discuss different approaches to the problem of formulating a model of this type. Examples of actual mathematical models and results obtained with them are provided by Fife (20), Rasch (21), Shedler (22), Shemer (23), Shemer and Heying (24) and Slutz (25). In regard to simulation modelling techniques, general discussions of simulation methods as applied to computer system modelling may be found in Blatny

et al (26), Cheng (27), MacDougall (28), and Nielsen (29,30). Various approaches to the problems involved in constructing simulation models of computer systems are discussed by Bell (31), Lynch (32), Nutt (33) and Seaman and Soucy (34). Practical examples of simulation models similar to the one described here may be found in Boote et al (35), Fine and McIsaac (36), Lehman and Rosenfeld (37), MacDougall (38), Morganstein et al (39), Noe and Nutt (40), Rehman and Gangwere (41), Scherr (42), Schwetman and Brown (43) and Winograd et al (44).

Summary of the Thesis

The remainder of this thesis discusses the model described briefly above and gives some examples of results obtained from studies done with it using different scheduling algorithms. Chapter two describes the functions of the various modules of the model and their interactions with one another. The third chapter discusses a comparative study of three schedulers embodying different scheduling strategies. Chapter four contains a discussion of the limitations of the model, suggestions for possible extensions and further experiments to be performed with it. Appendix A gives a discussion of how to use the model including a detailed description of the functions of its various

parameters. Appendix B contains listings of the three schedulers used in the studies described in chapter three and appendix C provides examples of the output produced by the output modules which may be invoked by the model. Appendix D gives a sample assignment which might be given to a student who is to write a scheduler to be run under the model.

CHAPTER 2

DISCUSSION OF THE MODEL

Overview

The differing aims of the various types of contemporary computer systems make it necessary to specify at least the general type of system under study if a realistic picture of system behavior is to be obtained. The model described here simulates a computer system running under a virtual memory, demand paging scheme. Each job is assigned a fixed number of blocks of core, limiting the number of its pages which may be in core simultaneously. The motivation for choosing this type of system was to make the model widely applicable; there are a number of systems of this sort in operation today, and many others which will use similar schemes are now being developed. The model also provides an environment which presents a significant challenge to writing a scheduler which will give good results. A multitude of factors are constantly interacting in a system of this type, and no "best" algorithm for such systems in general has yet been developed.

Another design consideration is the decision of whether the model is to represent a batch-processing or a time-sharing system environment. There is not a sharp

distinction between the two in contemporary computer systems. Time-sharing systems which allow users to run background jobs exhibit the characteristics of both types of systems, and standard batch-processing and time-sharing systems can be run simultaneously on the same hardware. A task in a batch-processing environment generally denotes a single job or job step. If we define a time-sharing task as the work performed for a terminal user between two successive reads to the terminal, then batch and time-sharing tasks are seen to be very similar. They may differ on the average in characteristics such as total compute time or number of disk accesses per job, but they are the same in basic form and may be treated as such. This is the approach taken in the model.

As noted in chapter one, the literature presents a spectrum of models which have been used to represent some or all the aspects of computer systems which are at issue here. For the most part models such as these have treated computer systems in a general and rather abstract manner, deriving detailed mathematical results about them, or have mirrored a single specific system in great detail. The model described here is a simulation model which represents a compromise between these two extremes in that it provides a framework which is both general enough to be similar to a number of different systems and yet detailed enough to yield

a fairly accurate and realistic picture of the behavior of such a system. It is constructed in a modular manner in order to facilitate the modifications which may be necessary to model different types of systems. As another aid to adaptability, a large number of the parameters governing the various functions of the simulated system are made available to the user to be set to appropriate values on each run.

Lucas (13) cites two basic approaches to simulation modelling which are most often used in the simulation of computer systems. One method is generally known as trace-driven modelling. This approach makes use of measurements of the behavior of an actual job stream as input to the model. Probes are placed in an actual system to measure the demands made on the system by the job stream. The sequence of demands obtained from these probes is then used to drive the model. Examples of this type of modelling are found in the work of Sherman et al (8), Cheng (27) and Noe and Nutt (40). This approach to modelling has the virtue that no validation of the job stream is needed, since it is taken directly from an actual system. Also, the performance of the system from which the trace is taken may be monitored to determine its performance during the processing of the traced job stream. This provides a very good basis for comparison of the model's behavior with that found in practice.

The other basic simulation method is known as event-structured or event-oriented simulation. This approach involves maintaining a list of the events which are scheduled to occur in the simulated system at various specified times in the future. The quantities describing the modelled system, such as job stream characteristics and I/O service times, are usually generated from probability distributions. Event-structured models of computer systems are described by MacDougall (28), Nielsen (30), Boote et al (35) and Fine and McIsaac (36). This method offers considerably more flexibility than the trace-driven approach. For example, it allows easy adjustment of certain characteristics of the job stream without necessitating changes in all of them. This capability enables the model user to study different job streams which may be of interest regardless of whether they occur in practice on systems he can monitor. It also makes it unnecessary to have available an actual system which may be monitored. One of the objectives in designing the model described here was to make it applicable to studying a number of different real-world system environments. The event-structured approach is better suited to this task than the trace-driven method, and it was the method chosen to be used here.

The general form of the system simulated by the model is shown in figure 2-1. The single CPU controls some set amount of main memory and some number of paging devices, disks and tape drives. The paging devices are assumed to be drums. The number of devices of each type available in the system may be varied within limits, as may characteristics describing the operation of each individual device. For time-sharing tasks the entering job stream is viewed as coming in from a set of user terminals, and terminating jobs are redirected to the appropriate terminal unit. The operating characteristics of the terminal units are not of significance in the model. An interactive job "arrives" for processing when the command line invoking it has been completely transmitted by a user terminal. The only parameter of interest to the model in this situation is the average arrival rate of the jobs from the set of all terminals on the system. For this reason the characteristics determining the operation of user terminals are not represented in the model. By the same token, the precise identification of the user terminal which issues the request initiating a given job is not of interest and is not represented in the model. Batch jobs submitted through card readers are assumed to be SPOOLed. In other words, they are read in under the control of a supervisory routine

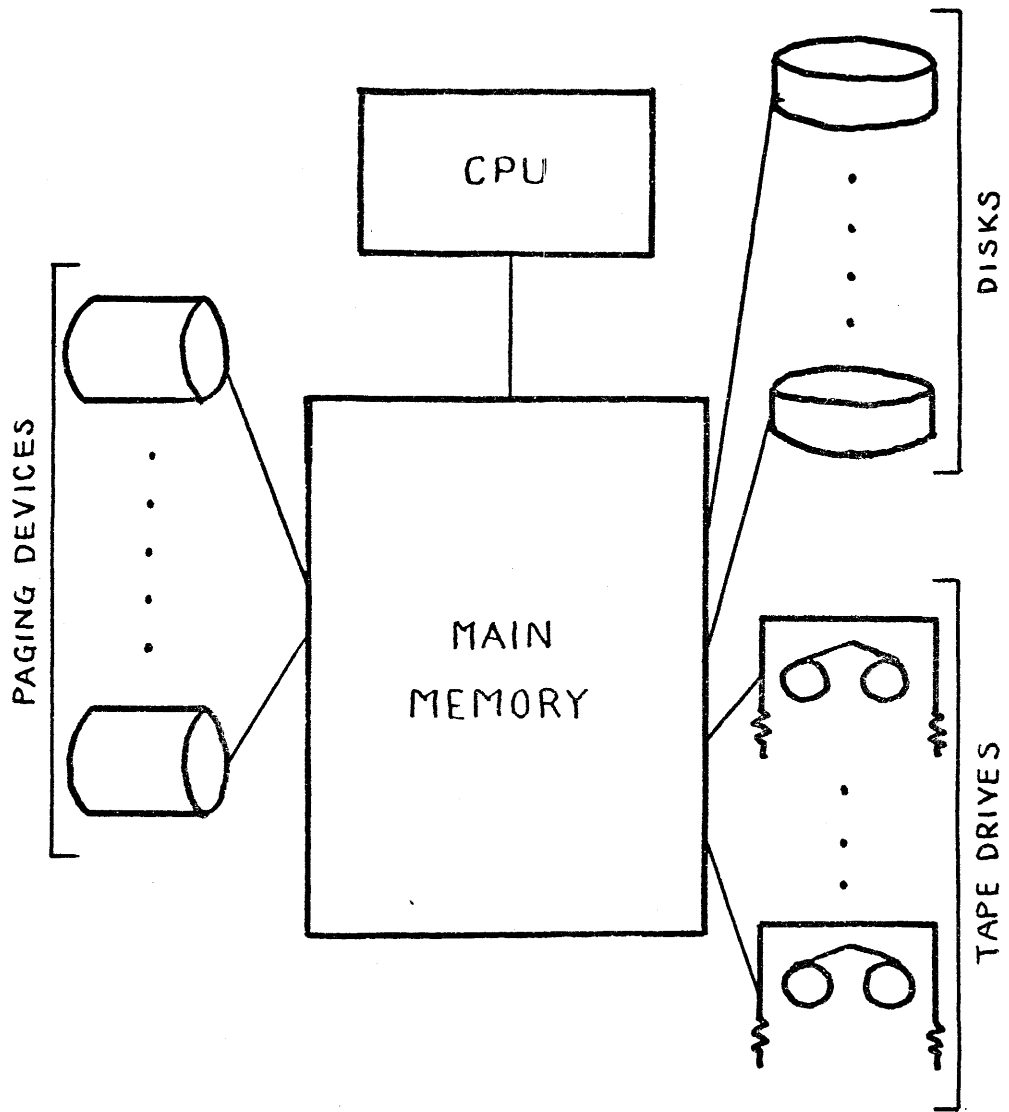


FIGURE 2-1
GENERAL FORM OF SIMULATED SYSTEM

which stores the card images on secondary storage. Thus the jobs are effectively read in from secondary storage. Output to printers and punches is similarly SPOOLed and is effectively written out to secondary storage. Thus the characteristics of any readers, printers and punches available in the system are not of importance and are not represented in the model.

The level of detail in the model varies among its different modules. The various scheduling algorithms which are to be investigated with the model must be written in considerable detail, since these routines are presented with the same variety of information as an actual system would give them. The level of detail in other parts of the model is not as great, since they are intended for support purposes rather than to have their behavior submitted to detailed examination. For instance, the job stream to be generated as input to the simulated system need not be generated as single individually representative tasks, but rather can be treated as a set of separate entities which combine to produce an overall picture of a job load. As Denning has shown (49), it is possible to model a user community and the requests it generates as a whole, but quite difficult to characterize an individual user. The hypothesis that the behavior of the total user population is statistically reproducible has been verified in practice

by measurements made on the Michigan Terminal System (46). Similarly, there is no need to consider which of a job's pages are in core at any time. The scheduler is concerned only with the question of whether a job is eligible to be run and perhaps with estimates of the probable length of time for which it will run before generating a page fault, but not with the particular page for which the fault will occur. Thus the other portions of the model are constructed in much less detail than would be required for a more general model.

Description of the Model

The model is implemented in PL/1. Its general structure is shown in figure 2-2. The arrows indicate the flow of control among the different parts of the model. The Scheduler (SCHED) is the module under investigation and its actions drive the rest of the model, whose actions are governed by the supervisory module (DRIVER). DRIVER maintains the model's data bases and calls the other program modules when their services are required. The Time Between Page Faults module (PGNXT) generates the time intervals between successive page faults for each job, and the Time to Service a Page Fault module (PGTIM) determines the time needed to swap in each requested page. The Time Between Peripheral I/O Requests module (DSTPNXT) generates the

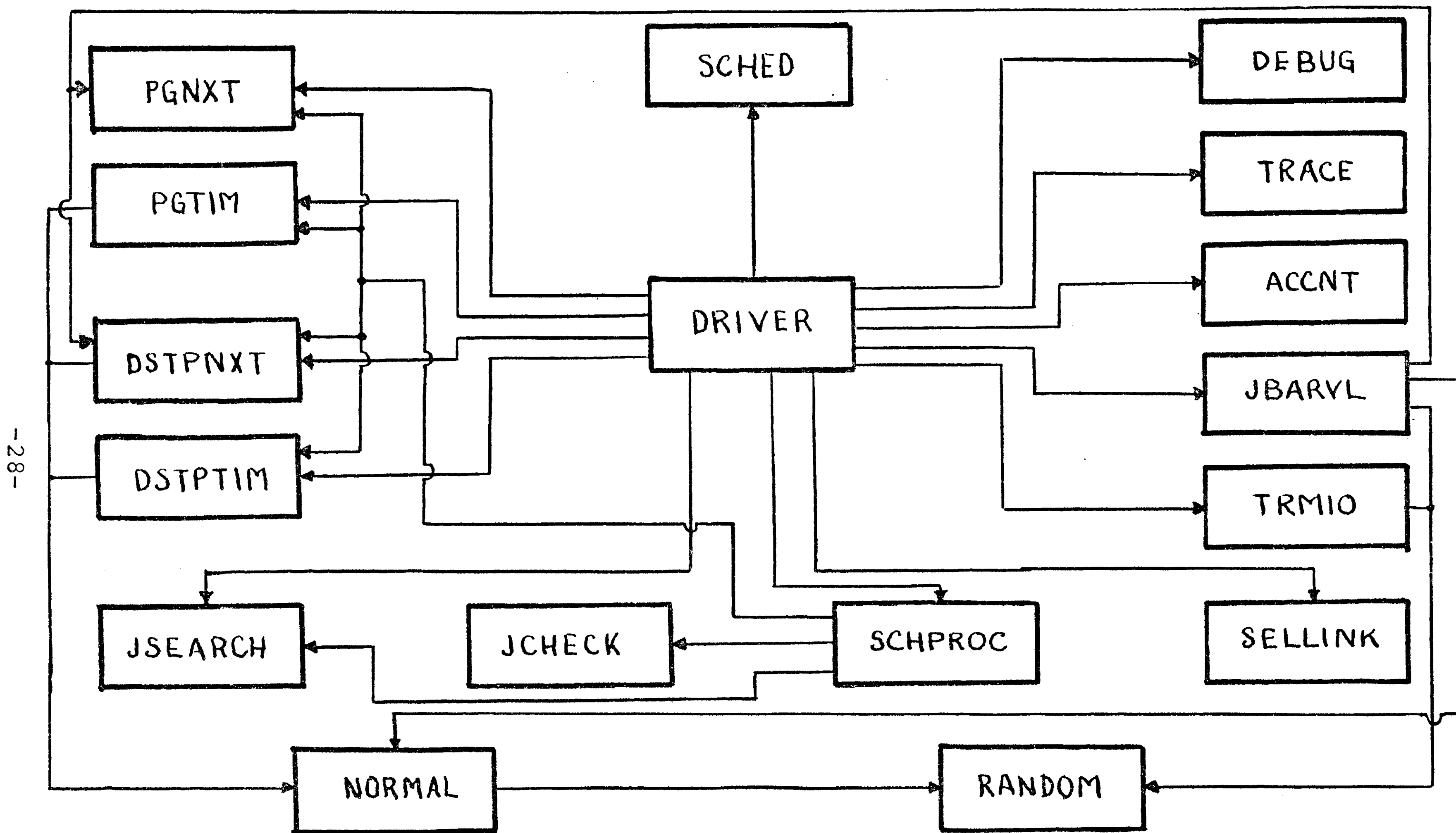


FIGURE 2-2
STRUCTURE OF THE MODEL

intervals between successive disk or tape I/O requests for each job, and the Time to Service a Peripheral I/O Request module generates the time needed to satisfy each such request. The Job Interarrival module (TRMIO) generates values for the interarrival times of new jobs coming into the system, and the Job Characteristics module (JBARVL) determines the static characteristics describing each of these jobs as it enters the system.

The Debug Print Module (DEBUG), the Trace Print module (TRACE), and the Accounting Routine (ACCNT) are modules which produce printed information of various sorts describing the operation of the model. DEBUG produces detailed dumps of the Job Stream List, the System Event List and the System Clock on each iteration of the model. These data bases are described in detail below. This information is useful for debugging the model should problems occur. TRACE produces a sequential listing of the events which occur in the course of the operation of the simulated system and the response of the scheduler to each of these events. ACCNT compiles a number of measures of the performance of the simulated system during the course of the run and outputs them at the conclusion of the run.

The Scheduler Command Processor (SCHPROC) examines the commands issued by the scheduler on each iteration. Correct commands are carried out; those which are in error for any reason are either ignored or replaced by default actions, depending upon the type of the command in which the error occurred. A message is printed explaining the error to the user. The operation of this module includes treatment of many special cases, and its detailed operation is described below. The Procedure to Maintain the System Event List (SELLINK) creates new entries for the System Event List and links them into the existing list in the proper chronological order. The Procedure to Locate a Job Description (JSEARCH) locates the description of a particular job in the Job Stream List and returns a pointer to it. The Procedure to Check Job Eligibility (JCHECK) examines the description of a given job to determine whether it is eligible to be run (i.e. both ready and active). These last two routines are called by various modules of the model as shown in the diagram whenever their functions are required; they are implemented as separate routines to avoid duplication of the code needed to perform these tasks.

The Normal Random Variable Generator (NORMAL) is a procedure which produces a normally distributed value based upon a mean and standard deviation passed as arguments.

The Random Number Generator (RANDOM) produces random numbers evenly distributed between zero and one. These two modules are service modules called by a number of the other modules in the system as shown in the diagram.

The model is designed in this modular fashion so as to allow any of the above modules to be replaced by another which performs the same function according to some other discipline. This makes it easier to adapt the model to reflect the characteristics of different systems. For instance, the model's routine to compute paging service times (PGTIM) as described below assumes FIFO queuing of page requests at the paging devices. If it is desired to explore the effects of different I/O scheduling schemes for paging on the operation of some scheduling algorithm, a different paging routine may be written which implements another scheme and this module may be used in place of the existing module. In addition to this the various parameters which combine to determine the behavior of the simulated system, such as the speeds of the I/O devices, average time between peripheral I/O requests or the arrival of new jobs to the system are accessible to the user on each run in order that he may change them to suit his needs. Details of the functions of these various parameters are given in appendix A. As for the realistic performance of the model,

data from the 370/165 batch processing system at MIT's Information Processing Center has been used as a yardstick to measure the performance of the model and ensure that it behaves in a manner approximating the operation of an actual system.

Data Bases

The model makes use of three major data bases: the Job Stream List, the System Event List and the System Clock. The Job Stream List contains an entry for each job currently in the system. The entries are implemented as PL/1 data structures linked together by pointers and have the form shown in figure 2-3. These figures provide a complete picture of the state of a modelled job at all times during its residence in the system. The Job Identifier Number, Job Type, Priority Level, Total Job Size, Working Set Size and Total CPU Time Required are static characteristics; they remain the same throughout the life of the job. These quantities are generated for each job by JBARVL when the job arrives at the system. They are discussed further in connection with that module. The fifth entry shown in the diagram is a pointer to the next description in the Job Stream List. The Memory Partition Size is assigned by the scheduler when the job is activated, and the Timeslice is

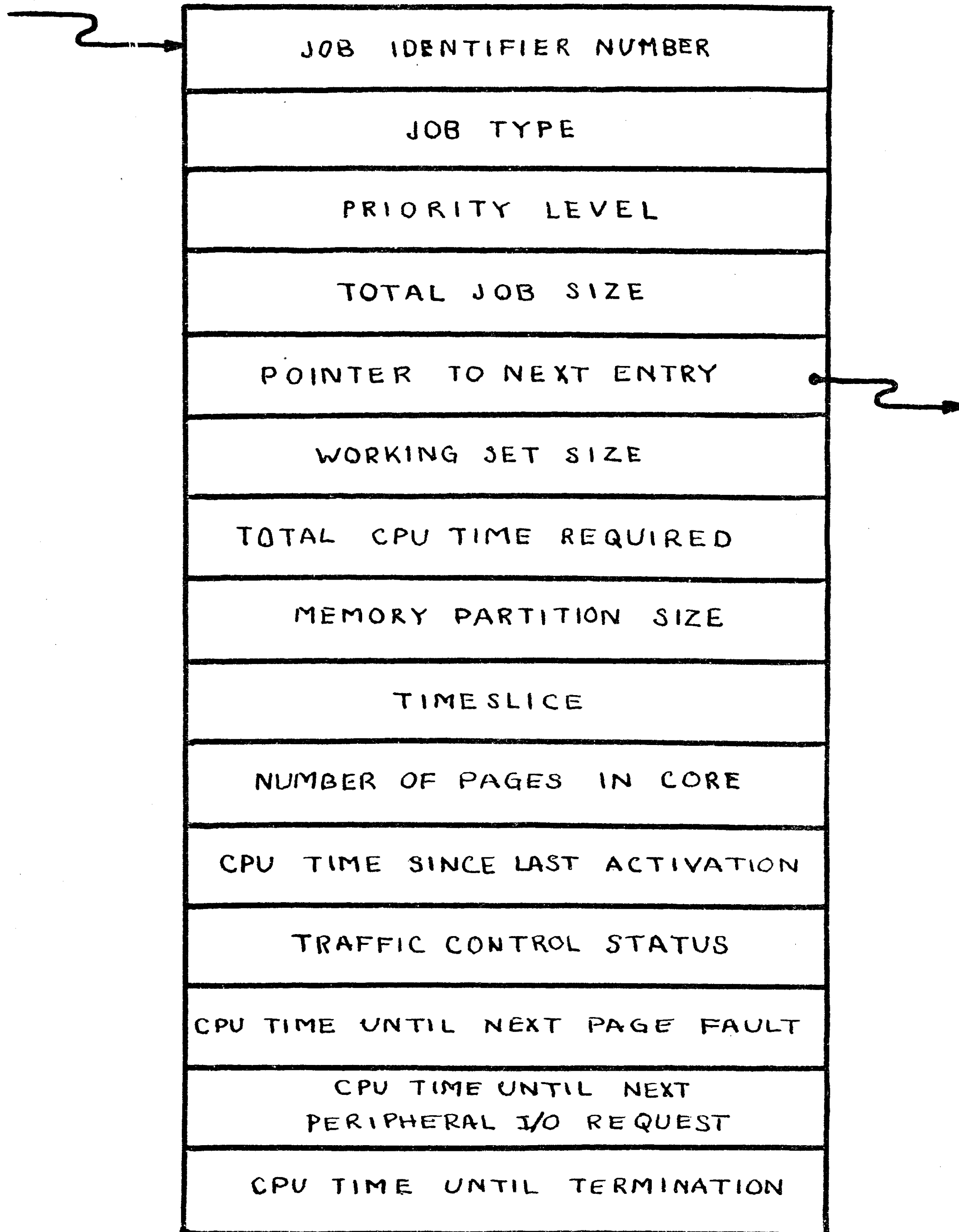


FIGURE 2-3

JOB STREAM LIST ENTRY

specified by the scheduler when he chooses the job to be processed. The Number of Pages in Core is a record of the number of pages of this job which are currently in main memory. This number is always less than or equal to the partition size assigned to the job. The CPU Time since Last Activation, also referred to as the Active Indicator, tells whether or not a job is presently assigned a partition in main memory. An inactive job has a value of minus one for this entry; one which is active has a value equal to the amount of processor time devoted to it since it was last activated. Traffic Control Status indicates whether the job is running, ready to be run, or blocked awaiting the completion of a service request. Note that it is possible for a job to be blocked awaiting more than one request at a time. In particular, it may issue a peripheral I/O request, causing it to enter the blocked state to wait for completion of that request, and may then be deactivated and reactivated, causing it to wait for its first page to be brought into core. Traffic Control Status takes into account the number of requests for which a job is waiting rather than just the fact that it is waiting in order to properly handle this situation. The Active Indicator and Traffic Control Status are needed in order to enable the model to check the operation of the scheduler to ensure that an ineligible job is not assigned to be processed.

CPU Time Until Next Page Fault and CPU Time Until Next Peripheral I/O Request record the processing time remaining for this job until its next page fault and next disk or tape I/O request, respectively. These figures are initialized with values generated by the appropriate modules (PGNXT and DSTPNXT) when the job is activated, and they are refreshed in the same manner whenever they go to zero. CPU Time Until Termination is initialized with the Total CPU Time Required value generated for this job, and when it reaches zero the job has finished processing and leaves the system. These last three entries are decremented each time the job is run by the amount of processing time it receives.

The based structure facility of PL/1 provides a convenient medium for implementing and maintaining this list. JBARVL allocates an entry for each job as it arrives and links it into the list, which is ordered by job number. A particular job description is accessed by searching through the list for a match with the corresponding job number, using the forward pointer in each entry to find the next element when a match is not found. When a job terminates, its description is deleted by adjusting the pointer in the element which precedes it in the list to point to the element following it and then freeing the

storage it occupies.

The System Event List is a record of all events which are to occur in the system at known times in the future, i.e. events which will occur at certain times regardless of the jobs which are chosen for processing in the intervening time. Such a list has been used with good results by MacDougall (28). In simulation terminology it constitutes a future events list for the model as described by Sussman (47). The events recorded in this list include the time at which the next job will arrive at the system for processing and the times at which all pending page requests and I/O requests will be satisfied. These events are assigned absolute times of occurrence rather than the length of elapsed time figures described above for entries in the Job Stream List. The form of the System Event List is shown in figure 2-4.

The events in this list are maintained in chronological order. As the time of occurrence of each of these events is determined an entry is allocated for it and linked into the list in the proper position to maintain the list in chronological order. As events occur the entries for them are freed after updating the pointer to the head of the list. Due to the chronological ordering of the list, the element to be deleted is always the first one in the list.

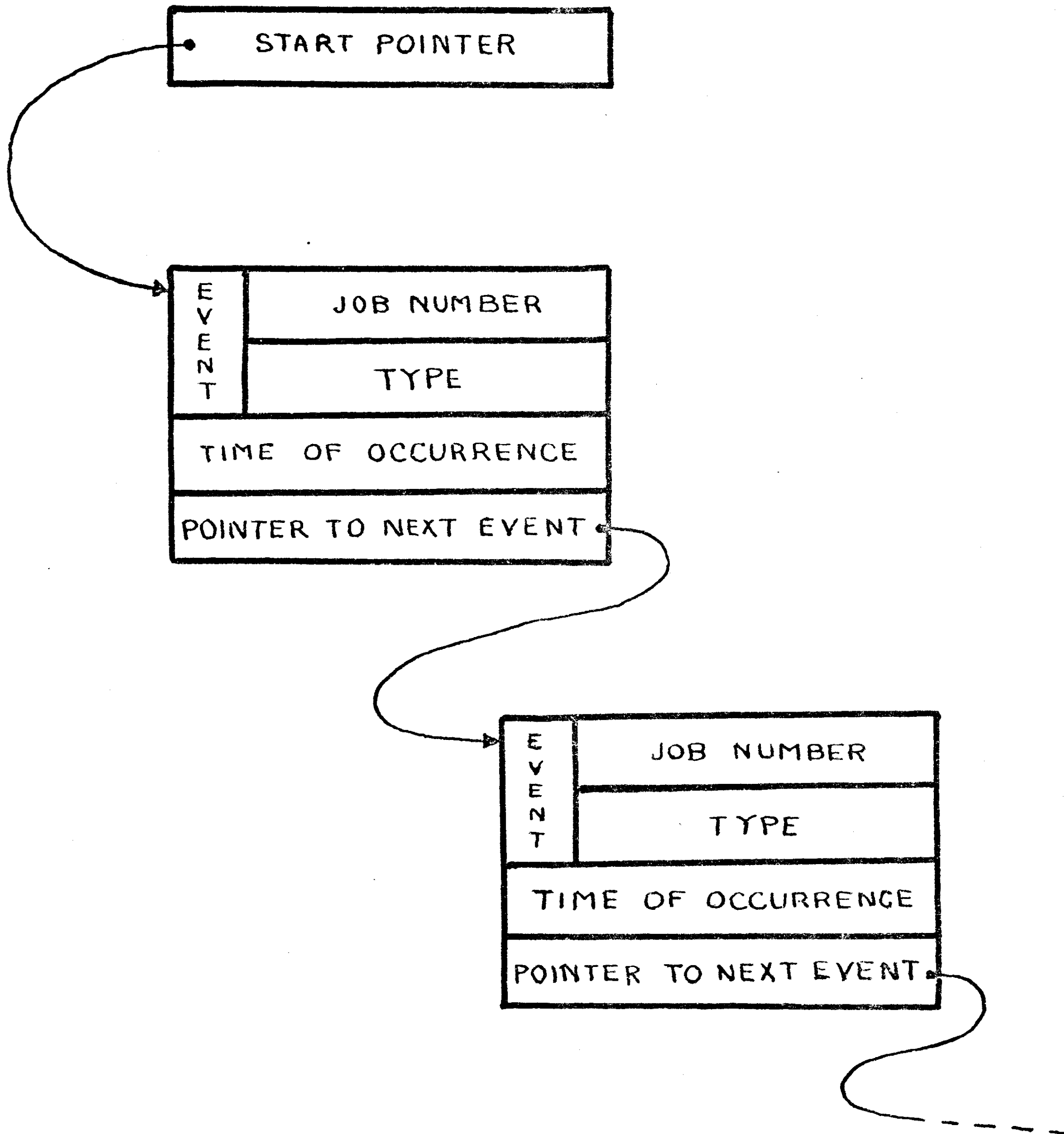


FIGURE 2-4

SYSTEM EVENT LIST

The *System Clock* is simply a variable that holds the current simulated time in the model throughout its operation. The clock is referenced in order to determine the length of time until the next event is to occur in the system, and it is updated by that value when the event occurs. Time moves forward in the model each time the value of the clock is increased.

Detailed Discussion of the Modules Comprising the Model

Supervisory Module (DRIVER)

The supervisory module defines the operation of the model since it is responsible for coordinating the activities of all segments of the model. Its functions in this regard include:

- providing an interface to the scheduler under investigation
- coordinating the other modules of the model
- maintaining the model's data bases
- simulating the flow of time in the modelled environment

The first of these functions involves two processes. The supervisory module passes to the scheduler the information on which it must base its decisions on jobs to be run,

activated and deactivated. Similarly, it receives the commands the scheduler issues in response to this information. The second function involves calling the various modules which comprise the model when their services are required. For instance, when a job issues a page request, the Supervisor Module first calls PGTIM to determine how long it will take for the requested page to be brought into main memory. It then calls PGNXT to determine the duration of time for which the issuing job will run once this page has been brought in before it again generates a page fault. The third function, that of maintaining the model's data bases, is an activity restricted primarily to the DRIVER routine. DRIVER can determine beforehand what information will be needed by a given module for the calculation it is to perform, and in general it accesses the model's data bases to determine that information and passes it to the module in the form of parameters. Similarly, the other modules communicate their results back to DRIVER to be entered in the appropriate data bases rather than entering the values into the data bases themselves. This centralizes the routines needed to maintain the data bases and avoids duplication of code needed to perform these tasks.

Time in the model is viewed in terms of the intervals between the various events occurring in the simulated

system. These events are of the following seven types:

- the arrival of a new job at the system
- the satisfaction of a page request issued by a previously running job
- the satisfaction of a peripheral I/O request issued by a previously running job
- timeslice runout by the currently running job
- a page fault incurred by the currently running job
- a peripheral I/O request issued by the currently running job
- termination of processing of the currently running job

A flowchart of the main loop of the Supervisory routine is shown in figure 2-5. This flowchart shows the way in which the simulation proceeds from one event to the next.

After each call to the scheduler DRIVER determines the event which is to occur next in the simulated system and then takes appropriate action to cause this event to occur. This action may involve calling various other modules of the system and/or making modifications to the entries of the Job Stream List or the System Event List. For instance, if the next event to occur is the incurring of a page fault by the running job, DRIVER calls PGTIM to generate a value for

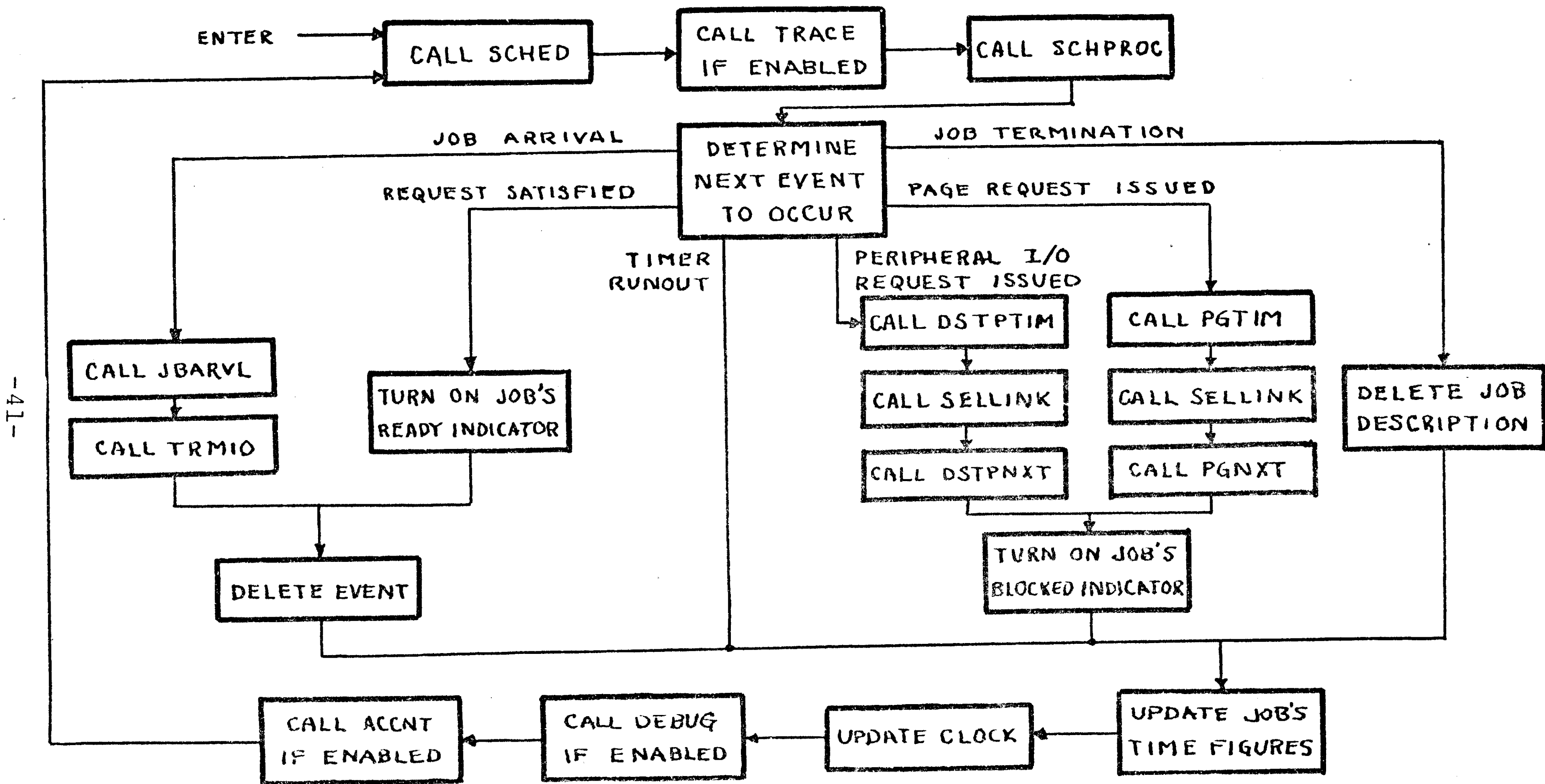


FIGURE 2-5

MAIN LOOP OF SUPERVISORY ROUTINE

the time at which the needed page will arrive in main memory. It calls SELLINK to create an entry in the System Event List to record an event for the completion of this page request and then calls PGNXT to generate a value for the time for which this job will run before it again generates a page fault. This figure is entered in the CPU Time until Next Page Fault entry in the description of the job in question in the Job Stream List. The Traffic Control Status entry in the job description is set to indicate that the job is now blocked.

As another example, if the next event is the completion of service of a previously issued I/O request (either for paging or peripheral I/O), DRIVER's only responsibilities are to note that the job has returned to the <u>ready</u> state by modifying its Traffic Control Status entry appropriately and to delete the entry corresponding to this event from the System Event List. Regardless of what event has occurred the Supervisor must decrement the pending time figures (CPU Time until Next Page Fault, CPU Time until Next Peripheral I/O Request, and CPU Time until Termination) for the job which has been running, thus registering the processing which has been done on it in this time interval. It must also update the System Clock to the time of occurrence of the event. Then the cycle is repeated, the scheduler being

informed of the latest event and making its choices of the job to be run next and any jobs to be activated or deactivated based on this new information. The output modules (TRACE, DEBUG and ACCNT) are called at the indicated points during each iteration of the Supervisor routine subject to the values of variables which control the space of simulated time over which they are enabled. These variables are described below in connection with each output module.

The flowchart in figure 2-5 shows only the steady-state operation of the model and does not include the initialization phase which is also part of the Supervisory Module. Initialization of the model on each run involves first calling JBARVL repeatedly to generate characteristics for the initial job mix. One of the characteristics generated for each job is its total size in pages. DRIVER assigns each of these jobs a partition in main memory, the size of which is arbitrarily chosen to be one-half of the total size of the job. When a job is generated which is too large to be assigned a partition from the remaining free memory the generation of jobs is stopped. The jobs which have been assigned partitions in main memory are treated as already having been partially processed; the characteristics describing them are generated as such by JBARVL. The procedure used for this is discussed further below in connection

with that module. The final job generated which can not be assigned space in main memory is assumed to be unprocessed. The rationale for this method of initiating the model's operation is that an initial job mix made up of jobs which have already been processed in varying degrees should produce a smaller startup transient in the behavior of the simulated system than other methods, such as starting with main memory completely empty or filled with totally unprocessed jobs. DRIVER also initializes the System Event List with the first known event to occur in the system, which is the time of arrival of the next job coming into the system for processing.

Scheduler (SCHED)

This module must perform all of the functions of a scheduling algorithm in an actual system. Its detailed form will not be specified here since a number of such routines may be run with the system and these different routines may use different algorithms to perform the tasks involved in scheduling. But the duties to be performed remain the same for each routine used for this purpose, as do the input and output parameters with which it is supplied. The general form of the scheduler is shown in figure 2-6.

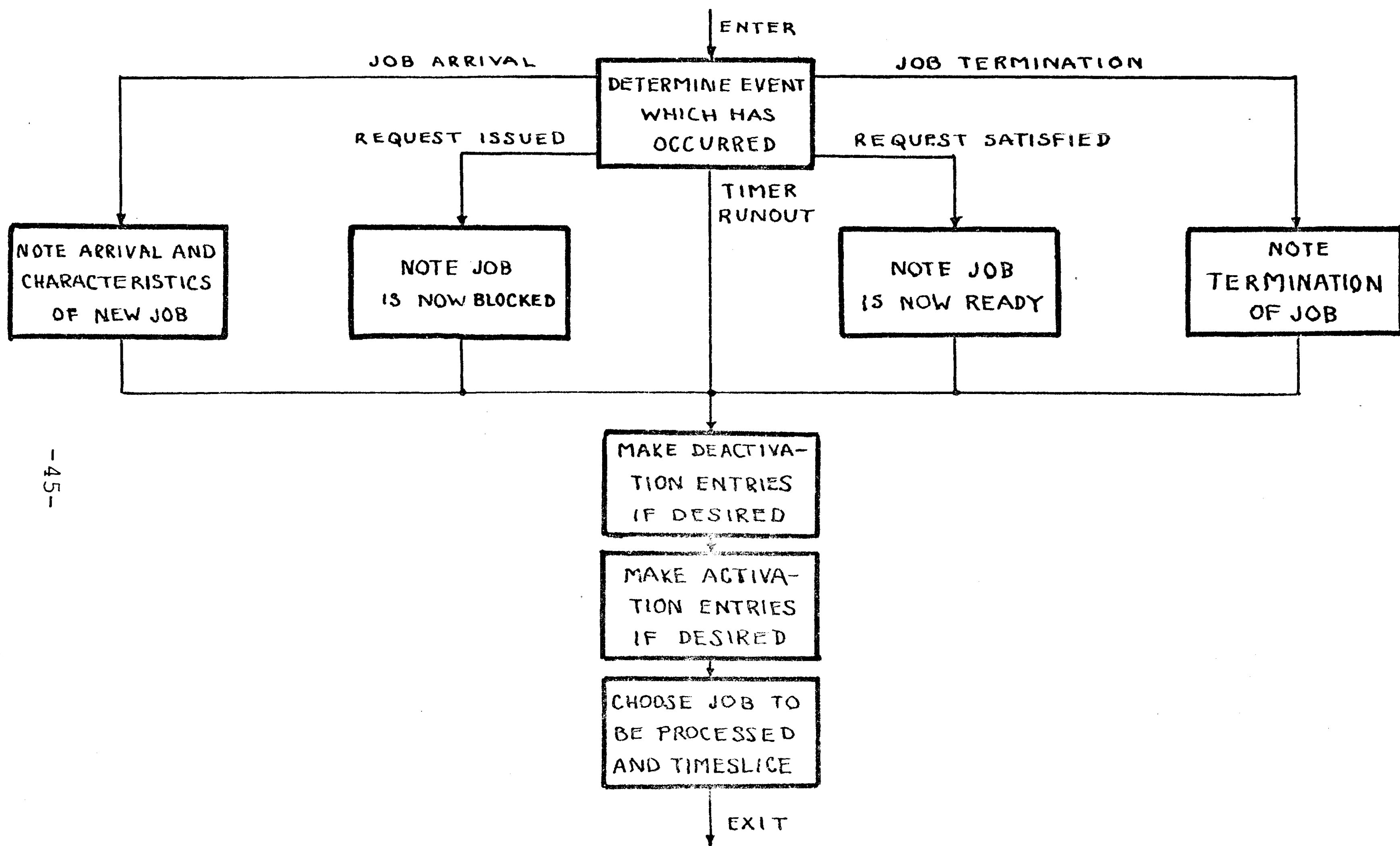


FIGURE 2-6

GENERAL FORM OF SCHEDULER

Its functions are as follows:

- choosing a job to be assigned to the processor each time the processor becomes free and assigning a quantum which limits the time for which that job may be run.
- preempting the current job (if desired) in favor of another which has just entered the system or has just returned to the <u>ready</u> state.
- deactivating (removing from main memory) and activating (assigning a partition in main memory) jobs, which includes assigning a maximum number of blocks of core (partition size) to each newly activated job.

The scheduler is called on each iteration of the DRIVER routine; i.e. whenever an event occurs in the simulated system. An event here is one of the seven types of events discussed above in connection with the supervisory routine. Each time the scheduler is called it must specify the job to be processed next. This job may be the same as the job currently being processed; the net effect in this case is that the current job continues to run without interruption since no time elapses in the simulated system during calls to the scheduler. The scheduler must assign a timeslice to

each job it selects to be run, limiting the length of continuous processing time which may be devoted to it. If desired, it may also activate or deactivate one or more jobs. For example, if a job terminates, the scheduler may activate another job to occupy the core freed by the old job. Or on any given call the scheduler may choose to order no activations or deactivations, but simply to do some internal bookkeeping, as may be the case if it is informed that a new job has entered the system but it does not wish to activate this job immediately. It may then simply make note of certain facts about the new job for future reference.

It is assumed in the model that jobs issuing peripheral I/O requests have buffer areas set aside to hold the data being brought into or written out of main memory. A job which is blocked for a peripheral I/O request is not prohibited from being deactivated. The buffer areas involved in the I/O request are assumed to be left in core until completion of the I/O request, and written out afterward if necessary. Similarly, a job which is blocked for a page fault may if desired be deactivated. This results in the new page being brought into core without a chance of being used, however, which is of questionable benefit in most cases.

There are several things any scheduling algorithm must

do in order to perform its various tasks. These include:

- defining a scheduling framework such as the ones shown in figures 1-1 and 1-2 and keeping track of which jobs are in which states under this framework at all times.
- defining a strategy to decide which eligible job should be chosen to be run next, how long it should be allowed to run, and, if desired, when to preempt a running job in favor of another job which becomes eligible.
- defining a strategy to determine under what conditions jobs should be activated and deactivated, and a method of choosing the particular job to be brought into or removed from core when activations or deactivations are to be performed.

In addition to these tasks a scheduler may wish to make note of additional information about the behavior of jobs in the simulated system. This information may include anything which might help the scheduler to predict the future behavior of jobs on the basis of past performance, such as the average length of time each job runs between page faults or other I/O requests.

All inputs to the scheduler are provided by DRIVER, and outputs from the scheduler are likewise returned to the supervisory module. The scheduler is not allowed to access the model's data bases, with the exception of the first five entries in the description of each job. The first four of these entries give the static characteristics of the jobs which provide data that the scheduler in an actual system would have knowledge of, such as the total size of the job and its priority level. The last entry accessible to the scheduler is the pointer linking each job description to the next description in the Job Stream List. All other data provided to the scheduler, such as the event which occurred most recently in the system, is provided via parameters passed by DRIVER. This is done for several reasons. First, it helps to make the scheduling routine less constrained by the structure of the model if it receives its information from another routine, as would be the case in an actual system, rather than reading it from certain global variables. Also, it prevents the scheduler from making modifications to the data bases and accessing information it should not have knowledge of, such as the length of time for which a given job will run before incurring its next page fault. The inputs it is given to work with include:

- a pointer to the initial entry in the Job Stream List, allowing the scheduler to access the static characteristics of each job in the system. (These characteristics include job type, priority level and total memory size along with the identification number of the job.)
- the event which has just occurred in the system and the job involved in this event. (If the event was a job termination, a page fault or peripheral I/O request issued or a timeslice runout the job involved is the current job; if it was a job arrival or the completion of a previous request it is another job in the system.)
- the present time in the model as recorded on the System Clock in the main routine. This information is of interest to the scheduler, for instance, when the scheduler wishes to allow the current job to continue to run governed by the timeslice originally assigned to it. It must be able to determine how long the job has already run so that its timeslice on this iteration may be reduced accordingly when it is reassigned to be processed.
- the total amount of main memory available to user programs. This data is used by the scheduler for

determining the amount of memory available for activating new jobs.

The scheduler may issue the following commands which are received by DRIVER and passed to SCHPROC for processing:

- activation commands, specifying the identification number of the job to be activated and the partition size to be assigned to that job.
- deactivation commands, specifying the identification number of the job to be deactivated.
- the job to be processed next, specified by identification number, and the timeslice to be assigned to that job.

The form of the data bases used by the scheduler to communicate its commands to the rest of the model are as follows. Activation and deactivation commands are passed via chain-linked lists, with one entry to describe each job to be brought into or removed from main memory on any given iteration. The form of these lists is shown in figure 2-7. A pointer variable corresponding to each of these chains is passed as a parameter to the scheduler on each call. If one or more activation commands are issued on a given call the pointer for the activation chain is set to point to the

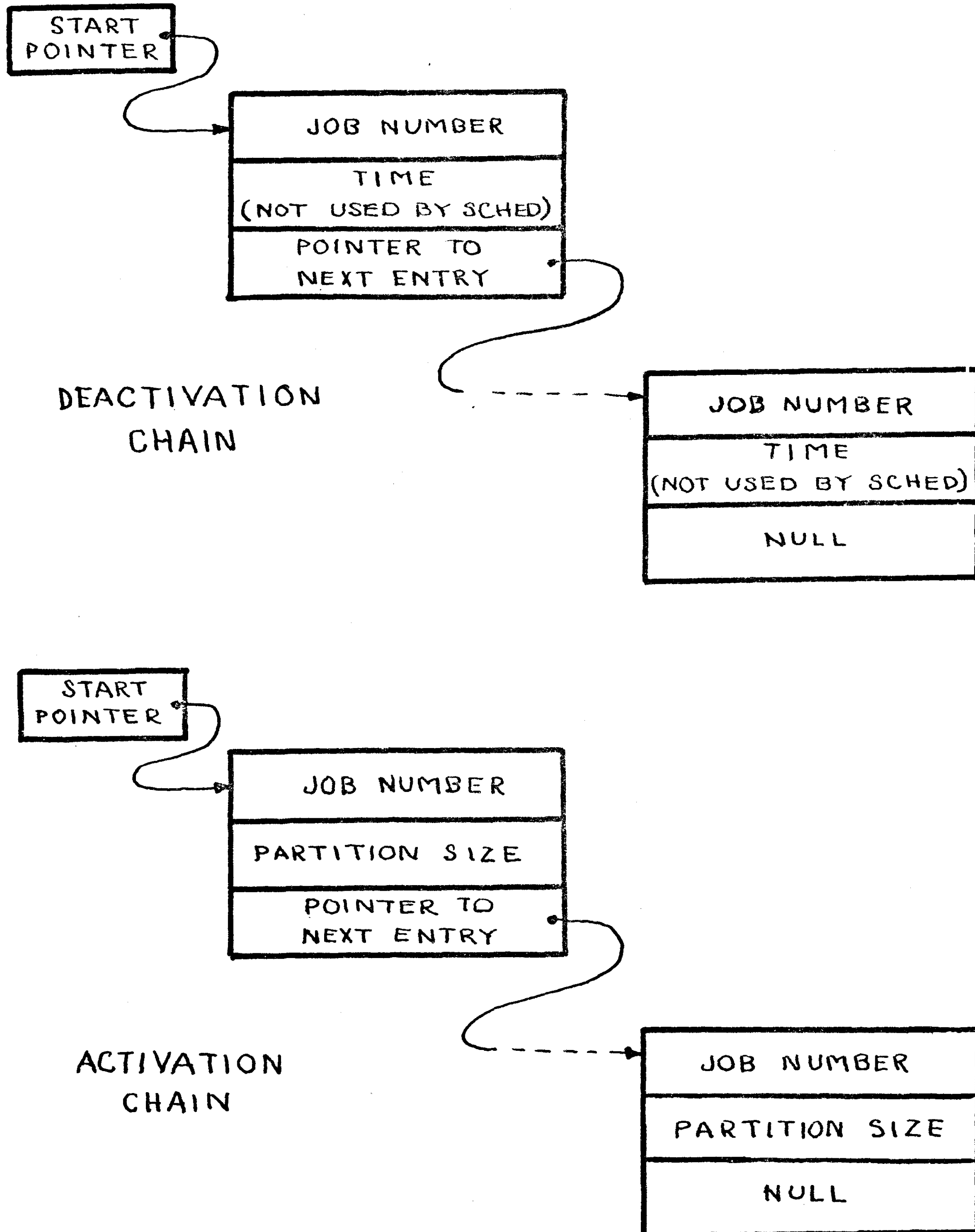


FIGURE 2-7

SCHEDULER DATA STRUCTURES

first entry in the activation chain; if not, it is given the value NULL. The pointer for the deactivation chain is treated in an analogous manner. Activation and deactivation commands are handled in this manner because the number of such commands issued by a scheduler on a given call is highly variable. It is dependent on the policy of the particular scheduler in regard to activating and deactivating jobs and on a number of other factors, such as the number of jobs in the system and the amount of user memory available. No apriori limit can reasonably be set on the number of these commands which may be issued by a scheduler at any one time, and thus a linked list where the number of entries may vary freely is the most appropriate method. The job chosen to be run next and the timeslice to be assigned to it, in contrast to this situation, are simple numbers, and are passed as individual parameters to be set by the scheduler on each run.

The form of the scheduler parameters and data bases discussed above is shown in the listings of the three schedulers given in appendix B. These listings show the declarations used to define the parameters passed to the scheduler and the precise form of the data structures used to reference the Job Stream List entries and to issue activation and deactivation commands. The schedulers shown

in this appendix are discussed in some detail in chapter 3.

Procedure to Process Scheduler Commands (SCHPROC)

This is the routine which checks the validity of the commands issued by the scheduler and carries them out on each iteration of the model. It is called by DRIVER on each iteration of the model immediately after the scheduler is called. Its first task is to process deactivation commands. This entails first locating the description of the job to be deactivated, a function performed by JSEARCH. When the description is found SCHPROC sets the job's Active Indicator to -1, indicating that it is no longer assigned space in main memory, sets its Number of Pages in Core entry to zero, and adds the amount of core assigned to this job (its partition size) to the total amount of free memory available in the system. If a description is not found for a job ordered deactivated (i.e., it is not in the system) or if the job is found to be already inactive an appropriate error message is printed out and the command is ignored.

When all deactivation commands issued by the scheduler on this iteration have been processed SCHPROC processes any activation commands. This involves calling JSEARCH to locate the description of the job to be activated and setting the job's Active Indicator to zero, signifying that this job is active but has not yet been processed on this activation. It also

enters the partition size assigned to this job by the scheduler in the activation command in its Job Stream List description and subtracts this amount of memory from the amount of free memory available in the system. Activating a job in the context of the model involves bringing its first page into core. SCHPROC calls PGTIM to generate a value for the length of time it will take to bring this page into memory, and then calls SELLINK to make an entry in the System Event List for the completion of this page request. A call to PGNXT yields a value for the length of time this job will run in its first page before generating another page fault, and a call to DSTPNXT produces a similar figure for the length of time the job will run before generating a peripheral I/O request. These two figures are entered in the appropriate entries of the job description. The job is now ready to be processed as soon as its first page arrives in main memory. As in the case of deactivations, an error message is produced if the scheduler issues an invalid command (i.e. if the indicated job is already active, if it is not in the system, or if the scheduler assigns it a partition size which is larger than the present amount of free memory or specifies a non-positive number of pages). The invalid activation command is ignored.

SCHPROC's final task is to assign the job chosen by the scheduler to be run next to the processor. This involves calling JSEARCH to locate the description of this job and then calling JCHECK to make sure that it is eligible to be run (i.e. it is both ready and active). If the chosen job is not eligible to be run or is not in the system, job zero is assigned to the processor as a default, i.e. the system remains idle. The timeslice assigned by the scheduler is then checked for validity (it must specify a positive time interval). If it is valid it is recorded in the description of the job to be run (either the chosen job or job zero); otherwise a default timeslice is used.

SCHPROC is responsible for freeing the storage occupied by the structures describing scheduler commands which have been processed. Descriptions of activation and deactivation commands are left intact throughout the iteration in which they are issued in order to make them available to DEBUG and ACCNT if these modules are called. SCHPROC maintains a pointer to the activation chain and the deactivation chain issued by the scheduler on a given iteration and then frees these structures on the next iteration when they are no longer needed.

Time Until Next Page Fault (PGNXT)

This module generates values for the length of time a particular job will run before generating its next page fault. The frequency with which a job generates page faults, or alternatively the length of time between page faults, sometimes called the page residence time, has been shown to depend primarily on the number of the job's pages which are already in core and the amount of CPU time it has received since it was last activated (48). The relationship between these quantities is shown in figure 2-8. An approximation to this curve has been used in generating times until the next page fault with good results in the SIM/61 simulation experiments (39, 44). The asymptote of the curve is the working set size of the particular job in question. Working set size is a term originated by Denning (49) to refer to the set of a job's pages which must be in core in order that it may execute without an intolerable number of page faults. Working set size in the context of the model is more broadly interpreted to mean the set of pages of a job which are actually used on a given execution. The curve is roughly exponential, and in the model the curve is approximated as exponential using the base of the natural logarithms, e, as the base of the curve. The exponent to be used differs among the

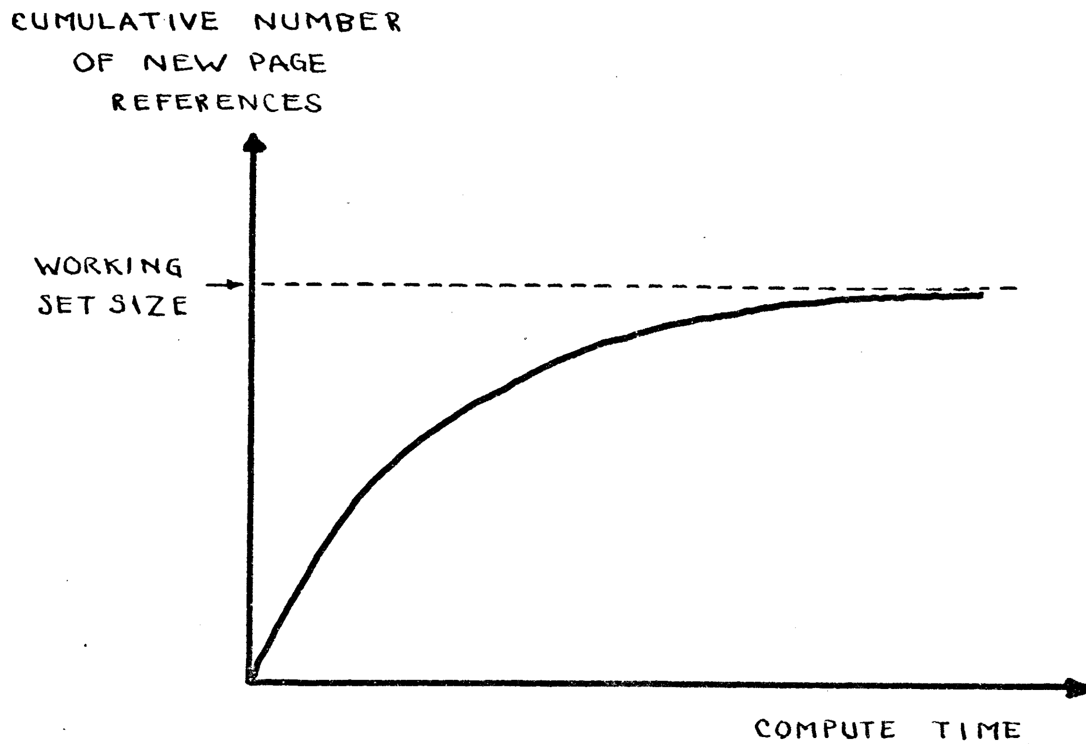


FIGURE 2-8
CURVE GOVERNING TIME BETWEEN
PAGE FAULTS

different types of jobs, allowing paging behavior of the different job types to be individually specified.

PGNXT operates by using the working set size of a given job and the paging exponent corresponding to its job type to determine the exact curve to be used. It then takes the number of pages which will be in core for this job after a given page fault and inverts the curve to find the corresponding value of compute time, which is the time at which the fault for this page is to occur. An analogous procedure is followed to determine the time of the succeeding page fault, and the difference between these two values is the time between the corresponding page faults, i.e. the time until the next page fault will occur. This process may be visualized more clearly with the aid of figure 2-9. This figure illustrates the computations performed by PGNXT when a page fault is incurred causing a job's nth page to be brought into core. The paging curve is inverted to determine the amount of compute time t_n which was theoretically received by the job before it incurred this page fault. A similar inversion is performed to determine the length of compute time t_{n+1} elapsing before the n+1st page fault. The difference $t_{n+1} - t_n$ is then the compute time elapsing between the nth and n+1st page faults for this job.

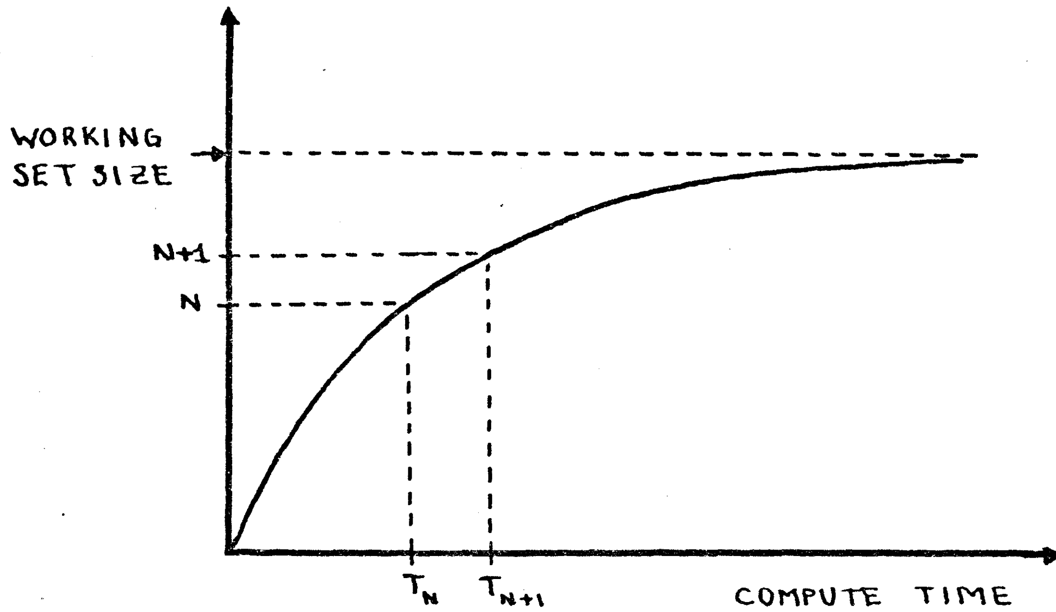


FIGURE 2-9
EXAMPLE OF GENERATION OF
TIME BETWEEN PAGE FAULTS

PGNXT must also take into account the partition size of the job which has incurred the page fault and the number of its pages which are already in core. If a job already has its full partition size in core but not its full working set, the time between its page faults remains the time between the fault for the last page in the partition and the fault for the next page. This is so because once a job has its full partition size in core each new page replaces one of the pages already in core for this job, leaving the likelihood of another page fault at any given time in the future the same as it was before. The working set size of a modelled job is the total number of pages it will reference during its execution. Thus if the partition size assigned to a job is large enough to accomodate its entire working set then as soon as all the pages in the working set are in core no more page faults will occur for this job. This is ensured by setting the CPU Time between Page Faults entry for this job in its description in the Job Stream List to a sufficiently large value that the job will terminate before incurring any more page faults.

PGNXT is called by DRIVER whenever a running job issues a page fault in order to determine how long the job will run before generating its next page fault. It is also called by SCHPROC each time a job is activated to determine the

processing time until the job will issue its first page request, and by JBARVL in generating the initial job load to provide a maximum value for the time until the next page fault for the partially processed jobs.

Time to Service a Page Request (PGTIM)

This routine determines the length of time needed to bring a page into main memory. This time interval is dependent upon the number and characteristics of the I/O devices used for paging and on the number of requests already queued for these devices. The I/O device characteristics include the average access time, which is the average time needed to locate a page on the device, and transmission rate, which is the speed at which information can be transferred from the device to main memory once it has been located. A fixed page size is assumed in the model on any given run; thus the transmission time is the same for all page requests. Access time for any given request is assumed to be normally distributed about the average access time of the device. Page requests are queued at each device in a first-in-first-out manner, thus generating an essentially random sequence of requests to each device, so a normally distributed access time seems a reasonable assumption. All device characteristics are parameters which may be set by the user on

each run.

PGTIM operates by first calling RANDOM to generate a random number which is used to determine on which paging device (if there is more than one) the required page is to be found. PGTIM then looks up the time at which all present requests queued for this device, if any, will be completed. A value for the access time needed to locate the page is generated, and the time needed to transmit it is determined. The access time, transmission time and the time at which the device will be free (which is the current time in the model if no requests are presently queued for the device) are then summed to give the time at which this page request will be completed. The time at which the paging device involved in this request will be free is set to this value for reference in regard to future page requests, and the result is returned to the calling procedure.

PGTIM is called by DRIVER each time a page fault occurs in order to determine the length of time needed to service the request. It is also called by SCHPROC whenever a job is to be activated to determine how long it will take to bring in the first page of the newly activated job.

Time Between Peripheral I/O Requests (DSTPNXT)

This module determines the time between peripheral

I/O requests issued by each job. Values for the time a job will run before it issues its next disk or tape I/O request are drawn from a normal distribution. The mean value of this distribution is specified separately for each job type, making it possible to represent different peripheral I/O behavior for jobs of different types. No distinction is made between requests to disk and to tape in generating interrequest times. It is assumed that any job issuing a tape request has the required tape drive assigned to it. Since the model is concerned only with the aggregate behavior of the job stream rather than dealing with individually representative jobs it is not necessary to specify which jobs have control of tape drives and may therefore do tape I/O. The only consideration of interest is the average rate of occurrence of requests for peripheral I/O issued by the job stream as a whole and the relative frequency of disk requests to tape requests.

DSTPNXT is called by DRIVER each time a job issues a previously scheduled disk or tape I/O request to generate a value for the time it will run before issuing another request for peripheral I/O. It is also called by SCHPROC whenever a job is activated in order to determine how long the job will run before generating its first peripheral I/O request after activation, and by JBARVL in generating the

initial job load to determine the maximum time until the next disk or tape I/O request for each job.

Time to Service a Peripheral I/O Request (DSTPTIM)

Although peripheral I/O requests are issued by jobs without specification of whether they involve disk or tape, the service times involved in requests to the different types of devices must be determined in different ways. Thus the first thing DSTPTIM must do when it is called is to decide whether the request which has just been issued involved a disk unit or a tape drive. This decision is made by generating a random number which is compared with a parameter which specifies the relative frequency of disk requests among all peripheral I/O requests issued. The type of the request as determined in this manner then causes one of two subprocedures to be called to determine the time needed to service the request.

The subprocedure which handles disk requests operates very similarly to the way in which PGTIM treats page requests. The exact device involved is determined through the use of a random number, and the time at which that device will be free (which may be the present simulated time) is added to a value generated for access time and the time needed for transmission to yield the time at which

the request will be satisfied. Again as in PGTIM this value is noted as the time at which this device will next be free.

The subprocedure which handles tape requests operates somewhat differently. As mentioned above, it is assumed that the job which issued the request has a tape drive dedicated to it to service the request. Thus the tape drive needed must already be free, and no tape drive wait enters into the calculation of the total time needed to service the request. Since tapes operate in a serial manner, the technique used to find values for tape service times is to first generate a value for the number of records which must be passed over (forward or backward) in order to reach the desired record for this operation. Given the transport speed and the time needed to come up to speed (read/write access time) characterizing the tape drive being used and the average record length it is then a simple matter to determine the time needed to reach the desired record. The length of the record to be read or written and the transmission rate of the device combine to determine the transmission time needed. These two values are summed to provide a value for the total service time needed. DSTPTIM is called by the same routines as is DSTPNXT to provide values for the service times needed for each disk or tape I/O request.

Job Interarrival Module (TRMIO)

This module differs somewhat from the modules which handle disk and tape I/O requests and page faults in that the requests initiating new jobs are not associated with jobs already in the system. Instead, in the case of a time-sharing system jobs are generated by communications from user terminals, while in a batch-processing environment they arrive via input devices such as card readers or remote terminals.

One of the problems in simulating processes such as the arrival of new jobs to the system is the probabilistic assumptions made about the population which generates them. In the case of a time-sharing system, if we consider the user population to be finite and assume that any user has at most one request pending at any time, then we must take into account the number of jobs already in the system as a decrease in the total user population generating further requests. Denning has shown (45) that in a large time-sharing system this need not be regarded as being the case, but that one may assume an infinite user population for purposes of predicting interarrival times for terminal messages. A large batch system with a sizable user community is analogous to this situation. The assumption of an infinite user population is therefore made in the model.

Several studies (50, 51) have been done on interarrival statistics for terminal communications, but they have concentrated primarily on the detailed interactions of a single user with a computer system rather than on the activity of the user population as a whole. However, Coffman and Wood (50) have found that the assumption of the independence of the arrivals of terminal communications is borne out in studies of actual systems, both for a single user and for the user population as a whole. Considering this independence and the infinite user population we have to draw from, the arrival process may be assumed to be Poisson in nature. Consideration of the user population of a large batch system leads to an analogous assumption for batch systems. The arrival process is therefore treated as Poisson in the model.

The generation of the time between arrivals of jobs to the system is performed by drawing a value from an exponential distribution. This distribution is conditioned on the average arrival rate of jobs to the system which is a parameter of the model. On any given call to TRMIO the value returned is the absolute simulated time at which the next arrival is to occur; this quantity is found as the sum of the value drawn from the exponential distribution and the time in the model when the call is made.

TRMIO is called by DRIVER during initialization of the model to generate the time at which the first job not in the initial job load will arrive at the system. Thereafter it is called each time a previously scheduled arrival occurs to determine the time of the next arrival.

Generation of Job Characteristics (JBARVL)

This module generates the characteristics which describe each job as it arrives at the system. Jobs are described by a set of five static characteristics: job type, total job size, priority level, working set size and total processing time required. A maximum of six different job types may be used on any model run. As an example, the job types used by Scherr (42) in his model of CTSS were:

- File Manipulation
- Program Input and Editing
- Program Running and Debugging
- Program Compilation and Assembly
- Miscellaneous

Jobs of different types customarily perform different kinds of tasks and place different average demands upon the system.

Interactive jobs will be considered here as individual requests only rather than as sequences of related tasks, as

discussed above in connection with the Job Interarrival module (TRMIO). Neilsen (30) and others have discussed schemes whereby a time-sharing task load is modelled by generating jobs which consist of a sequence of identical interactions repeated some number of times. This method was shown to give good results, but it involves more detail than is needed here. Since we are concerned only with the overall behavior of the simulated system over a significant period of time, whether a task is represented by a sequence of repeated commands from a specified user or by a number of such commands from undesignated users interspersed among other requests does not affect the overall results. A scheme more like that used in RCA's SIM/61 system (39) is used in the model described here. Jobs are generated individually according to the probability of occurrence of each job type, which may be entered as a parameter. This method is quite general, is applicable to both batch and time-sharing environments, and can be tailored to approximate different job mixes on different runs of the model. Priority level is generated separately according to another probability distribution.

The total size, working set size and total processor time required for any job are interrelated quantities. Working set size as used in the model denotes the actual

number of pages required during the execution of a given job. This is some fraction of the total number of pages associated with this job, since not all of a job's pages are necessarily used on any given execution. For each job type an average working set size and a standard deviation for this size are specified as parameters to the model. A value for the working set size of any given job is generated from a normal distribution conditioned on the mean working set size associated with the type of the job. The total size of the job is then found by multiplying the mean working set size by a factor which is also a parameter of the model and again drawing a value from a normal distribution. The total CPU time required by this job is generated using the same curve which will be used for finding the time between page faults for the job. This curve is described above in connection with the paging module PGNXT. The job will terminate at some time after its final page is brought into core, assuming that sufficient space has been assigned to the job to allow its entire working set to be in core at once. Using this assumption, an approximation is used to generate a value for the total run time from the paging curve.

In general the remaining entries of the description for each job are initialized to zero, with the exception of the

Active Indicator (CPU Time since Last Activation) and CPU Time until Termination. The Active Indicator is set to -1, indicating that the job is inactive and CPU Time until Termination is initialized with the total CPU time requirement for the job. In the case of the initial job load, however, JBARVL must generate jobs with characteristics indicating that they have already been partially processed. This is done by generating a job description in the manner described above and then modifying certain entries. These entries include Partition Size, Number of Pages in Core, Active Time, and the entries giving the times remaining until the job's next page fault and next peripheral I/O request and time until termination. The partition size of a job is normally assigned by the scheduler when the job is activated, but in this case we wish the job to be already active, so JBARVL arbitrarily assigns each job in the initial job load a partition size equal to one-half of its total size. Number of Pages in Core and CPU Time since Last Activation are interrelated, since the job must have run for a length of time appropriate for it to have issued page faults to bring in the number of pages it has in core. A value for the number of pages in core for each job is drawn from a normal distribution centered on one-half of the partition size assigned to the job, and subject to the constraints

that the value generated must be less than or equal to the partition size and also less than or equal to the total working set size of the job. A value for CPU Time since Last Activation is then generated as a random value uniformly distributed between the time at which the last page fault for the pages in core for this job would have been issued and the time at which the next page fault is to occur. These times may be found from the paging curve for this job as described in the discussion of PGNXT above. The CPU Time until Termination for this job is then simply its total required time minus the time for which it has already been active. Values for CPU time until next Page Fault and CPU Time until next Peripheral I/O Request are found by calling PGNXT and DSTPNXT respectively to yield maximum values for the corresponding intervals and then choosing a random value evenly distributed between zero and this maximum time in each case.

A number of possible inaccuracies are inherent in this method of generating an initial job load. All jobs initially in core are treated as having been activated at some time in the past and having been allowed to remain in core thereafter; i.e. no deactivations have been performed on these jobs. The job which is generated last in the initialization calls and which is not active has not been processed at all, and there are no other jobs waiting for service. This is not a

particularly likely state in which to find the system during the course of its operation, except perhaps in the case of a scheduler which does no deactivations. Also, all jobs in the initial load are ready to be run, and this is also an unlikely event in normal operation. However, this approach was felt to be more like the conditions found under normal operation than most other feasible initial loads, and the overhead involved in generating the initial jobs is little more than that involved in generating normal jobs.

Routine to Produce Debugging Information (DEBUG)

This module is called by the supervisor routine on each iteration of the model whenever the value of the System Clock lies between an upper and a lower bound which are set via parameters. Each time it is called, DEBUG prints out three types of information. First, it produces a record of the commands issued by the scheduler on the most recent iteration of the model. This includes the identification numbers of all jobs to be deactivated, the identification numbers of all jobs to be activated together with the partition sizes to be assigned to them, and the number of the job to be processed next together with the timeslice to be assigned to it. Then it produces a dump of the System Event List, which records all the events scheduled to occur at

specified times in the future in the simulated system together with the job involved in each event and the time at which it is to occur. Lastly, it prints out a listing of all the job descriptions presently in the Job Stream List, showing the complete status of each job in the system. Finally, it prints out the value of the System Clock, which is the present time in the simulated system. This information provides a clear and complete picture of the state of the system and is quite useful in debugging the model should problems occur.

The parameters which determine whether DEBUG will be called on any given iteration allow it to be enabled for all or for only a portion of a run. This saves on print costs, since a sizable amount of output is produced each time the routine is called, and this scheme facilitates calling it only in the time interval for which detailed information is needed. An example of the output produced by DEBUG is shown in appendix C.

Procedure to Output Trace Listing (TRACE)

This procedure is called by DRIVER once during each model iteration when the value of the System Clock is between an upper and a lower bound specified for the trace routine. These limits may be set via parameters as

described above for DEBUG. Each time it is called TRACE prints out the most recent event to have occurred in the system and the job involved in that event (e.g. Job #6 has incurred a page fault). It also provides a listing of any activation and deactivation commands issued by the scheduler after it was informed of this event, together with the number of the job it has selected to be run next. The output provided comprises a brief summary of the events occurring in the simulated system in the course of a run. If a summary of only part of a run is desired this is easily obtainable via appropriate settings of the limit parameters. Sample trace output is shown in appendix C.

Procedure to Produce Summary Information (ACCNT)

This routine collects statistics on the events occurring in the simulated system in the course of a run and processes them to produce figures describing the performance of the scheduler being used and the behavior of the simulated jobs running under that scheduler. It is called by DRIVER once during each iteration of the model subject to a parameter setting which specifies the minimum simulated time at which ACCNT is to be called. Setting this parameter to zero causes ACCNT to be called on each iteration of the model;

giving it a larger value than the total run time specified for a particular run ensures that it will not be invoked at all on that run. Setting it to some intermediate value causes ACCNT to be invoked only during the latter portion of the run, thus producing a final summary which considers only this portion of the run. This may be desirable in order to minimize the effects of initial transients in the operation of the simulated system on the figures compiled, thus yielding a more realistic picture of the steady-state performance of the system being simulated. On each call during iteration of the model ACCNT collects data on various aspects of the operation of the system. In addition, if it is enabled, it is called a final time after the model run is over to process these figures and print out the results.

The figures produced by ACCNT fall into four basic categories. First, it produces overall figures describing the average state of the simulated system. These figures include the average number of jobs in the system and in main memory, the average amount of main memory assigned to jobs, and the average number of pages actually in core. The percentage of simulated time spent idle by the CPU is also shown. Second, it provides figures describing the characteristics of the jobs generated on this run. This includes average total size and working set size, average CPU time

requirement, and type and priority level distributions. The third group of figures gives information on the actions of the scheduler used on a given run. This includes the number of activations and deactivations performed and the average timeslice assigned to jobs to be processed. The last group of statistics describes the behavior of the jobs in the system operating under this scheduler. This includes the average occupancies of running, ready and blocked states, total numbers of page faults and peripheral I/O requests generated, average wait time for each type of request, and total number of timeslice runouts. It also gives the number of jobs which terminated in the course of the run and their average turnaround times, including breakdowns of turnaround time by job type and priority level. Taken as a whole, these figures give a fairly comprehensive picture of the overall performance of the system simulated on any given run. As in the case of DEBUG and TRACE, an example of the output produced by ACCNT is provided in appendix C.

The jobs considered by ACCNT in compiling statistics on job and scheduler behavior include only those whose processing is initiated after the beginning of the accounting scan. The rationale behind this is as follows. Jobs which are active before this time have already been partially processed. When the run ends a number of jobs

will be left in the system only partially processed. If the jobs considered in compiling statistics on the number of jobs processed include only the jobs which are activated after the beginning of the scan then the processing done on the jobs which are initially active and which are not counted after the beginning of the scan should on the average balance off the work not yet done on the jobs left unfinished when the run terminates. This should yield a fairly reasonable count of the jobs actually processed by the system during the course of the run. On runs of short duration when few new jobs arrive at the system this may lead to misleadingly small figures. On longer runs, however, this method produces fairly representative results.

Procedure to Add an Entry to the System Event List (SELLINK)

This is the procedure which is used each time an event is to be added to the System Event List. It is called by the supervisor routine each time the time of occurrence of a future event is determined. Whenever a job incurs a page fault or issues a peripheral I/O request an event must be set up for the completion of that request, and whenever a job arrives at the system an event must be set up for the arrival of the next job. In these situations DRIVER calls SELLINK, passing it a number indicating the kind of event

which is to occur, the identification number of the job involved in the event, and the time at which the event is to occur. SELLINK then allocates an entry for this event, sets the variables in that entry to the appropriate values, and links it into the existing System Event List chain in ascending chronological order.

Procedure to Find a Job Description (JSEARCH)

JSEARCH is called by DRIVER and SCHPROC to locate the description of a given job in the Job Stream List. It takes as an argument the identification number of the job to be located. It returns a pointer which points to the desired description if the job is found in the list, or has a value of NULL if the indicated job is not in the list (i.e. is not presently in the system).

Procedure to Check Eligibility of a Job (JCHECK)

This procedure is called by SCHPROC to determine whether or not a given job is eligible to be run, i.e. whether it is both ready and active. JCHECK takes as an argument a pointer to the description of the job in question and returns the same pointer if the job is eligible, or NULL if it is not.

Procedure to Generate Random Numbers (RANDOM)

This procedure produces values evenly distributed between zero and one via the multiplicative congruential method (52). It is called by a number of the other routines of the model whenever a random number is needed for the generation of a sample value. It generates a repeatable sequence of values on each run, beginning with the same seed and using each generated value as the seed for the value returned on the next call.

Procedure to Generate Normally Distributed Values (NORMAL)

This procedure is called by the other routines of the model whenever a value drawn from a normal distribution is needed. It takes as arguments the mean and standard deviation of the desired normal distribution, along with a third parameter which governs the number of iterations to be used in producing the sample value. It returns an integer value drawn from the indicated distribution by means of the Central Limit Approach (52).

CHAPTER THREE

SOME EXPERIMENTAL RESULTS OBTAINED FROM THE MODEL

As described in chapter two, the model under discussion includes a large number of parameters which may be modified to approximate different sets of conditions existing in actual or hypothetical systems. These parameters are in many cases highly interrelated, and results from model runs with different parameter values must be compared with care. As an illustration of the kinds of results obtainable with the model and their relation to behavior observed in actual systems, a set of nine model runs was performed. These runs involved systematic variation of the parameter which governs the total amount of main memory available to user programs. Three different schedulers were used, each of them being run with three different values for user memory size. All other parameters were held constant, with the exception of the average interarrival rate of the jobs coming into the system for processing. This parameter was varied as needed in an attempt to maintain a sufficient number of newly arrived jobs to keep the simulated systems in a saturated state. Cost constraints prohibited its being set to some arbitrarily high value which could have been held constant.

Parameter Values Describing the Simulated System

In considering the results obtained from the test runs it is first important to discuss the values assigned to the various parameters of the model for these runs, including those which were held constant as well as those which were varied. The settings of these parameters determine the characteristics of the system being simulated, and they should represent a system similar to those found in practice if the model runs are to yield results approximating those found in actual systems. The parameter settings used in the test runs were based on figures describing the 370/165 batch processing system at MIT's Information Processing Center in September, 1972 (53, 54, 55). This system does not use paging but runs under OS/MVT, a multiprogramming system where the number of tasks in main memory is variable. Thus the characteristics of the paging devices in the modelled systems and the paging behavior of the simulated job streams could not be based directly on actual figures from this system. They were chosen to be similar to those found in systems which do use paging. The remainder of the parameters are based directly on the IPC system.

The parameters may be broken down into two major classes: those which describe the physical equipment available in the simulated system, and those pertaining to the job stream processed by the system.

Parameters Governing System Configuration

The first class includes parameters governing the number and characteristics of the I/O devices present in the simulated system. These variables were set to reflect the following configuration:

- twenty disk drives, including
  - twelve disk drives having the characteristics
    - average access time = 75 milliseconds
    - transfer rate - 312,000 bytes/second

  (IBM 2314 disks)
  - eight disk drives having the characteristics
    - average access time = 30 milliseconds
    - transfer rate = 806,000 bytes/second

  (IBM 3330 disks)

  All disks were assumed to have the same frequency of usage.
- an unspecified number of tape drives having the characteristics
  - read/write access time = 4 milliseconds
  - transfer rate = 160,000 bytes/second

  (IBM 2820 Model 4 tape drives)

  The model assumes that any job issuing a tape request has a dedicated tape drive available to service that request; no effort is made to limit or monitor the

number of tape drives in use at any time.

- two drums having the characteristics
  - average access time = 4 milliseconds,
  - transfer rate = 1200,000 bytes/second

  (IBM 2301 drums)

The drums were used exclusively as paging devices.

In addition to the above parameters which describe the operating characteristics of the I/O devices there are several parameters which specify factors affecting the usage of these devices. These parameters had values as shown below:

- pagesize = 4096 bytes
- disk record size = 1000 bytes
- tape record size = 800 bytes

Pagesize is in general a constant; disk and tape record sizes are not constant in an actual system but may be approximated as constant at their average values; this is the approach used here.

- ratio of the number of disk requests to the total number of peripheral I/O requests (disk and tape) issued = .821

Peripheral I/O requests are generated in the model without specification of the type of device they address. When such a request is issued by a simulated

job it is necessary to determine the type of device involved since the service times are computed in different ways for the different devices. This ratio is used to determine whether a given request involves disk or tape.

These figures correspond in general to the configuration and average usage pattern of the IPC system in September, 1972 and are representative of a number of large-scale batch installations.

Parameters Governing Job Stream Characteristics

The second class of model parameters deals with the characteristics of the jobs which are to be serviced by the simulated system. The model provides for up to six different types of jobs, each type having a separate set of characteristics describing it. In the experiments to be described here only one job type was used. Detailed data needed to break the jobs down into several classes (e.g. I/O bound, CPU bound, etc.) was not available for the IPC system. This single job type may be viewed as representing an average of the characteristics of the aggregate job stream serviced by the IPC system. There is considerable variation in the characteristics of the individual jobs of this single type produced by the model due to the probabilistic methods used

to generate them.

Each job was assigned one of three priority levels according to the following distribution:

level 1 - 7.61% (high priority)
level 2 - 37.49% (medium priority)
level 3 - 54.90% (low priority)

These numbers correspond to percent usages of the three major priority levels on the IPC system.

The mean time between peripheral I/O requests for the simulated system was 16 milliseconds. The exponent used for the paging curve (the curve used to determine the time until the next page fault) was .0003. This number does not have any direct physical interpretation. It was chosen by experiment to determine a value which resulted in reasonable paging behavior.

Job size is generated in two steps, each one governed by a different parameter. First the working set size is determined, which in this context is the total number of pages which will actually be used by the job during its execution. The mean value for working set size for the jobs generated was twenty-five pages. Total job size (all pages associated with a job, regardless of whether or not they are used on a given run) is then generated using a mean

value determined by multiplying the mean working set size by a constant factor. For these runs this multiplier had the value two. In other words, the simulated jobs used half of all their pages on the average during execution. As mentioned above, the IPC system is not a paging system, so this figure is again an estimate of realistic behavior. Total size is used only as an estimate to pass to the scheduler; working set size is the size measure used by the model in simulating job behavior. The average total size for jobs submitted to the IPC system was about 200K bytes, or fifty 4096-byte pages. Thus the modelled jobs resemble very closely the actual jobs in total size.

The total CPU time required by each job is determined by an extrapolation on the paging curve. (A job terminates sometime after its last page has been brought into core.) Thus a job's CPU time requirement is affected by the exponent of the paging curve. It is also affected by another parameter which again has no direct physical interpretation but which governs the length of time for which a job will run after its last page has been brought into core. In these runs this parameter had a value of 1.5, again determined through experimentation, which yielded CPU time requirements averaging 408.3 milliseconds. The IPC data indicate that an average job on that system runs for .63

minutes, or about ninety-three times as long as this. This disparity is the result of a tradeoff between two opposing factors. Since simulation runs using the model described here are quite expensive, it was necessary to make them fairly short. At the same time, in order to obtain meaningful figures on turnaround time and throughput it was desirable to have a fairly large number of jobs terminating in the course of each run. A compromise was reached by making the jobs quite short and running each test case for thirty simulated seconds. The behavior of the simulated jobs in respects other than CPU time requirements was generated and measured on a per-unit-time basis rather than on a per-job basis. For example, peripheral I/O behavior was considered in terms of the average time elapsing between requests rather than in terms of requests issued per job. Thus each job in the simulated system behaves as an actual job might behave over a short fraction of its run time. The difference in CPU time requirements therefore should not affect the behavior of the simulated system except in terms of throughput and turnaround times, and its effect on these measures should be a simple linear increase.

The parameter which was varied among the nine test runs, as mentioned above, was the total size of user memory. The values assigned to it were fifty pages, one hundred pages and two hundred pages. The IPC system runs with a

maximum of 750K of user core, which is approximately one hundred eighty-eight 4096-byte pages. This is representative of a large-scale batch system. This figure corresponds fairly closely to the upper limit of two hundred pages used in the model runs. However, this similarity in total size is outweighed by the difference in the number of jobs being multiprogrammed in the actual and the simulated systems. The IPC system, running as a non-paged system, loads an entire job into memory before running it. This means it can accomodate an average of three to four jobs in core simultaneously. In the simulated system under the schedulers being used here, in contrast, each job is assigned a partition size equal to half of its total size. (This is described more fully below.) The modelled system can accomodate up to eight of these jobs in main memory simultaneously when it is given two hundred pages of user memory. Thus it achieves a much higher degree of multiprogramming for the same memory size than the IPC system does. An even larger number of jobs could of course be accomodated by the simulated system if each job were assigned a smaller partition. However, doing this would mean that all the pages needed by a job (its working set) could not be in core simultaneously. This introduces the effects of page contention, which tend to complicate the

issues involved. Assigning partition sizes equal to half of the total size of a job eliminates most page contention for the job mix simulated here. When considered in terms of the number of jobs being multiprogrammed, then, the one hundred page simulated systems correspond most closely to the IPC system. The other two sizes used represent values on either side of this point, giving a spread of behavior about this central focus.

The systems simulated in the test runs, then, represent batch processing systems of varying sizes operating under a virtual memory scheme. We should expect to see behavior appropriate to that type of system reflected in the results of the runs. Due to the lack of paging in the IPC system, the short run times of the simulated jobs and a number of other factors which lead to inaccuracies of representation in the modelled systems, the results produced by the model could not be expected to accurately mirror the operation of the IPC system. However, since the model parameters have realistic values and these values are held constant across the various runs performed, we can expect that comparisons of the results from runs using different core sizes and different strategies will parallel those which might be found in practice.

The Schedulers

The schedulers explored in the test runs embody three different scheduling policies. All three schedulers are built around the same basic code for performing the necessary functions of scheduling in this system. The first scheduler uses a simple round-robin selection scheme. The second uses a preemptive scheme where jobs of higher priority are given preferential treatment. The third pursues a policy of deactivating any job which issues a disk or tape I/O request, scheduling all other jobs in a round-robin manner. Listings of the code for the three schedulers may be found in appendix B. These modules will be discussed individually below.

Similarities Among the Schedulers

Before turning to the differences among the three schedulers tested it is useful to point out the things they have in common. Each of the three schedulers uses the same basic data structure for keeping track of relevant information about all jobs currently in the system. This structure, called STATUS, has entries for the following pieces of information:

- the identification number of a job
- an indicator telling whether it is currently active or inactive

- an indicator telling whether it is currently ready or blocked
- the partition size assigned to it (if it is active)
- the clock time at which it was last assigned to the processor
- a pointer used to chain together the separate copies of this structure which describe the different jobs

The declaration used to define the form of this structure appears in the listings in appendix B. The information maintained in these structures, together with the data passed to the scheduler on each call, constitutes most of the information needed to perform scheduling decisions. When a job is to be chosen to be brought into core, the active/inactive indicator in the STATUS entry is used to distinguish between jobs which are eligible to be activated and those which are already in core. This indicator is also checked in selecting a job to be run, in order to ensure that a job which is not in core is not assigned to be processed. The ready/blocked indicator is used in an analogous manner to ensure that a blocked job is not assigned to be processed. The partition size assigned to each job is used when a job terminates or

is deactivated to determine the resulting amount of free core. The time of the beginning of a job's run interval is used in adjusting the timeslice assigned to the job if its processing is interrupted by some independent event. For instance, if some other job's previously issued page request is completed while a job is running, the scheduler is called and notified of this fact. Often the scheduler wishes to continue processing the running job. It then reassigns the running job to the processor with a timeslice equal to its original timeslice minus the processing time it received before the paging interrupt occurred. This has the desired effect of limiting a job's total processing time to the timeslice originally assigned to it, rather than resetting the timeslice to its full original value every time an independent event occurs in the simulated system. All three schedulers assign an initial timeslice of fifty milliseconds to all jobs. The items of information recorded in the STATUS entry for each job are updated as they change during the course of its processing. New entries are added to the STATUS chain as jobs arrive at the system for processing, and entries are deleted when the job they describe terminates.

The basic tasks performed by the three schedulers each time they are called are as follows: First the information passed via parameters by the DRIVER module is processed.

This involves changing appropriate entries in the STATUS structures and other variables. Then, based on the present state of the system as recorded in the STATUS chain and related variables, commands are issued to activate as many jobs as possible (if any), keeping main memory as fully utilized as possible. The partition size assigned to any job, as mentioned above, is half of its total size. Finally, if one or more jobs are eligible to be run, one of these jobs is chosen to be assigned to the processor. Job zero is selected (i.e. the system is allowed to remain idle) only when no job in the system is eligible to be run. The scheduler makes appropriate entries in its output variables to indicate the commands it wishes to issue and returns control to the main routine. The code used to perform these tasks in the various schedulers is quite similar, differing primarily in the policies used in selecting jobs to be activated and run.

The Round-Robin Scheduler

The round-robin scheduler selects jobs to be activated on the basis of size considerations. It scans the STATUS chain for a job that will fit into the amount of free core available. Since the STATUS chain is constructed by adding new entries onto the end of the chain and each scan is begun

with the first (oldest) entry, this gives jobs which have been in the system longest the best chance of being activated. If a job which requires an amount of core less than or equal to the available space is found, an ACTIVATE command structure is set up to activate it, and the partition size assigned to this job is deducted from the amount of available memory space. The search of the STATUS chain then continues, looking for another job which will fit into the amount of free core still unassigned. This process continues until the end of the STATUS chain is reached. At this point there are no other jobs which can be activated. Once a job is activated it remains active until it terminates. This scheme discriminates against large jobs in favor of keeping core utilization high. In the system modelled in the test runs this discrimination is minimized by the fact that all jobs are in the same size range. In a simulation where this was not the case one might wish to use some other selection scheme in order to ensure that large jobs do not incur unnecessarily long waits before being activated. For instance, jobs might be activated strictly in the order in which they arrive at the system.

The selection of the next job to be run is made in a round-robin manner subject to the status of the job assigned to be run on the last call. If the previously assigned job is still eligible to be run and has not exceeded its time-

slice, it is reassigned to the processor with a reduced time-slice as discussed above. If the running job is now blocked, another job must be chosen to be run. The scheduler searches through the STATUS chain starting with the entry following the one pertaining to the previously running job. The first job encountered in this scan which is both ready and active is selected to be run. This scan is conducted in a circular manner. If the last element in the STATUS chain is reached without encountering a job which is eligible to be run, it continues with the first element in the chain. If no eligible job is found after a complete search of the STATUS chain, job #0 is chosen to be run (i.e. the system remains idle).

The Preemptive Scheduler

The preemptive scheduler operates in a manner similar to the round-robin scheduler, except that it takes the priority levels of the jobs into account. It should be noted that this scheduler has an extra entry in its STATUS structures to record the priority level of each job. When searching for jobs to be activated it makes several passes through the STATUS chain. On the first pass any jobs of the highest priority level which will fit are chosen to be activated. Then a second pass is made to scan for jobs of

the next highest priority level which will fit, and so on, down through the lowest priority level being used. Under this scheme a smaller job of low priority might be activated before a larger high priority job. Thus it is not a strict priority activation scheme, but gives preference to high-priority jobs while keeping core utilization as high as possible. As in the case of the round-robin scheduler, once a job is in core it remains active until it terminates. No job is preempted from active status in favor of a higher priority job.

In choosing the job to be processed next the preemptive scheduler first determines the highest priority level among eligible jobs. If the previously running job is still eligible to run, has not run out its timeslice and is of the highest priority level among the set of eligible jobs, it is reassigned to the processor. If a higher priority job is eligible, the running job is preempted in favor of that job. If the running job is no longer eligible or has timed out, another job must be chosen. As in the case of the round-robin scheduler, a circular scan of the STATUS chain is made, beginning with the entry after the one pertaining to the job run previously. The first eligible job encountered in this scan which is of the highest priority among the set of eligible jobs is chosen to be processed. This scheme

is in effect a round-robin selection within the set of jobs having the highest priority level among the set of eligible jobs. As before, if no jobs of any priority level are eligible to be run job #0 is selected.

The Deactivating Scheduler

The third scheduler resembles the round-robin scheduler in all respects except that it deactivates jobs whenever they issue peripheral I/O requests. The rationale behind this practice is as follows. A job which issues a request to disk or tape will be ineligible to be run for a sizable period of time while its I/O is being performed. If it is deactivated there is a possibility that another job activated in its place may be able to accomplish useful work during this period. Paging I/O is much faster than disk or tape I/O, so jobs waiting for pages may be allowed to remain in core with a much smaller penalty incurred in terms of memory space occupied by an ineligible job. The choice of jobs to be activated on each call is made from among the set of jobs which have been deactivated and whose I/O has been completed as well as jobs which have not yet been activated for the first time. Due to the ordering of the STATUS chain (jobs longest in the system preceding newer arrivals), jobs which have been deactivated and are now

ready to be activated again are given preference over those which have not yet been activated. As before, however, the size of the job is also a factor. A new job which is small enough to fit into available memory space will be activated before a larger job which has been active previously. The selection of a job to be assigned to the processor by the deactivating scheduler is made from among the set of ready and active jobs in the same manner as in the round-robin scheduler.

The three schedulers described above are very simple and unsophisticated as compared with many schedulers used in practice, such as the CTSS onion-skin scheduler described by Scherr (42) or the multiqueue scheduler used in IBM's CP/67 system (56). There are several reasons for this simplicity. First, since the test runs are intended merely as an illustration of the results obtainable from the model and the conclusions which may be drawn from them, simple scheduling schemes are sufficient. Indeed they are preferable in some ways, since they involve fewer factors which must be taken into account in interpreting and comparing the results obtained with each one. Secondly, there are a number of factors which are considered in scheduling for practical systems that are not represented or are not appropriate to consider in the test runs. For instance,

in actual batch systems scheduling strategies often take factors such as job type into account in order to achieve a balanced mix of jobs in core. Since the simulated jobs used for the test runs have not been separated according to type it would not be practical or helpful to use schemes which consider the type of the job in making scheduling decisions. As another example, time-sharing systems often use complex scheduling strategies aimed at providing good response to the terminal users. Since the system under simulation here is a batch system, no effort need be made to ensure quick response. In short, the schedulers used here are not intended to correspond to any particular schedulers used in practice. Rather, they embody three different scheduling philosophies in very simple form and thus provide a basis for the comparison of those philosophies.

A Word of Caution

Several points which have bearing on the accuracy of the results obtained from the test runs should be mentioned before discussing and comparing these results. First, due to the way in which the model is constructed, the actual job stream presented to the simulated systems for processing was not identical across the nine test runs. A single random number generator is used by the model to generate values for the interarrival times of jobs, jobs characteristics,

I/O wait times, etc. The first quantities generated on each run are the characteristics of the jobs initially in core. Since there are more jobs in core initially in a one hundred page system than in a fifty page system, runs using a core size of one hundred pages require more random numbers to perform the generation of characteristics for the initial job load than those using a core size of fifty pages. This difference extends in the same manner to runs using a size of two hundred pages. Thus the random number sequence used to begin actual operation of the model is different for runs using different core sizes. This results in different job interarrival times, and thus differing numbers of jobs arriving for processing during the course of the various runs. Also, for the same reason, the characteristics describing the jobs generated on a given run do not correspond directly to those of the jobs on other runs.

Similar problems exist among runs where the core size was constant but different schedulers were used. The three schedulers select jobs for activation and processing in different ways. For example, the job selected to be run by one scheduler may issue a page fault, requiring the use of one random number to determine the paging device involved in the request and another random number to generate the time until its page request will be satisfied. The job chosen by another scheduler may time out,

requiring no calls to the random number generator. Thus even though these runs begin with the same initial job load, they will eventually use different random numbers for generating corresponding quantities, again producing nonuniform job streams. Probabilistically speaking the job streams for the different runs are identical, since they are generated using the same values for the mean and standard deviation for each characteristic. Due to the differences in the random number sequences, however, the resulting job streams are not the same in actuality. Given runs of sufficient duration these discrepancies would be minimized. In the relatively brief runs made for testing purposes, however, the differences are significant. These discrepancies must be borne in mind in comparing the results obtained from the different runs. In particular, small numerical variations in results may well be due at least in part to differences in the job streams presented to the simulated systems. Only clear trends or marked differences in value should be interpreted as significant.

A second factor which affects the accuracy of the results obtained with the model stems from its method of handling disk I/O. The model treats all disk units as separate entities, any number of which may be accessed simultaneously. This represents a simplification of the

operation of an actual computer system, since in general the interconnections of the I/O channels and control units with the disks results in only certain combinations of disk units which may be accessed in parallel. Thus the representation of the twenty disk units in the test runs creates a simulated environment where contention of disk usage is considerably lower than it would be in an actual system with that number of disk units. This will of course affect the results obtained with the model in terms of disk usage and other related aspects of system behavior. This must be borne in mind in analyzing these results. Similar problems could be expected in modelling systems with interconnected networks of paging devices. However, since there are only two paging devices in the systems simulated here and since the paging store in not intended to resemble that of any specific actual system, we can safely assume that they can be accessed in parallel. Thus the treatment of paging devices in the model should not introduce any specific error factor into the results obtained in the test runs.

Summary of Results from the Test Runs

A sample of the summary output obtained from the model is shown in appendix C, illustrating the different measures collected and output by the model on a standard run. The statistics produced by the test runs may be best compared

by breaking them down into classes to be considered individually. Three classes have been selected for discussion here. They include figures on resource usage (usage of main memory, paging drums and peripheral I/O devices), measures of job behavior under the scheduling schemes used, and figures on overall system performance. Not all the data produced by the test runs is presented here, but rather a representative selection of the available statistics is shown. The values obtained for the statistics in each class are discussed and compared below. Conclusions from the separate classes are then combined to yield a comparative analysis of the performance of the three schedulers used in the simulated systems of varying size.

Resourse Usage - Main Memory Usage

In discussing resource usage a logical starting point is main memory usage. Main memory is an expensive resource, and it is economically favorable to make the fullest possible use of it. Also, the more fully it is utilized the larger the number of jobs that may be multiprogrammed simultaneously. Table 3-1 shows the average number of jobs assigned partitions in primary memory at any one time for each of the nine test runs. The partition size assigned to a job by each of the three schedulers is half of its total size, or about 25

	Round-Robin	Preemptive	Deactivating
50 pages	1.97	1.98	1.76
100 pages	3.66	3.99	3.91
200 pages	7.76	7.37	7.63

Table 3-1

Average Number of Jobs in Main Memory

pages on the average. Since some jobs require slightly more space than the average and some slightly less, we cannot expect an exact fit of main memory size/25 pages jobs on the average. Rather, a figure slightly less than this would be a reasonable expectation. And that is indeed what is observed in each case, with minor numerical fluctuations.

The average amount of primary memory assigned to jobs at any given time (Table 3-2) shows the average number of pages allotted to jobs on each run. These figures run parallel to those of Table 3-1, as would be expected.

	Round-Robin	Preemptive	Deactivating
50 pages	48.49	48.59	44.16
100 pages	92.20	97.88	96.89
200 pages	193.67	185.36	191.91

Table 3-2

Average Amount of Main Memory Assigned to Jobs (in blocks)

The average number of pages actually in main memory (Table 3-3) shows some differences among the three schedulers. These figures represent the actual usage of primary memory as opposed to its allocation to each job, and thus represent a much better guage of main memory usage than do figures on the amount of memory assigned to jobs. Table 3-4 expresses

	Round-Robin	Preemptive	Deactivating
50 pages	35.31	34.70	4.45
100 pages	67.96	73.57	8.93
200 pages	139.57	134.73	16.23

Table 3-3

Average Number of Pages in Main Memory

the figures of Table 3-3 as percentages of the maximum possible number of pages in core. These figures show that

all three schedulers maintain roughly constant percentages of main memory usage over the three core sizes tested. The usage under the round-robin and preemptive schedulers is roughly the same, while that under the deactivating scheduler is almost a factor of ten lower. This observed behavior is consistent with the policy used by this scheduler. The deactivating scheduler removes jobs from main memory each time they issue a disk or tape request. These jobs are replaced by other jobs which start off with a single page in main memory. Other pages of these jobs are brought in on demand. Only a small number of pages for any given job are brought in on the average before it issues a peripheral I/O request, causing its pages to be swapped out and another job to be brought in, again starting with a single page.

	Round-Robin	Preemptive	Deactivating
50 pages	70.6%	69.4%	8.9%
100 pages	68.0%	73.6%	8.9%
200 pages	69.8%	67.3%	8.1%

Table 3-4

Percentage of Maximum Possible Number of Pages in Main Memory (Actual Main Memory Usage)

This leads to low utilization of main memory, and indicates that a partition size of 25 pages is considerably larger than is necessary for this job stream under a deactivating scheme.

Resource Usage - Paging Devices

The figures on the usage of paging devices in the simulated systems illustrate several trends and differences among the schedulers and their performances in the various systems modelled. These figures are shown in Table 3-5 and Table 3-6. The figures on the total number of page faults issued (Table 3-5) show an increase in page faults as memory size increases for all schedulers. This increase may be attributed to the larger number of jobs being multiprogrammed in the larger systems, leading to less idle time and more overall activity in the system. The numbers of page faults generated under the round-robin and preemptive schedulers are roughly the same for corresponding sizes. Systems running under the deactivating scheduler, in con-

	Round-Robin	Preemptive	Deactivating
50 pages	715	721	3290
100 pages	1179	1270	5214
200 pages	1807	1778	5887

Table 3-5

Total Number of Page Faults Incurred

trast, experience three to four times as many page faults as those running under the other two schedulers. As discussed above in connection with core usage, this difference reflects the large percentage of jobs with a small number of pages in main memory on the average under the deactivating scheme. Jobs with only a few of their pages in memory require new pages at a much higher rate than those which have more pages in memory. Thus the deactivating scheduler causes jobs to incur page faults at a rate considerably higher than the round-robin and preemptive schedulers which activate jobs initially and allow them to remain in main memory until completion.

The figures on average page wait time (Table 3-6) show the response of the paging devices to these different levels of paging activity. As discussed above in the section on parameters, the simulated system includes two drums which

	Round-Robin	Preemptive	Deactivating
50 pages	7.612	7.636	8.418
100 pages	7.880	7.893	13.610
200 pages	8.453	8.491	26.112

Table 3-6

Average Page Wait Time (in milliseconds)

are used exclusively for paging. The page requests issued are assumed to be evenly distributed between these two devices. A request issued to an idle device is serviced as rapidly as possible within the limits imposed by the characteristics of the hardware (i.e. its processing is begun immediately). One issued to a device which is already busy must wait until all requests previously queued for this device have been serviced before its servicing is begun. Page requests are queued in a simple first-in-first-out manner in the model, as are peripheral I/O requests. The higher the level of paging activity, the longer the average queue length at the paging devices and thus the longer it will take on the average to service each page request. This reasoning is borne out in the average page wait times observed in the test runs. As mentioned above, larger user memory sizes allow more jobs to be active simultaneously and thus cause a higher level of paging activity. Table 3-6 shows that page wait times increase as memory size increases under all three test schedulers. Also, runs made with the deactivating scheduler, with its heavier paging load due to deactivating and reactivating jobs, show significantly longer page wait times than runs made using the same memory size but another scheduler.

Resource Usage - Disk and Tape Units

Peripheral I/O device usage shows trends similar to those discussed above for paging device usage. Increasing memory size allows more jobs to be active and thus more disk and tape requests are generated, as shown in Table 3-7.

	Round-Robin	Preemptive	Deactivating
50 pages	691	688	837
100 pages	1149	1237	1322
200 pages	1790	1757	1504

Table 3-7

Total Number of Peripheral I/O Requests Issued

Contrary to the pattern shown by page faults, however, runs made under the deactivating scheduler show results which are generally similar to those found in runs made with the other two schedulers. This is to be expected from the mode of operation of the deactivating scheduler. It allows each job to run until it generates a peripheral I/O request and then removes it from core to make room for other jobs which are treated in the same manner. Thus though the processing of a given job may be spread out over a longer period of time (it may not be reactivated immediately when its request is completed), the aggregate level of peripheral I/O requests

issued should be about the same.

The figures on average peripheral I/O wait time (Table 3-8) show trends similar to those found in paging wait times under the round-robin and preemptive schedulers. As user memory size increases and the number of requests grows, wait times for peripheral I/O increase under all three schedulers. Note that while the numbers of page requests and peripheral I/O requests issued are quite similar across the board (excluding the special case of page requests issued under the deactivating scheduler), the average percentage increase in page wait times in going from the fifty page to the two hundred page system in runs under the round-robin and pre-

	Round-Robin	Preemptive	Deactivating
50 pages	58.832	59.398	59.634
100 pages	62.034	62.976	62.124
200 pages	63.157	62.841	62.086

Table 3-8

Average Peripheral I/O Wait Time (in milliseconds)

emptive schedulers is 11.7%, while the corresponding average percentage increase in peripheral I/O wait times over runs made with all three schedulers is 5.6%. This difference in relative change represents a balance between two opposing

tendencies. Disk requests require six to seven times as long to service as do page requests; thus a request issued to a busy disk will be delayed much longer on the average than a request to a busy paging drum. (Tape requests experience no queuing delay in the modelled systems; however, disk requests make up over 80% of the peripheral I/O requests issued in the test runs.) The longer service time for disk requests tends to extend queue lengths and thus increase wait times in comparison to those experienced at the paging devices. This tendency is balanced by the difference in the number of devices of each type available. All page requests must be satisfied by one of two paging drums. Disk requests, in contrast, are equally distributed among twenty disk units, any number of which may be accessed simultaneously in the simulated environment. Thus the probability of a disk request being issued to a busy device is much lower than the corresponding probability for page requests. This tends to make the percentage increase in disk I/O wait time smaller relative to page wait time as the number of requests issued increases.

We must bear in mind here the inaccuracies introduced by the model's treatment of all disk units as individual, simultaneously accessible devices. In an actual system where the interconnections of I/O devices affect the

accessibility of the devices we would expect to find considerably more contention for disk usage than was observed in the simulated system. The probability of issuing a disk request to an inaccessible device is higher in the practical case. The addressed device may itself be busy, or the hardware needed to communicate with it (channels and control units) may be busy servicing other devices. This decreases the benefits of the wide availability of disk units found in the simulated systems and should cause higher average disk I/O wait times than those found in these runs. The amount of difference between an actual system and a simulated version of it would depend on the interconnections used in the actual system and the pattern of references to the different units.

Job Behavior - Page Faults

The figures reflecting job behavior in the simulated systems include measures of average time between page faults and between peripheral I/O requests, and distributions of time spent in each of the running, ready and blocked states. These figures are highly interrelated with the figures on device usage presented above. In many cases they reflect the same facts about the events occurring in the simulated

system as seen from the point of view of the job stream rather than the physical devices. Table 3-9 shows the data

	Round-Robin	Preemptive	Deactivating
50 pages	41.976	41.647	9.120
100 pages	25.451	23.625	5.754
200 pages	16.604	16.874	5.096

Table 3-9

Average Time Between Page Faults (in milliseconds)

gathered on the average time between page faults in the various test runs. This quantity is effectively the converse of the total number of page faults occurring during a run. The figures show that the time between page faults decreases as memory size increases for all three schedulers tested. This pattern reflects the higher level of activity in the larger systems as discussed above in connection with device usage. The deactivating scheduler shows much shorter average times between page faults than either of the other two schedulers for each memory size used. This difference reflects the effects of the mode of operation of the deactivating scheduler from the point of view of the job stream, just as the number of page faults generated reflects it from the device usage point of view.

Job Behavior - Disk and Tape Requests

The figures on average time between peripheral I/O requests are given in Table 3-10. These figures are the converses of the total number of peripheral I/O request figures given in Table 3-7. The pattern of decreasing inter-request times with increasing core size is again followed here as it is in the case of paging behavior as discussed above. The small change in this figure between the one hundred page and two hundred page systems under the deactivating scheduler as compared to the corresponding difference under the other two schedulers again parallels

	Round-Robin	Preemptive	Deactivating
50 pages	43.434	43.645	35.846
100 pages	26.115	24.255	22.693
200 pages	16.762	17.076	19.948

Table 3-10

Average Time Between Peripheral I/O Requests

(in milliseconds)

the situation found in the paging case. In the case of peripheral I/O, however, this effect cannot be attributed to having reached the limiting disk service rate. This is clear from the fact that the other two schedulers show

lower interrequest time for the two hundred page runs. It can be explained instead in terms of the limitations that the paging devices place upon the system. Jobs waiting for pages cannot generate disk or tape requests until after their page requests have been completed. Jobs in the two hundred page system under the deactivating scheduler spend a large fraction of their time waiting for pages as discussed above. Their productivity in terms of both useful computation and requests issued to other devices is decreased accordingly.

Job Behavior - Active Time Distribution

The average distribution of time among the running, ready and blocked states for an active job is shown in Table 3-11. As an example of the interpretation of these figures, the average job in a fifty page system under the round-robin scheduler spent 19.17% of the time it was in main memory in the running state, i.e. in control of the CPU. It spent 28.98% of its time in the ready state waiting for its chance at the processor, and the remaining 51.85% of its time in the blocked state waiting for paging or peripheral I/O to be completed. The time during which a job is not active is not considered in compiling these figures. Several significant trends are displayed by these figures. First, the

Running

	Round-Robin	Preemptive	Deactivating
50 pages	19.17%	18.95%	27.45%
100 pages	17.29%	16.87%	18.55%
200 pages	12.59%	12.96%	10.70%

Ready

	Round-Robin	Preemptive	Deactivating
50 pages	28.98%	29.44%	33.47%
100 pages	22.69%	22.17%	14.98%
200 pages	37.84%	36.20%	8.88%

Blocked

	Round-Robin	Preemptive	Deactivating
50 pages	51.85%	51.61%	39.08%
100 pages	60.02%	60.96%	66.47%
200 pages	49.57%	50.84%	80.42%

Table 3-11

Average Percentages of Active Time in Each Traffic Control State

percentage of time spent in the *running* state decreases uniformly as user memory size increases. This is due to the fact that larger memory sizes allow more jobs to be active on the average, and this represents an increased number of jobs which must share the processor. The more active jobs there are the less time each one of them can spend in the *running* state, i.e. in control of the processor. These figures may be compared to the corresponding maximum theoretical values for *running* state occupancy to get an idea of system utilization. For instance, in the case of the two hundred page system running under the round-robin scheduler we have an average of 7.76 jobs in main memory at any given time (from Table 3-1). This means that if the CPU were in use constantly each job would receive on the average

$$100/7.76 = 12.88\%$$

of the available processor time. The observed figure for this system is 12.59%, which is quite close to this maximum value. This indicates very high CPU utilization. In contrast, the fifty page round-robin system has a maximum theoretical *running* state occupancy of

$$100/1.97 = 50.8\%$$

The observed value in this case is only 19.17%, indicating that the processor is idle a good deal of the time.

Another point of interest in these figures is the difference between the data from the deactivating scheduler runs and that from runs made with the other two schedulers tested. The <u>running</u> state occupancy for the fifty page runs is markedly higher for the deactivating scheduler than for the round-robin or preemptive schedulers. This reflects the fact that no active jobs are waiting for peripheral I/O under the deactivating scheduler. In a small system where there are only one or two jobs in main memory at a time this results in having a job eligible to be run during a significantly larger fraction of the time than a scheme which allows jobs to tie up core while waiting for peripheral I/O to be performed. This advantage diminishes and eventually disappears, however, as we look at the larger systems where more jobs can be multiprogrammed. The <u>running</u> state occupancy observed in the one hundred page system under the deactivating scheduler is not markedly different from that found for the other two schedulers. For the two hundred page system the deactivating scheduler shows a lower value for <u>running</u> state occupancy than do the other two schedulers. This may be attributed to the heavy paging load in this system as discussed above. Jobs in this system are forced to wait much longer for page faults to be satis-

fied, thus preventing them from reentering the _running_ state as soon as they might under other circumstances. This pattern indicates that the deactivating scheme has an advantage over the round-robin and preemptive schedulers in terms of processor utilization in a system where the degree of multiprogramming is small, but this advantage is lost as the number of active jobs grows.

The benefits to be obtained from using a deactivating scheme in a small system may not be as great in reality as they were found to be in the modelled environment. The model does not simulate the paging out of a job which is deactivated. It merely assumes that this paging out occurs in parallel with the execution of some other job. In most cases this does not cause any significant distortion of reflected behavior. However, in the case of the deactivating scheduler used here the level of paging activity is very high. Here the paging out of deactivated jobs could contribute significantly to the congestion at the paging devices, serving to slow the paging response time down and decrease the _running_ state occupancy.

The figures on occupancies of the _ready_ state for the round-robin and preemptive schedulers first decrease and then increase with increasing memory size. The _blocked_ state occupancies follow the reverse pattern. This

behavior may be explained in terms of the interaction of two opposing factors. First, as the number of active jobs goes up the I/O wait times increase because of loading on the devices as discussed above. This means that a job which issues an I/O request must wait longer for it to be completed on the average in a larger system than in a smaller one. This tends to increase the time that jobs spend in the <u>blocked</u> state. This is turn decreases the percentage of time the jobs spend in the <u>ready</u> state. With more jobs in the <u>blocked</u> state a job which becomes ready has fewer jobs to compete with for the processor, and can expect to leave the <u>ready</u> state more quickly. In opposition to this tendency is the effect of the variation in the availability of the CPU. As mentioned above in discussing occupancies of the <u>running</u> state, the two hundred page systems come very close to the theoretical maximum for CPU usage, at least under the round-robin and preemptive schedulers, while the smaller systems show considerably less than maximum utilization. As the number of active jobs increases and the CPU becomes more fully utilized, jobs which become ready to run must wait longer to have a chance at the processor, since there is more competition for it. This tends to cause jobs to pile up in the <u>ready</u> state while they wait for their turn at the processor, and brings

about a corresponding decrease in the number of jobs in the <u>blocked</u> state. Jobs cannot become blocked until they have a chance to be processed and issue an I/O request or incur a page fault. From the pattern of occupancies found in Table 3-11 one can conclude that the effects of I/O wait times are dominant over those of competition for the CPU in the one hundred page systems, while in the two hundred page systems the balance has shifted in the other direction. Here both CPU utilization and I/O wait times have increased, but CPU availability has become the dominant factor.

The pattern of <u>ready</u> and <u>blocked</u> state occupancies for runs made with the deactivating scheduler show a uniform decrease with increasing core size rather than the more variable behavior discussed above for the other two scheduling schemes. This may again be explained in terms of the opposing trends outlined above. In the case of the deactivating scheduler, as opposed to the round-robin and preemptive schedulers, however, the effects of increased I/O wait times outweigh those of contention for the CPU in both the one hundred page and the two hundred page systems. This is due to the much heavier paging load occurring under the deactivating scheme in the larger systems.

Overall System Performance - CPU Idle Time

Overall system performance for the simulated systems is reflected by figures on CPU idle time, throughput and turn-around time as shown in Tables 3-12 through 3-14 and 3-16. CPU idle time (Table 3-12) is the percentage of time during which none of the active jobs were eligible to be run and the scheduler was forced to run job #0. These figures are measured directly by the model's accounting routine, and they corroborate the conclusions reached above from the comparison of theoretical maximum values and observed values for running state occupancy. For all three schedulers tested CPU idle time decreases as user memory size increases. This is due to the increased benefits of multiprogramming with increasing numbers of jobs in core.

	Round-Robin	Preemptive	Deactivating
50 pages	62.42%	62.49%	54.28%
100 pages	37.55%	32.56%	27.63%
200 pages	2.34%	4.44%	18.49%

Table 3-12

CPU Idle Time

The more jobs there are in memory, the more likely it is that at least one of them will be eligible to be run when

it comes time to choose a new job to be processed. It has been found in practice in demand paging systems (Madnick and Donovan (61)) that multiprogramming more than a certain number of jobs leads to degraded performance due to thrashing. However, due to the fact that the model is organized to give each job its own partition and since the schedulers used here assign partitions large enough to minimize page contention, this effect is not realized here.

The patterns of the CPU idle time figures under the round-robin and preemptive schedulers are quite similar to one another. The deactivating scheduler, on the other hand, while showing a decrease in idle time with increasing core size, does not achieve nearly as low a value for idle time in the two hundred page system as do the other two schedulers. This may be attributed to the heavy paging load in the two hundred page system under the deactivating scheduler, which causes longer I/O waits and lower occupancy of the *ready* state as described above. Note that for the smaller memory sizes the deactivating scheduler performs better in terms of idle time than the round-robin and preemptive schedulers. It is not until the core size becomes quite large that its performance degrades relative to that of the other two schedulers. This is an indication that the deactivating scheme has advantages in smaller sys-

tems. If jobs are deactivated when they issue peripheral I/O requests, fewer active jobs are blocked for I/O on the average than in systems where such jobs are allowed to remain in core, and thus more useful work can be accomplished. In larger systems, however, even though some jobs are waiting for long periods for their I/O to be completed there are enough active jobs that the probability that at least one job is runnable is fairly high.

Overall System Performance - Throughput

The figures on average throughput(Table 3-13) represent the average number of jobs terminating per second in the simulated systems. The similarity between the performances

	Round-Robin	Preemptive	Deactivating
50 pages	.867	.90	.933
100 pages	1.37	1.53	1.60
200 pages	2.10	2.03	1.60

Table 3-13

Average Throughput (jobs/second)

of the round-robin and preemptive schedulers is again apparent here. In both cases throughput increases as user memory size increases. This is in line with the data on

CPU idle time, which indicates that the CPU is more fully utilized as memory size increases, thus accomplishing more useful work. The deactivating scheduler, on the other hand, though it compares favorably with the other two schemes in the fifty page and one hundred page systems, shows no gain in throughput in the test runs in going from one hundred to two hundred pages. This is in line with the figures on idle time for this scheduler, and bears out the conclusion reached above that its performance degrades relative to those of the other two schedulers in systems using large memory sizes.

Overall System Performance - Turnaround Time

Figures on average turnaround time were compiled both for the aggregate job stream and for each priority level individually. Table 3-14 shows the turnaround figures for the job stream as a whole. Since the time required for input and output is not considered in compiling these figures, what is referred to as turnaround time here might be more accurately termed system residence time. These figures show that average turnaround time decreases as memory core size increases for all three of the schedulers tested. At first this might seem to run counter to intuitive expectations, since with more jobs in core at a time in the

	Round-Robin	Preemptive	Deactivating
50 pages	9657.484	7176.623	7130.944
100 pages	5649.057	5533.752	8284.443
200 pages	4919.109	4396.211	8248.683

Table 3-14

Average Turnaround Time for the Aggregate Job Stream (in milliseconds)

larger systems each job should get a smaller percentage of the processor time per unit time and thus would have its total system residence time extended. However, there are several other factors which have bearing on this situation and must also be considered. First, it was observed above that the smaller simulated systems had higher CPU idle times. Thus the additional active jobs in a larger system are taking up part of the processor time which was going to waste in the smaller systems rather than causing the same amount of processor time to be shared among a larger number of jobs. Another factor to be considered is the average length of the queue of jobs waiting to be activated for processing. The figures describing the average number of jobs in each of the simulated systems is shown in Table 3-15. In the case of the round-robin and preemptive schedulers there

are in general more jobs resident in the smaller systems than in the larger ones. The round-robin and preemptive schedulers activate each job only once, so the jobs coming into the system form a queue to wait to be activated. Turnaround time measures the time elapsing between the arrival of a job at the system and the completion of its processing. A job which arrives at a system under these scheduling schemes where twenty jobs are already queued for activation must wait considerably longer in general to be activated than if it had arrived at a system where only ten jobs were already queued. Since processing cannot begin until the

	Round-Robin	Preemptive	Deactivating
50 pages	23.10	36.15	13.20
100 pages	10.51	24.81	17.12
200 pages	12.73	11.45	20.57

Table 3-15

Average Number of Jobs in System

job is activated, larger numbers of jobs in the system in the smaller systems cause jobs to wait longer for activation. This in turn leads to higher turnaround times. These conclusions must be modified somewhat in systems where priority considerations are taken into account (i.e.

under the preemptive scheduler), but they are still valid for the job stream considered as a whole.

The average number of jobs in the simulated systems running under the deactivating scheduler, in contrast to the trend discussed above for runs made with the other test schedulers, increases with increasing core size. This is an externally caused effect, accomplished by modifying the value of the parameter governing the interarrival rate of jobs coming into the system in order to ensure that there were enough jobs in the system so that when a job was deactivated another job was in general available to be activated in its place. From the figures on the average number of jobs in core (Table 3-1) we can see that this objective was accomplished. The systems run under the deactivating scheduler show numbers of jobs in main memory comparable to those found with the other two schedulers.

Under the deactivating scheduler jobs do not follow the sequence of awaiting activation, being activated, being processed and terminating as they do under the regimes of the round-robin and preemptive schedulers. Instead, jobs are activated and deactivated a number of times during the course of their processing. We may in effect view this situation as multiprogramming among most, if not all, of the jobs in the system rather than only among

the set of active jobs. The execution of a job run under the deactivating scheduler is made up of a pattern of periods when the job has control of the processor interleaved with periods when the job is blocked. During these blocked periods the job may be either active or inactive, depending on whether it went blocked for paging I/O or peripheral I/O. This means that jobs under this scheduling scheme do not experience the relatively long initial waiting period incurred by jobs in systems run under the other two schedulers. As memory size increases CPU idle time decreases as discussed above; however, the number of jobs effectively sharing the CPU under the deactivating scheme is quite large for all these memory sizes. Thus the effect of the decrease in CPU idle time as memory size increases is not as significant on the individual jobs under the deactivating scheme as it is under the round-robin or preemptive schemes. This resulted in the test runs in turnaround times which increase as memroy size increases for runs made with the deactivating scheduler.

Turnaround time by priority level is significant only in the case of the preemptive scheduler, since this method is the only one of the three schemes considered which takes priority level into account. The figures shown in Table 3-16 pertain only to the preemptive scheduler. The entry for

	Level 1	Level 2	Level 3
50 pages	4064.740	8732.563	----
100 pages	2951.904	4376.914	9732.781
200 pages	2743.777	3041.722	5582.023

Table 3-16

Turnarcund Time by Priority Level (in milliseconds)

(Preemptive Scheduler Only)

priority level 3 (the lowest level) for the fifty page case is blank, indicating that no level 3 jobs terminated during that run. The differences among the turnaround times for the various priority levels decrease as memory size increases, in parallel with the decrease in overall turnaround time. This indicates that the benefits accorded to level 1 jobs become less significant as the system becomes larger and turnaround times decrease for all jobs.

Conclusions

From the foregoing analysis of the data gathered from the nine test runs we can draw several conclusions regarding the relative performances of the three scheduling schemes tested under various user memory sizes in the modelled environment. First, the round-robin and preemptive scheduling schemes show very similar overall performances under most

conditions. The scheduling of jobs according to priority level and the preemption of low priority tasks in favor of higher priority ones does not noticeably degrade overall system performance as measured by such quantities as throughput and CPU idle time. In the simple batch-type systems modelled in the test runs there is no particular benefit to be derived from using the preemptive scheduler rather than the round-robin scheduler. In many cases in practical systems, however, the preemptive approach is much more useful. This is true, for instance, in time-sharing systems where fast terminal response is desired, and in real-time systems where certain tasks must be performed at certain times. The evidence gathered in the test runs indicates that, at least in the type of system being modelled here, preemptive scheduling can be used without degrading overall system performance. We can expect this conclusion to carry over into practical systems to a greater or lesser extent depending upon the degrees of similarity between a practical scheduler and the one used in the test runs, and between an actual system and the simulated environment.

The deactivating scheduler performs somewhat better than the other schemes tested in small systems where the degree of multiprogramming is low. In these situations

it achieves lower CPU idle time and higher throughput than the other schedulers. In larger systems its performance is not as good, however. In the case of a large system with a high degree of multiprogramming it is bogged down by the heavy paging load it generates, resulting in higher idle time figures and lower throughput than the round-robin and preemptive schedulers.

Given the characteristics of the system modelled in the test runs and the nature and inherent unsophistication of the three schedulers tested, we can state the following general conclusions. For a relatively small-scale system a deactivating scheme produces better overall performance in this environment than a scheme which does not perform deactivations for peripheral I/O requests. For larger systems non-deactivating methods are preferable. The choice between the simple round-robin and the preemptive schemes depends upon the need for and emphasis placed upon fast response to certain tasks at the expense of longer turnaround times for others.

The comparison of results from the nine model runs performed indicates that the model produces results which are intuitively realistic. The results obtained here are internally consistent and can be explained in terms of the physical constraints of the modelled systems and the pat-

terns of demands made by the job streams simulated. The figures produced by the model provide a fairly comprehensive picture of the behavior of the simulated system. Graphs of the model results might well provide further insights into trends in the various measures compiled. Graphical display was not considered appropriate here, however, due to the small number of samples taken with any one scheduler. Careful consideration reveals the different statistics produced to be highly interrelated, in many cases reflecting the same facts about system performance from different viewpoints. For instance, the observed percentage occupancies of the ready state as compared to their theoretical maximum values reflect CPU utilization from the point of view of the individual job, while overall figures on CPU idle time show the same situation in terms of the system as a whole.

CHAPTER FOUR

LIMITATIONS OF THE MODEL AND SUGGESTIONS FOR FURTHER STUDY

The construction of a model such as the one described here consists of several phases. The basic design must first be worked out, dividing the tasks to be performed by the model into classes which will be carried out by the various modules. These modules are then coded and tested individually, and when this is complete the modules are assembled to form the model. The task of model-building from this point on becomes an iterative process of comparing the model's behavior to that of actual systems and making modifications to the basic model which improve its approximation to reality or the efficiency with which it performs its various functions. This last phase of development is perhaps the most important one in the entire process, since a model which does not behave in a realistic manner is of little use, regardless of how cleverly it is designed or how elegantly it is coded. It can also be a very time-consuming phase, for many different modifications may be necessary to achieve a good approximation to realistic behavior, and many test runs are needed to determine the model's responses to various sets of conditions.

There is no well-defined end point in this process.

One can almost always come up with another change to a model which might further improve some aspect of its performance. Nonetheless, one must choose a point at which to stop development of the model, at least temporarily, if any useful studies are to be made with it. Several criteria may be used to select this stopping point. Accurate, detailed models are needed for some purposes, such as detailed studies of small changes in a single system parameter. Rougher, more approximate models are adequate for more general studies. The degree of accuracy required in a given model has bearing on the amount of effort needed for the iterative phase of its development. The practical constraints of time and resource limitations also have their effects on this decision.

Taking these various factors into account, the decision was made to stop modifying the model described here when it was in the form outlined in chapter two. It was acknowledged in the discussion in that chapter that the model as such could only be expected to provide estimates of aggregate behavior rather than detailed information on underlying processes. It was used to provide such aggregate information in the test runs described in chapter three. The experience gained in making these runs and analyzing the results obtained from them has brought out a number of interesting points about the model in its present form.

For instance, some limitations and inaccuracies of representation which are inherent in the design of the model have become apparent. These limitations have varying impacts on the usefulness of the model in different contexts. A number of changes which could be made to further improve the model were also pointed out by these experiments. Some of these changes would affect the operation of the model in all cases, while others might be desirable under certain circumstances. Finally, this experience has suggested a number of other experiments which might be performed with the model. These limitations, modifications and further experiments are discussed individually in some detail below.

Limitations and Inaccuracies

Perhaps the first limitation of the model which becomes apparent in considering the results of the runs described in chapter three is in the area of scheduler efficiency. Efficiency is an important consideration in the choice of a scheduling algorithm. A scheduler which takes many factors into account in choosing jobs to be activated or assigned to the processor may require a good deal of CPU time to make its decisions. If too much CPU time is taken up by the execution of the scheduler, any benefits to be gained by its elaborate scheme will be negated in a practical system

by the fact that the jobs it schedules must share a smaller amount of CPU time. The model described here provides no method of measuring or comparing scheduler efficiencies. One can get an estimate of relative efficiency by comparing the execution times of runs made with different schedulers using identical model parameter settings. Due to the variabilities and vagaries of operation of the real world system on which the model program itself is executed, however, this can provide only an approximate measure. The set of jobs being multiprogrammed at any time in the real system, for instance, affects the operation of the system, and this set of jobs will be different on each model run. This leads to discrepancies in the execution times of the different runs. A somewhat better estimate might be obtained by inserting code in the model to note the time in the real world system at which the scheduler is called on each iteration of a simulation and the time at which it returns control to the model supervisor. These figures could be used to generate an estimate of the total real time spent in executing the scheduler. The variabilities of the actual system would still have an effect on these figures, of course, but since the time interval in question is much shorter than that for execution of the entire model we could expect the amount of uncertainty introduced to be smaller.

Another limitation of the model arises from its methods of handling supervisor functions. In an actual system of the type mirrored by the model, supervisory programs are generally used to perform a large number of functions. These include such tasks as maintaining certain items of information on all the jobs in the system, maintaining and referencing page tables, collecting accounting information for billing purposes, and performing scheduling functions. These tasks are represented in various ways in the model. The Job Stream List maintained by the model's driver routine keeps track of information on each job in the system. The paging functions of the operating system are largely ignored in the model since paging is treated only on the macroscopic level. Accounting information of sorts is collected by the model's accounting routine, and scheduling functions are of course handled by the scheduler.

None of these tasks is viewed as requiring any CPU time in the simulated system. There is no convenient way to estimate how much simulated time these tasks would require, and the model is set up in such a way that no simulated time elapses during their performance. The modelled job stream receives one hundred percent of the processor time in the modelled system. Clearly this is not an accurate representation of an actual system, where supervisor functions often

require a significant fraction of the total processor time. One could get around this problem in part by assuming that the CPU time shared among the user jobs is that fraction of the total processor time which is not required by the operating system. However, in many cases the amount of time required for supervisor functions is not constant but instead varies with the activities going on in the system. Many of these activities are controlled by the scheduler. For instance, there would be more paging overhead in a system with a high level of paging activity than in one where fewer pages were demanded. As the operating system overhead goes up, the portion of the CPU time devoted to user jobs goes down. This causes turnaround time to go up and throughput to go down in general, i.e. system performance is degraded. Due to the fact that supervisor execution time is not deducted from total CPU time in the model these factors are not reflected in the results obtained with the model. It is quite possible that a scheduler which produces better performance measures than some other scheme in the environment of the model might perform worse in comparison to the other scheduler when supervisor time requirements are taken into account. This blind spot in the information provided by the model must be kept in mind in drawing conclusions based on comparisons of model runs using different scheduling schemes.

An important part of the design of any model is the choice of the scope of the simulation, i.e. the set of factors which will be represented in the model. In any model of manageable proportions only those aspects of actual systems which are most central to the items under study can be included. This leads of necessity to inaccuracies of representation due to the factors which are left out. For instance, in the model described here jobs are viewed as arriving at the system in a state in which they may be assigned to be processed immediately if desired. The assumption behind this is that there is an input SPOOLing routine which brings in each job from an input device and puts it out to secondary storage, making it possible to load it directly into main memory when it is activated. This input routine is outside the scope of the model; i.e. it is not explicitly represented in the model. Similarly, an output SPOOLing routine is hypothesized to handle output functions, and this routine is also outside the scope of the model. This choice of scope for the model imposes a limitation on the accuracy of the results obtained in that any interactions of the SPOOLing routines with the rest of the model cannot be represented. As an example, in a small system with few disks, the SPOOLing routines may well interfere with user job usage of the disks. The model is unable to reflect

the effects of this competition for access to the disks, which could in some cases have a significant impact on overall system performance.

Another limitation due to the choice of scope stems from the decision to view the processes involved in paging in a simplistic, generalized manner. No effort is made to keep track of which pages of a job are in core in the model. Thus we cannot tell if a page for which a fault occurs has already been referenced and has been written onto the drum or has never been referenced and must be fetched directly from the disk. In the model it is assumed that all pages are fetched from the paging device(s), which is strictly reasonable only if we assume that an entire job is copied onto the paging store when it is first activated. Since this is not the general practice in actual systems, this represents an inherent inaccuracy in the representation of paging behavior. In order to treat the paging process in a more reasonable manner, however, significant changes to the model would be needed. It would be necessary to maintain a list of the pages of each job which had been previously referenced. This list would have to be accessed each time a page fault is incurred to determine whether the page request should be issued to a disk or to a paging device. The additional accuracy of representation in the

model to be gained from this did not seem to merit the extra execution time and storage space it would require. Some improvement in the representation of page wait times could be obtained in the framework of the present model by increasing the average access time and decreasing the transmission rate of the paging devices. The amount by which these quantities should be modified is difficult to determine, however, and there is no easy way to approximate the effects of queuing the requests at different devices.

Along the same lines, the model does not consider the capacity of any of the secondary storage devices it represents. This could result in inaccuracies of representation if, for instance, the size of the paging store on a system being simulated was quite small. In this case the pages of presently active and previously active jobs might overflow the paging store. In many practical systems this would necessitate transfers of pages from the paging devices back to secondary storage. Since paging storage capacity is not represented in the model, this situation could never occur in the simulated system. This again leads to inaccuracies of representation.

A number of studies (57, 58, 59, 60) have been done on the effects of variation of the page size on the paging behavior of programs and on the operation of systems as a

whole. Due to its macroscopic treatment of paging, the model described here is not suited to performing experiments of this nature. Although pagesize is an input parameter to the model, there is no provision for modifying the paging behavior of jobs with changes in this parameter. Instead, paging behavior is determined solely by the exponent used in the paging curve. The value to be used for this exponent must be determined via experimentation using a given page size. Once this exponent is set, changes in pagesize will not change the times between page faults issued, but will only change the lengths of time required to bring new pages into core. Clearly this is highly unrealistic behavior. However, modifying the model to cause it to respond more realistically to changes in pagesize requires incorporating pagesize as a factor in the generation of page wait times. This is an extremely complex task. The effects of pagesize on paging behavior are not yet fully understood, and determining a mathematical relation between them which would be valid even in an average sense is a major task. This task was beyond the scope of the modelling effort described here. Any experiments using different page sizes performed with this model must use paging curves calibrated to produce behavior appropriate to those page sizes.

From the above examples it is clear that there are a number of limitations and inaccuracies inherent in the

model described here. Some of these drawbacks are necessary consequences of the way in which the model was constructed; others represent a tradeoff between the accuracy desired in the results and the modelling effort and computer time and space required to achieve it. Some limitations and inaccuracies are to be expected in any practical model, and they must be kept in mind when drawing conclusions based on results produced by the model.

Additions and Improvements

One problem with the model in its present form became especially apparent in the test runs. This was the use of only one random number generator to produce all the random numbers needed by the model. As discussed in chapter three, the job streams presented to the schedulers on the different model runs were nonuniform due to this fact, and thus the different runs were not directly comparable. This problem could be quite easily remedied by adding a second random number generator to the model. One random number generator could be used to generate job characteristics and interarrival times while the other was used for all other purposes, such as deciding which device was addressed by a given request and how long it would take to service it.

In this case jobs would arrive at the same simulated times and would have identical characteristics in all runs which used the same values for the parameters governing these quantities and the same main memory size. This would provide a much better basis of comparison for the results obtained from the different runs.

For runs using different memory sizes the problem becomes slightly more complicated. Even assuming that two random number generators are used, runs made with different user memory sizes would in general require different numbers of random values for generating the initial job load. This would result in different sequences of random numbers being used to generate the jobs arriving after the beginning of the run. This problem could be solved by resetting the seed value of the random number generator used to generate the job stream to some standard value after generation of the initial job stream was complete on each run. The random numbers produced from that point on would then be identical, resulting in the desired uniform job streams. Note that runs using different memory sizes will still have different initial job mixes, since larger systems in general will have more jobs in core initially than smaller systems. This discrepancy is inherent in the process of using an initial job load, and there is no way to get around it short of

starting all systems off with main memory initially empty.

The paging behavior module (PGNXT) in the present model uses an exponential curve to generate values for the time between page faults. Comparison of the interfault times obtained using these exponential curves with the data from actual systems (Fine et al (48)) shows that the shape of the paging curve is not strictly exponential in general. The SIM/61 experiments (39, 44) use an approximation to this curve with good results. A somewhat more complex paging behavior module which makes use of a more accurate representation of this curve would improve the representation of paging behavior. The form of the curve depends upon the paging behavior of the particular jobs being simulated, and might differ among jobs of different types on a single run. Considerable study might be necessary to determine appropriate approximations. However, once determined, these approximations could be substituted for the present exponential approximation, and could be expected to produce significantly better results.

As discussed briefly in chapter three, the present form of the model makes the rather simplistic assumption that all secondary storage devices may be accessed simultaneously. In a real system this is in general not the case. There are I/O channels and control units involved in

the transfer of data between main memory and these devices (disks, drums and tape drives). Often the devices serviced by a given channel or control unit can only be accessed one at a time. A somewhat more detailed discussion of this situation is presented by Madnick and Donovan (61). One way of representing this situation within the present framework of the model is to view each channel and the devices connected to it as a single device rather than representing each device individually. For instance, as a simple example, consider the system diagrammed in figure 4-1. This system has twelve disk units (D1 through D12) and three I/O channels (C1 through C3). Each channel is connected to four disk units. We could view this group of devices and channels as three large disks. These disks would have the same access times and transmission rates as the individual units attached to each channel. Since the capacity of secondary storage units is not represented in the model, the greater capacity of these aggregate devices makes no difference.

This solution is adequate for systems in which there is only a single path to each device. In many practical systems, however, such as the one shown in figure 4-2, there may be two or more paths to some devices, and this situation presents more complicated problems. This system has the same number of disk units and channels as the one

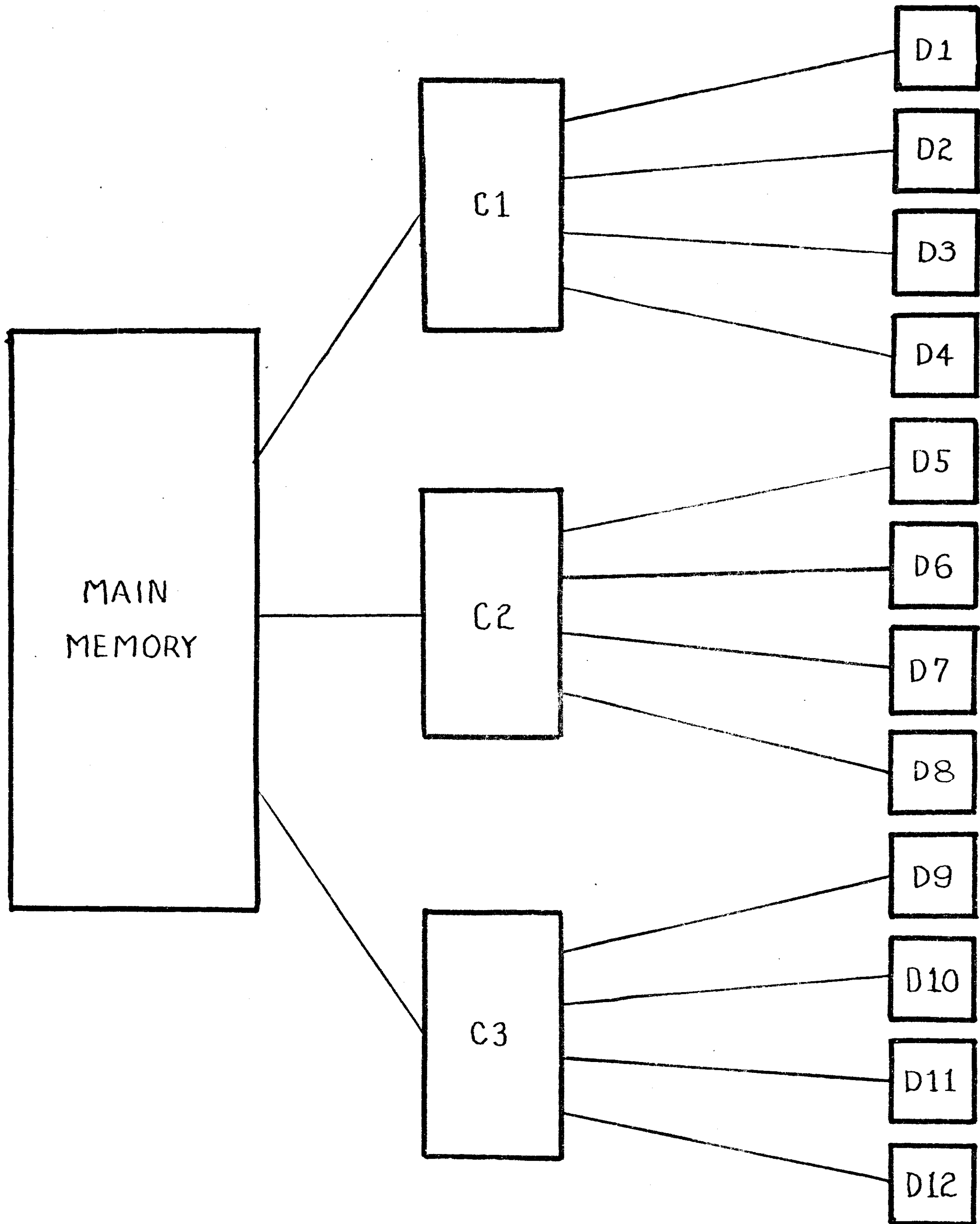


FIGURE 4-1
SIMPLE I/O NETWORK

shown in figure 4-1, but here there are two paths to six of the disks and a single path to each of the other six disks. In order to handle this situation the model must record which devices are connected to each channel. When a request is issued to a given disk it can be serviced only if the device itself and at least one of the channels connected to it is free. If this is not the case the request must wait until a path becomes free. This presents a more complicated queuing problem than the scheme presently in use in the model, since here a request must be queued for two things (a channel and a device) rather than simply for a device. The introduction of control units into the network in addition to I/O channels adds an additional level of complexity to the picture. A corresponding increase in the complexity of the approach described above would be required to handle that situation. A scheme analogous to the one described above would be required to handle the interconnections to the paging devices on the system.

Handling interconnections to tape drives presents different problems than those discussed above for disks and drums since the particular tape drive addressed by any given request is not specified in the model. A partial solution in this case might be to delay each tape request by some probabilistically generated length of time to account for

the effects of contention for channels and control units. The probability distribution to be used in generating these times would depend on the interconnections of the tape drives and the channels and control units in the particular system being simulated. It is not clear how to determine an appropriate form for this distribution in general.

The complicated scheme outlined above should produce more realistic I/O behavior than the present simpler approach in modelling systems which have highly interconnected networks of I/O devices. It should be clear from the discussion above, however, that a considerable amount of additional overhead would be incurred in adding the capability of handling this more complex situation. This extra overhead must be weighed against the more realistic behavior of the model it would produce in deciding whether such an addition would be merited. The simpler scheme of aggregating devices and channels might well prove sufficient in many cases. In studies where disk utilization might be a limiting factor on system performance and where the interconnections of channels and devices are quite complex, on the other hand, it would be unrealistic not to consider these interactions.

In contrast to the complexity of handling I/O networks, an addition to the model which could be made fairly easily is to add the capability of prepaging. Many practical

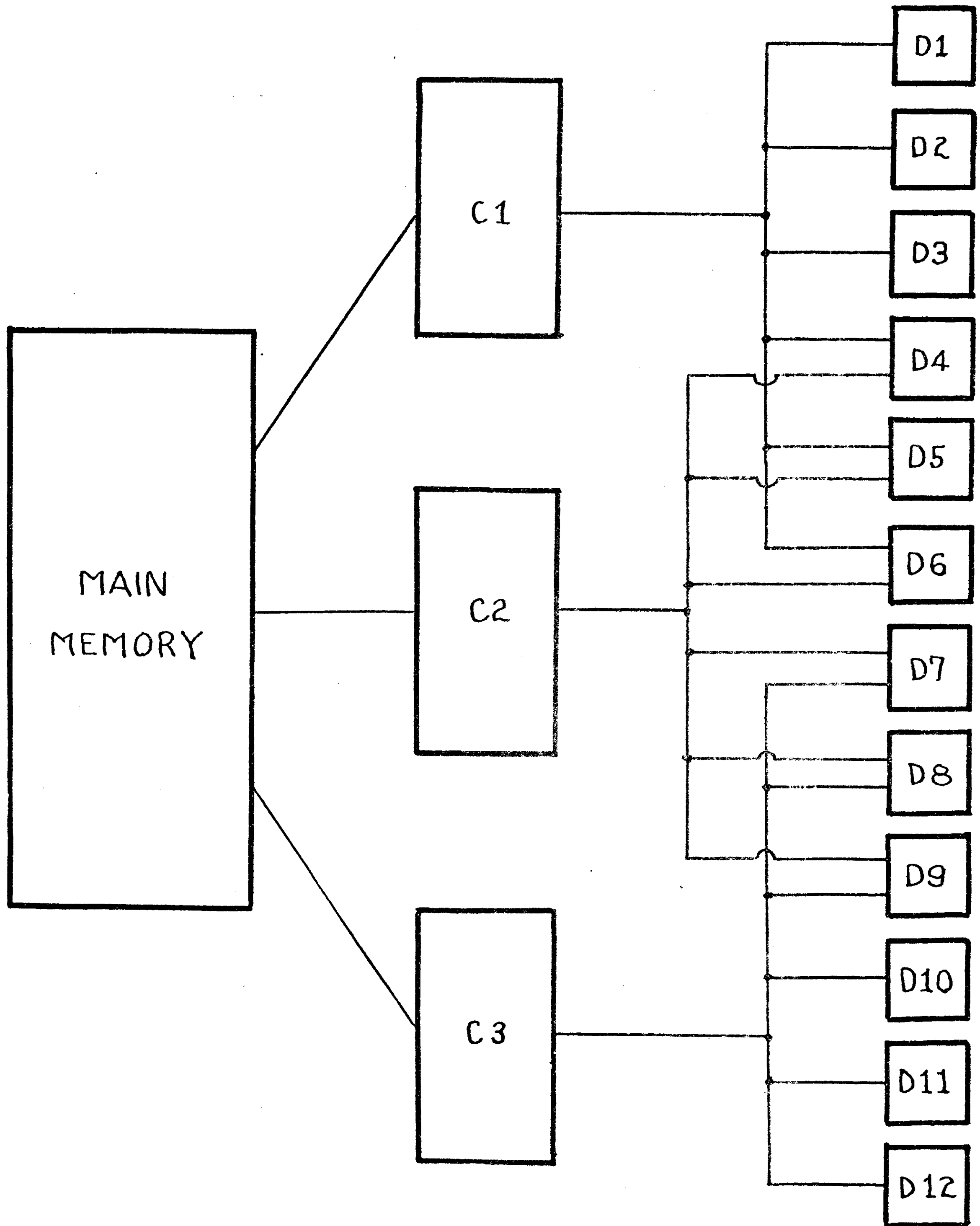


FIGURE 4-2
MORE COMPLEX I/O NETWORK

systems, such as the Multics system developed at MIT's Project MAC (62), bring in several of a job's more recently used pages when it is reactivated rather than starting it off with only a single page and forcing the job to issue page faults to bring in all other pages. To accomplish prepaging in the model one could simply add an additional model parameter to specify the number of pages to be brought in initially when a job is reactivated. If a value of one were input for this parameter the present practice of bringing in a single page would be followed; if some larger value were provided, prepaging would be performed. The choice of which pages are to be brought in need not be considered since pages are not identified in the model. The module which generates page wait times would have to be modified to cause an initial page wait appropriate to the number of pages being brought in. This addition would be quite worthwhile if the model were to be used to simulate systems which perform prepaging.

Analysis of the statistics produced by the model on the test runs suggested several measures not presently provided by the model's accounting routine which might be of use. Depending on the studies to be performed with the model, these or other figures would make useful additions to the output reports produced by the model.

First, throughput figures would be desirable. Throughput for the test runs was computed manually by dividing the elapsed simulated time by the number of jobs terminating during that interval. More reliable measures of this quantity could be obtained more easily by incorporating the compilation of throughput data into the accounting routine. Throughput could be easily broken down by job type and priority level, as is presently done for turnaround time. Other potentially useful figures include the average and maximum queue lengths for requests at each disk and paging device, along with the average queue length for all disks and for all paging devices. We might also wish to collect data to compute the percentage of time for which each device in the simulated system was in use, and again average these figures for all disks and all paging devices.

The foregoing discussion provides only a sample of the additions and modifications which might be made to the model as it exists now. They range from changes as simple as the inclusion of routines for collecting new measures of system performance to quite complex modifications such as accomodating the representation of I/O networks. The decision of whether to make a given change should be based on the tradeoffs between the need for it in a given application

and the extra overhead it would entail in the execution of the model.

Some Further Experiments

A number of ideas for further experiments emerge as simple extensions of the test runs described in chapter three. For instance, it would be interesting to explore the effects of the variation of other parameters besides main memory size on the performance of one or more of the schedulers described there. Such parameters as the number or characteristics of the I/O devices on the system or the characteristics of the simulated job stream might be varied. Experiments might be performed where several factors are varied simultaneously in a coordinated manner. This would yield information on the interactions of the quantities being varied. For example, the I/O demands of the simulated jobs and the capabilities of the I/O devices could be varied to find the point at which these devices become the limiting resource in the simulated system under different levels of I/O activity. As another possibility, the simulated system might be configured to resemble some actual system which could be tried out under various job loads to investigate its capabilities in handling different mixes of tasks. Alternatively, a given job load found in practice could be

submitted to a variety of simulated systems to determine the type of system that was best suited to processing it.

Schedulers that are far more complex than those used in the test runs could be written and run under the model to explore their relative strengths and weaknesses. For instance, one might try out schedulers which might compute dynamic priorities for jobs during the course of their execution, maintain multiple queues for jobs behaving in various different ways, or balance the mix of active jobs according to job type. Schedulers which incorporate a number of such disciplines simultaneously could also be tested. As mentioned in chapter one, a number of studies have been done on the comparison of different scheduling schemes in environments similar to those representable by the model described here. These studies provide an excellent source of suggestions for schedulers to be tried and results with which to compare the data obtained with the model described here. In order to obtain meaningful results from such studies, the job stream to be generated by the model should be described more explicitly in terms of job types and their characteristics than was done for the test runs.

Experiments beyond the realm of practical systems could also be performed in order to gain insights into

hypothetical situations. This might, for instance, involve simulating devices having capabilities which are not possible under current technology but which might someday be realized. This would allow exploration of the differences such devices might make in system performance and overall behavior. As an extension of this we might explore the effects of faster and faster device speeds on the benefits of multiprogramming. The theory behind multiprogramming relies on the fact that jobs are blocked for I/O for some fraction of their run time. Having multiple jobs in core simultaneously allows the processor to go on to another job when the job it has been executing must wait for the completion of an I/O request. As I/O speeds increase, the fraction of its time that each job spends in the blocked state will decrease. This should lead to a decrease in the benefits to be gained from multiprogramming. As I/O wait times approach zero some limiting behavior should obtain. Just what will occur in this range is difficult to determine a priori. The model provides an excellent framework for investigating such a question.

One of the unanswered questions about the model at present is the duration of its startup transient, i.e. the length of time for which it must run before steady-state behavior is achieved. As described in chapter two,

the model is initialized on each run with a full set of active jobs in varying stages of execution. This is done in an effort to minimize the startup transient. Since each scheduler has its own conventions on the way it handles jobs and the amount of core it assigns to each one, however, it still takes some time for the model to settle down under the scheduler being used on a given run. No work has yet been done on exploring how long this transient is under different conditions. This is an important question, since data collected by the accounting routine during the startup phase is included in the final results produced and tends to decrease their accuracy in reflecting the steady-state behavior we wish to measure. Sets of experiments could be performed in which the accounting scan is begun at later and later simulated times. When the results obtained from these successive runs become uniform we can conclude that the startup phase is over.

One of the major uses envisioned for the model described here was as a pedagogical tool. The model provides a realistic environment under which students may test out schedulers they have written. It allows them to observe the behavior of these algorithms in various system environments. Some experience with the model in this context has already been gained from its use by a limited number of students.

Its major drawbacks as a teaching tool appear to be insufficient error-checking of scheduler commands on the part of the model supervisor and an efficiency of operation which is not as good as might be desired. Changes are planned to remedy these problems as much as possible and improve the model's usefulness as a teaching tool.

Finally, experiments which require additions or modifications to the code of the model could be undertaken as an exercise in the processes involved in model building. For instance, the model presently assumes a simple first-come-first-served scheduling of I/O requests to each device. A new module could be written to perform I/O scheduling in some other manner, perhaps according to priority level. This module would have to be incorporated directly into the model rather than replacing some other module, since there is no separate module which performs this task at present. Some changes would have to be made to the model's supervisor routine to interface with the new module.

These examples of opportunities for further work with the model illustrate its applicability to various tasks and the wide range of purposes for which it might be used. Due to its modular construction and the large number of adjustable parameters available to the user the model may be adapted to many different purposes. The model as it now

exists treats many of the processes which interact to make up the operation of a computer system in a macroscopic fashion, but in many cases it can be adapted to provide more detail in a given area without requiring major changes in its overall structure. Though it has limitations and is not in a finalized form the model is nonetheless a useful tool for many applications.

APPENDIX A

HOW TO USE THE MODEL

Running the Model

The model consists of sixteen program modules coded in PL/1, not including the scheduler being used on a given run. The easiest and most inexpensive way to use the model involves compiling each of these program modules separately, linking the object modules together, and storing the resulting load module on secondary storage. A sample of the JCL used to perform this task on the MIT IPC 370/165 system is shown on page 164. The scheduler to be tested on a given run may then be compiled and linked into this load module to form a complete version of the model for execution.

The parameters of the model which are accessible to the user are read in via GET DATA statements during the initialization performed by the supervisory routine on each run. Two separate GET DATA statements are executed. The first input statement reads variable names and values from a data file called THDATA, which must be present in order for the model to be run. This file contains the default values for all user-accessible parameters in the form

VARIABLE = VALUE

```
//JA2222CR JOB 1,
// RIGGINS,CLASS=A,MSGLEVEL=(1,0)
/*MITID USER=(M9999,9999)
/*SRI STANDARD
//LKED EXEC PGM=IEWL,PARM='LIST,XREF,LET',REGION=128K
//SYSPRINT DD SYSOUT=A
//SYSUT1 DD UNIT=SYSDA,SPACE=(CYL,(2,1))
//SYSLIB DD DSN=SYS1.PLILIB,DISP=SHR
//         DD DSN=SYS2.PLISSP.SUBR,DISP=SHR
//SYSLMOD DD DSN=U.M9559.10529.LM128,DISP=(MOD,KEEP)
//SYSLIN DD *

                .

                .

                .

OBJECT VERSIONS OF MODULES COMPRISING THE MODEL

                .

                .

                .

  NAME TELM2
/*
/*EOJ JA2222CR
```

SAMPLE JCL FOR MAKING LOAD MODULE

```
DNMEAN(1) = 16.  DNMEAN(2) =  0.  DNMEAN(3) =  0.  DNMEAN(4) =  0.
DNMEAN(5) =  0.  DNMEAN(6) =  0.     DNSTAN_DEV = 2.

DTACCESS(1)  = 75.  DTACCESS(2)  = 75.  DTACCESS(3)  = 75.
DTACCESS(4)  = 75.  DTACCESS(5)  = 75.  DTACCESS(6)  = 75.
DTACCESS(7)  = 75.  DTACCESS(8)  = 75.  DTACCESS(9)  = 75.
DTACCESS(10)= 75.  DTACCESS(11)= 75.  DTACCESS(12)= 75.
DTACCESS(13)= 30.  DTACCESS(14)= 30.  DTACCESS(15)= 30.
DTACCESS(16)= 30.  DTACCESS(17)= 30.  DTACCESS(18)= 30.
DTACCESS(19)= 30.  DTACCESS(20)=30.

DTTRANS(1)  = 312. DTTRANS(2)  = 312. DTTRANS(3)  = 312.
DTTRANS(4)  = 312. DTTRANS(5)  = 312. DTTRANS(6)  = 312.
DTTRANS(7)  = 312. DTTRANS(8)  = 312. DTTRANS(9)  = 312.
DTTRANS(10)= 312. DTTRANS(11)= 312. DTTRANS(12)= 312.
DTTRANS(13)= 806. DTTRANS(14)= 806. DTTRANS(15)= 806.
DTTRANS(16)= 806. DTTRANS(17)= 806. DTTRANS(18)= 806.
DTTRANS(19)= 806. DTTRANS(20)=806.

DTFREQ(1)  = .05. DTFREQ(2)  = .05. DTFREQ(3)  = .05. DTFREQ(4)  = .05.
DTFREQ(5)  = .05. DTFREQ(6)  = .05. DTFREQ(7)  = .05. DTFREQ(8)  = .05.
DTFREQ(9)  = .05. DTFREQ(10)= .05. DTFREQ(11)= .05. DTFREQ(12)=.05.
DTFREQ(13)= .05. DTFREQ(14)= .05. DTFREQ(15)= .05. DTFREQ(16)=.05.
DTFREQ(17)= .05. DTFREQ(18)= .05. DTFREQ(19)= .05. DTFREQ(20)=.05.

DTREC_SIZE = 1000.  DTSD_ACCESS = 3.  DTRECORDS = 10.  DTTPTRANS = 160.
DTSD_RECORDS = 3.
DTTPREC_SIZE =800.    DTINTERREC_TIME = 4.  DTREL_OT = .821.

PNPAGE_EXP(1) =.0003. PNPAGE_EXP(2) =.0.    PNPAGE_EXP(3) =.00.
PNPAGE_EXP(4) =.0.    PNPAGE_EXP(5) =.0.    PNPAGE_EXP(6) =.0.

PTFREQ(1) = .5.  PTFREQ(2) = .5.  PTFREQ(3) = 0..  PTFREQ(4) = 0.
PTFREQ(5) = .0.
```

```
PTACCESS(1) = 4,  PTACCESS(2) = 4.  PTACCESS(3) = 0.  PTACCESS(4) = 0,
PTACCESS(5) = 0.

PTTRANS(1) =1200,  PTTRANS(2) =1200.  PTTRANS(3) = 0,
PTTRANS(4) =    0,  PTTRANS(5) =    0.
PTSTAN_DEV = 1,

TYPE_ARRAY(1) = 1.0. TYPE_ARRAY(2) = 0..  TYPE_ARRAY(3) = 0.,
TYPE_ARRAY(4) = 0.,  TYPE_ARRAY(5) = 0.,  TYPE_ARRAY(6) = 0.,

PRIOR_ARRAY(1)=.0761, PRIOR_ARRAY(2)=.3749. PRIOR_ARRAY(3) = .549.
PRIOR_ARRAY(4) = .0.  PRIOR_ARRAY(5) = .0.  PRIOR_ARRAY(6) = 0.,
PRIOR_ARRAY(7) = .0.  PRIOR_ARRAY(8) = .0.  PRIOR_ARRAY(9) = 0.,
PRIOR_ARRAY(10)= 0.,

SFACTOR1 = 2..  SFACTOR2 = 1..  SIZE_SD = 3.
SIZE_MEAN(1) = 25,  SIZE_MEAN(2) =  0,  SIZE_MEAN(3) =  0.
SIZE_MEAN(4) =  0,  SIZE_MEAN(5) =  0,  SIZE_MEAN(6) =  0.

JTFACTOR = 1.5,

INT_RATE = .01,
MAXTIME = 400000,  TOTAL_SPACE = 100.  PAGESIZE = 4096,
PRECISION = 10.  DEBUG_ON = 0.  DEBUG_OFF = 0,
TRACE_ON = 0,    TRACE_OFF = 0.
ACCNT_TIME = 0.  STDEFAULT = 50000:
```

SAMPLE VERSION OF THDATA

```
//JTEST JOB 1,
// PIGGINS,CLASS=C,MSGLEVEL=(1,0)
/*MITID USER=(M9999,9999)
/*MAIN TIME=3,LINES=1
/*SRI LOW
// EXEC PGM=IEWLDRGO,REGION=128K
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=2036)
//SYSLOUT DD SYSOUT=A
//THDATA DD DSN=U.M9559.10529.THDATA,DISP=SHR
//SYSLIB DD DSN=SYS1.PL1LIB,DISP=SHR
//        DD DSN=SYS2.PL1SSP.SUBR,DISP=SHR
//SYSLIN DD DSN=U.M9559.10529.LM128(THLM1),DISP=SHR
// DD *

            .

            .

OBJECT VERSION OF SCHEDULER

            .

            .

/*
//SYSIN DD *
 MAXTIME = 30000000,
 TOTAL_SPACE=50,
 INT_RATE = 2.5;
/*
/*EOJ JTEST
```

SAMPLE DECK SETUP FOR RUNNING THE MODEL

where VARIABLE is the name of a parameter and VALUE is the value to be assigned to it. The different assignments are separated by commas, and the sequence is terminated by a semicolon. The deck used to create the copy of THDATA used in the test runs is shown on pages 165 and 166. Use of this file permits the adjustment of default values without the necessity of recompiling and relinking the model. No initializations are performed in the code of the model itself; thus it is essential that all parameters be assigned values in this file.

The second GET DATA statement reads variable names and values from SYSIN. Any variable which should have a value different from that assigned to it in THDATA may be reassigned via card input, using the same format as described above for entries in THDATA. This facility allows for the modification of one or more parameters on each run without the necessity of making a new copy of THDATA each time. The use of the DATA option for input allows easy specification of the particular variables whose values are to be modified. At least one variable must be specified in SYSIN on each run. If no modification is desired to the value of any parameter assigned in THDATA then some variable name should be input via a data card reassigning it the value given to it in the file. A sample deck setup for running the model is given

on page 167. It shows the JCL used to link a scheduler in object form into the load module made up of the other modules of the model. Three variables are assigned values via SYSIN in this sample.

The Parameters

The parameters which are accessible to the user are listed below, along with a brief discussion of the function of each one. Variable names which are followed by numbers in parentheses are array variables. The number associated with each name indicates the size of that array. All arrays are one-dimensional, and have index values beginning with one. Those variables which deal with memory units are described below as being expressed in bytes or words. The size of the memory unit used is of no concern to the model itself. As long as all specifications of memory size are made using the same basic unit, the model will function in the proper manner. The unit used on any given run must, however, be taken into account in assigning values to parameters and interpreting the results produced.

The parameters are broken down for ease of reference into the following categories:

- simulation control parameters.

These variables determine overall model characteristics such as the length of the run to be performed, user memory size, and arrival rate of jobs coming into the system.

- parameters specifying job characteristics.
  These quantities govern the nature of the jobs which will be generated by the model, e.g. their size, CPU time requirement, and I/O and paging behavior.
- parameters governing the characteristics and usage of the I/O devices.
  These include device speeds and access times, record sizes and distributions of device usage.
- parameters specifying the type(s) of output to be produced by the run.
  This includes options for DEBUG and TRACE printing and for the standard accounting summary.

Simulation Control

MAXTIME - time limit for the model run, in microseconds

TOTAL_SPACE - total main memory space available to user programs, in pages.

INT_RATE - interarrival rate of jobs entering the system, in jobs per second.

STDEFAULT - default value, in microseconds, to be used as a timeslice if the value specified for this quantity by the scheduler is invalid.

PRECISION - the number of iterations to be performed in the routine used to generate normally distributed values. A value of ten for this variable yields good sample values with minimal overhead.

Job Characteristics

TYPE_ARRAY(6) - relative frequency of occurrence of jobs of the corresponding job type. If fewer than six job types are needed the unused entries should be set to zero. The sum of all entries in this array must equal one.

PRIOR_ARRAY(10) - relative frequency of occurrence of jobs of the corresponding priority level. As above, if fewer than ten priority levels are desired the unused entries in this array should be set to zero, and all entries must sum to one. Level one is the highest priority; level ten the lowest.

SIZE_MEAN(6) - mean value for the working set size of jobs of the corresponding type, in pages.

SIZE_SD - standard deviation of values for working set size.

SFACTOR1 - a multiplier applied to the mean values for working set size given in SIZE_MEAN to yield mean values for the total size of jobs of each type.

SFACTOR2 - a multiplier applied to SIZE_SD to yield a value for the standard deviation of the total size figures.

JTFACTOR - a factor used in generating values for the total CPU time required by a job. It governs the length of time for which a job will run after its last page is in main memory. This variable must have a value greater than 1.0, and values close to 1.0 yield run times which are longer than those obtained using larger values.

PNPAGE_EXP(6) - exponent used in generating values for the time for which a job of the corresponding type will run before generating its next page fault. The larger the exponent, the smaller the time intervals generated.

DNMEAN(6) - average compute time between peripheral I/O requests issued by jobs of the corresponding type, in milliseconds.

DNSTAN_DEV - standard deviation of the times between peripheral I/O requests.

DTREL_DT - relative frequency of disk operations versus tape operations. A value of .5 for this parameter implies that disk and tape operations are equally likely; values greater than .5 imply that more disk operations than tape operations are performed.

Characteristics and Usage of Devices

Paging Devices:

PTACCESS(5) - average access time for the corresponding paging device, in milliseconds

PTSTAN_DEV - standard deviation of access time values for paging devices.

PTTRANS(5) - transmission rate of the corresponding paging device, in bytes or words per millisecond.

PAGESIZE - number of memory units (bytes or words) per page.

PTFREQ(5) - relative frequency of reference to the corresponding paging device. If fewer than five paging devices are desired, the unused entries in the array should be set to zero. The sum of all entries in the array must equal one.

Disks:

DTACCESS(20) - average access time for the corresponding disk, in milliseconds.

DTSD_ACCESS - standard deviation of disk access time values.

DTTRANS(20) - transmission rate of the corresponding disk, in bytes or words per millisecond.

DTREC_SIZE - record size to be used in disk operations, in bytes or words.

DTFREQ(20) - relative frequency of requests to the corresponding disk. A maximum of twenty disks may be specified as being part of the simulated system at any time; if a smaller number is desired the unused entries should be set to zero. The sum of all entries in this array must equal one.

Tape Drives:

DTRECORDS - the average number of tape records which must be scanned before the desired record is reached on a given tape operation.

DTSD_RECORDS - standard deviation of the number of records scanned before reaching the desired record.

DTINTERREC_TIME - interrecord time (read/write access time) of the tape drives, in milliseconds.

DTTPTRANS - transmission rate of the tape drives on the simulated system, in bytes or words per millisecond.

DTTPREC_SIZE - record size to be used in tape I/O operations, in bytes or words.

Output to be Produced

DEBUG_ON - simulated time at which DEBUG printing is to begin, in microseconds.

DEBUG_OFF - simulated time at which DEBUG printing is to be discontinued, in microseconds.

(If no DEBUG printing is desired, DEBUG_ON should be given a value greater than or equal to that given to DEBUG_OFF.)

TRACE_ON - simulated time at which TRACE printing is to begin, in microseconds.

TRACE_OFF - simulated time at which TRACE printing is to be discontinued, in microseconds.

(As above, if no TRACE output is desired, TRACE_ON should be assigned a value greater than or equal to that given to TRACE_OFF.)

ACCNT_TIME - simulated time at which the standard accounting scan is to begin, in microseconds. If this variable is assigned a value of zero, accounting information is compiled throughout the run; if a positive value is input for it, information is collected only after the simulated time in the model exceeds this value. If no accounting report is desired this variable should be set to a value greater than that given to MAXTIME, the total duration of the simulation run.

APPENDIX B

LISTINGS OF THE TEST SCHEDULERS

```
STMT LEVEL NEST
                  /* *****                  ROUND ROBIN SCHEDULER                   ***** */
                  /*****************************************************************/
  1               SCHED:PROCEDURE(JOBNUM,INDEX,TIME,SPACE,SJPTR,NXTJOB,TSLICE,IPTR,OPTR);

                  /* THIS SCHEDULER CHOOSES A JOB TO BE RUN FROM AMONG THE READY JOBS IN
                     CORE IN A ROUND ROBIN FASHION.  JOBS ARE ACTIVATED IN THE ORDER IN
                     WHICH THEY ARRIVED AT THE SYSTEM AS SOON AS THERE IS ROOM FOR
                     THEM.  THEY REMAIN IN CORE UNTIL THEY TERMINATE. */

                  /* JOBNUM IS THE NUMBER OF THE JOB WHICH HAS BEEN RUNNING JUST
                     PRIOR TO THIS CALL TO THE SCHEDULER.  INDEX INDICATES THE CAUSE
                     OF THE TERMINATION OF ITS PROCESSING.  INDEX VALUES HAVE
                     MEANINGS AS FOLLOWS:
                        INDEX = -1  ---    INITIALIZATION
                                 0  ---    TERMINATION
                                 1  ---    PAGE REQUEST ISSUED
                                 2  ---    DISK OR TAPE REQUEST ISSUED
                                 4  ---    TIME SLICE RUNOUT
                                10  ---    JOB ARRIVAL
                                11  ---    PAGE REQUEST SATISFIED
                                12  ---    DISK OR TAPE REQUEST SATISFIED
                     SPACE GIVES THE TOTAL MEMORY SPACE AVAILABLE TO USER PROGRAMS, AND
                     TIME THE PRESENT TIME IN THE MODEL AS RECORDED ON THE SYSTEM CLOCK
                     IN THE MAIN ROUTINE.  SJPTR IS A POINTER TO THE FIRST ENTRY IN THE
                     JOB STREAM LIST MAINTAINED BY THE MAIN ROUTINE. NXTJOB IS THE
                     NUMBER OF THE JOB CHOSEN BY THE SCHEDULER TO BE PROCESSED NEXT,
                     AND TSLICE IS THE TIMESLICE ASSIGNED TO IT.  IPTR IS A POINTER TO
                     THE FIRST SWAPIN COMMAND ISSUED BY THE SCHEDULER AND OPTR IS A
                     POINTER TO THE FIRST SWAPOUT COMMAND. */

  2     1            DCL(JOBNUM,INDEX,SPACE) FIXED BIN(15),TIME FIXED BIN(31),SJPTR PTR;
  3     1            DCL (IPTR,OPTR)PTR, NXTJOB FIXED BIN(15), TSLICE FIXED BIN(31);
                                               /*ARGUMENTS*/

                  /* STRUCTURES FOR INDICATING SWAPIN AND SWAPOUT COMMANDS */

  4     1            DCL 1 SWAPOUT BASED(SOPTR),
                           2 JOB# FIXED BIN(15),
                           2 ATIME FIXED BIN(31),
                           2 NEXT POINTER;

  5     1            DCL 1 SWAPIN BASED(SIPTR),
                           2 JOB# FIXED BIN(15),
                           2 SIZE FIXED BIN(15),
                           2 NEXT POINTER;

                  /* VERSION OF JOB DESCRIPTIONS AVAILABLE TO SCHEDULER */

  6     1            DCL 1 SJOB BASED(SJPT),
                            2 JOB# FIXED BIN(15),        /*MATCHES UPPER PORTION OF*/
                            2 TYPE FIXED BIN(15),        /*JOB STREAM LIST ENTRIES*/
```

```
STMT LEVEL NEST

                        2 PRIORITY FIXED BIN(15),    /*USED BY MAIN ROUTINE*/
                        2 SIZE FIXED BIN(15),
                        2 NEXT PTR;

   7     1          DCL 1 STATUS BASED(STATPT),       /*STRUCTURE FOR KEEPING TRACK OF*/
                        2 JOB# FIXED BIN(15),        /*ACTIVE/INACTIVE AND TRAFFIC*/
                        2 ACT_IND BIT(1),            /*CONTROL STATUS, PARTITION SIZE*/
                        2 TC_IND BIT(1),             /*AND BEGINNING OF EACH RUN */
                        2 PART_SIZE FIXED BIN(15),  /*INTERVAL FOR EACH JOB*/
                        2 BEG_TIME FIXED BIN(31),
                        2 NEXT PTR;

   8     1          DCL(FSTATPT,LSTATPT) POINTER STATIC;
                                          /*HOLD LOCATION OF INITIAL AND FINAL
                                            STATUS BLOCKS*/

   9     1          DCL(TPTR,DPTR) PTR, FOUND BIT(1), (NEXTJOB,KPTR) PTR STATIC;
  10     1          DCL MEMSPACE FIXED BIN(15) STATIC;   /*RUNNING RECORD OF FREE MEM.*/
  11     1          DCL ZERO_FLAG BIT(1) INITIAL('0'B);
  12     1          DCL RTIME FIXED BIN(31);

                 /* PROCESS INPUT INFORMATION */

  13     1          IPTR = NULL;                  /*INITIALIZE SWAPIN POINTER TO NULL*/

  14     1          IF INDEX = -1
  15     1          THEN DO;                      /*FIRST CALL - INITIALIZE THINGS*/
  16     1     1    OPTR = NULL;                  /*INITIALIZE SWAPOUT POINTER TO NULL -
                                                    NO SWAPOUTS DONE BY THIS SCHEDULER*/
  17     1     1       MEMSPACE = SPACE;          /*INITIALIZE RECORD OF FREE MEMORY*/
  18     1     1       TPTR = SJPTR;
  19     1     1       DPTR = NULL;
  20     1     1       DO WHILE (TPTR ¬= NULL);               /*CREATE A STATUS ENTRY*/
  21     1     2          ALLOCATE STATUS SET(STATPT);       /*FOR EACH JOB CURRENTLY*/
  22     1     2          STATUS.JOB# = TPTR->SJOB.JOB#;     /*IN THE SYSTEM*/
  23     1     2          STATUS.TC_IND = '1'B;       /*ALL JOBS INITIALLY READY*/
  24     1     2          STATUS.ACT_IND = '1'B;        /*ALL JOBS INITIALLY ACTIVE*/
  25     1     2          STATUS.PART_SIZE = (TPTR->SJOB.SIZE + 1)/2;
                                          /*INITIAL PARTITION SIZE IS HALF OF
                                            TOTAL SIZE*/
  26     1     2          MEMSPACE = MEMSPACE - STATUS.PART_SIZE;
                                          /*KEEP TRACK OF HOW MUCH MEMORY IS STILL
                                            FREE*/
  27     1     2          IF DPTR = NULL         /*PERFORM LINKING*/
  28     1     2          THEN FSTATPT = STATPT;
  29     1     2          ELSE DPTR->STATUS.NEXT = STATPT;
  30     1     2          DPTR = STATPT;
  31     1     2          TPTR = TPTR->SJOB.NEXT;
  32     1     2       END;
  33     1     1       LSTATPT = DPTR;            /*KEEP LOCATION OF FINAL BLOCK*/
```

```
STMT LEVEL NEST

  34     1     1          LSTATPT->STATUS.ACT_IND = '0'B;
                                                  /*LAST JOB GENERATED IS INACTIVE*/
  35     1     1          MEMSPACE = MEMSPACE + LSTATPT->STATUS.PART_SIZE;
                                                  /*CORRECT COUNT OF TOTAL FREE MEMORY
                                                    FOR INACTIVE JOB*/
  36     1     1          LSTATPT->STATUS.NEXT = FSTATPT;    /*MAKE LIST CIRCULAR*/
  37     1     1          KPTR = FSTATPT->STATUS.NEXT;  /*THIS POINTER HOLDS A PLACE IN
                                                             THE JOB CHAIN*/
  38     1     1          OPTR = NULL;              /*NO SWAPOUTS DONE BY THIS SCHEDULER*/
  39     1     1       END;

  40     1             ELSE IF INDEX = 0
  41     1             THEN DO;                   /*A JOB HAS TERMINATED*/
  42     1     1          TPTR = FSTATPT->STATUS.NEXT;
  43     1     1          DPTR = FSTATPT;
  44     1     1          FOUND = '0'B;
  45     1     1          DO WHILE (FOUND = '0'B);        /*SEARCH FOR STATUS ENTRY*/
  46     1     2             IF TPTR->STATUS.JOB# = JOBNUM
  47     1     2             THEN DO;               /*ENTRY FOUND*/
  48     1     3                MEMSPACE = MEMSPACE + TPTR->STATUS.PART_SIZE;
                                                  /*RETURN CORE TO FREE AREA*/
  49     1     3                DPTR->STATUS.NEXT = TPTR->STATUS.NEXT;
                                                  /*ADJUST LINKING OF STATUS CHAIN*/
  50     1     3                IF LSTATPT = TPTR
  51     1     3                THEN LSTATPT = DPTR;  /*RESET POINTER TO LAST ENTRY*/
  52     1     3                FREE TPTR->STATUS;    /*DELETE THIS ENTRY*/
  53     1     3                FOUND = '1'B;
  54     1     3             END;
  55     1     2             ELSE DO;               /*KEEP LOOKING FOR PROPER ENTRY*/
  56     1     3                DPTR = TPTR;
  57     1     3                TPTR = TPTR->STATUS.NEXT;
  58     1     3             END;
  59     1     2          END;
  60     1     1       END;

  61     1             ELSE IF (INDEX=1)|(INDEX=2)|(INDEX=11)|(INDEX=12)
  62     1             THEN DO;                   /*I/O REQUEST ISSUED OR SATISFIED*/
  63     1     1          TPTR = FSTATPT;
  64     1     1          FOUND = '0'B;
  65     1     1          DO WHILE (FOUND = '0'B);  /*SEARCH FOR DESCRIPTION OF JOB IN
                                                       QUESTION*/
  66     1     2             IF TPTR->STATUS.JOB# = JOBNUM
  67     1     2             THEN DO;               /*PROPER ENTRY FOUND*/
  68     1     3                IF INDEX >= 11
  69     1     3                THEN TPTR->STATUS.TC_IND = '1'B;
                                                  /*REQUEST SATISFIED - JOB IS READY*/
  70     1     3                ELSE TPTR->STATUS.TC_IND = '0'B;
                                                  /*REQUEST ISSUED - JOB IS BLOCKED*/
  71     1     3                FOUND = '1'B;
  72     1     3             END;
```

-180-

```
STMT LEVEL NEST

 73     1    2              ELSE TPTR = TPTR->STATUS.NEXT;    /*KEEP LOOKING*/
 74     1    2           END;
 75     1    1        END;

 76     1             ELSE IF INDEX = 10
 77     1             THEN DO;                      /*NEW ARRIVAL - GENERATE STATUS ENTRY*/
 78     1    1           ALLOCATE STATUS SET(STATPT);
 79     1    1           STATUS.JOB# = JOBNUM;      /*INITIALIZE VALUES FOR NEW ENTRY*/
 80     1    1           STATUS.ACT_IND = '0'B;
 81     1    1           STATUS.TC_IND = '1'B;
 82     1    1           FOUND = '0'B;
 83     1    1           TPTR = SJPTR;              /*INITIALIZE POINTER TO JOB STREAM LIST*/
 84     1    1           DO WHILE (FOUND = '0'B);         /*FIND JOB DESCP. FOR SIZE*/
 85     1    2              IF TPTR->SJOB.JOB# = JOBNUM
 86     1    2              THEN FOUND = '1'B;     /*PROPER ENTRY FOUND*/
 87     1    2              ELSE TPTR = TPTR->SJOB.NEXT;      /*KEEP LOOKING*/
 88     1    2           END;
 89     1    1           STATUS.PART_SIZE =(TPTR->SJOB.SIZE+1)/2;
 90     1    1           LSTATPT->STATUS.NEXT = STATPT;  /*LINK NEW ENTRY INTO CHAIN*/
 91     1    1           LSTATPT = STATPT;
 92     1    1           STATUS.NEXT = FSTATPT;
 93     1    1        END;

                   /* SWAP AS MANY JOBS AS POSSIBLE INTO CORE*/

 94     1             DPTR= FSTATPT->STATUS.NEXT;     /*INITIALIZE POINTER TO STATUS LIST*/
 95     1             DO WHILE (DPTR¬= FSTATPT);
                                                    /*SEARCH THROUGH THE JOB LIST FOR
                                                      INACTIVE JOBS THAT WILL FIT INTO FREE
                                                      CORE*/
 96     1    1           IF (DPTR->STATUS.ACT_IND = '0'B)&(MEMSPACE>= DPTR->STATUS.
                            PART_SIZE)&(DPTR->STATUS.TC_IND = '1'B)
 97     1    1           THEN DO;                   /*NEXT JOB IS READY AND WILL FIT - MAKE
                                                      ENTRIES TO SWAP IT IN*/
 98     1    2              ALLOCATE SWAPIN;
 99     1    2              IF IPTR = NULL
100     1    2              THEN IPTR = SIPTR;      /*INITIALIZE POINTER TO FIRST SWAPIN*/
101     1    2              ELSE TPTR->SWAPIN.NEXT = SIPTR;     /*LINK SWAPINS*/
102     1    2              TPTR = SIPTR;           /*KEEP POINTER TO THIS ENTRY FOR
                                                      LINKING*/
103     1    2              SWAPIN.JOB# = DPTR->STATUS.JOB#;         /*ENTER DATA IN */
104     1    2              SWAPIN.SIZE = DPTR->STATUS.PART_SIZE;   /*SWAPIN ENTRY*/
105     1    2              SWAPIN.NEXT = NULL;
106     1    2              MEMSPACE = MEMSPACE - DPTR->STATUS.PART_SIZE;
107     1    2              DPTR->STATUS.ACT_IND = '1'B;    /*JOB IS NOW ACTIVE*/
108     1    2              DPTR->STATUS.TC_IND = '0'B;
                                                    /*JOB IS BLOCKED UNTIL FIRST PAGE IS
                                                      BROUGHT INTO CORE*/
109     1    2           END;
110     1    1           DPTR = DPTR->STATUS.NEXT;
```

```
STMT LEVEL NEST

 111     1     1        END;

                     /* NOW CHOOSE JOB TO BE RUN - EITHER PREVIOUS JOB OR A NEW ONE */
 112     1              IF (INDEX >= 10) & (INDEX <= 12) & (NXTJOB ¬= 0)
 113     1              THEN DO;                      /*JOB RUN PREVIOUSLY IS STILL RUNNABLE -
                                                        REASSIGN IT WITH REMAINING TIMESLICE*/
 114     1     1           RTIME = TIME - NEXTJOB->STATUS.BEG_TIME;
 115     1     1           TSLICE = TSLICE - RTIME;
 116     1     1           NEXTJOB->STATUS.BEG_TIME = TIME;
                                                      /*RESET BEGINNING OF RUN INTERVAL*/
 117     1     1        END;
 118     1              ELSE DO;                      /*PREVIOUS JOB WAS GIVEN ITS FULL
                                                        ALLOTMENT OR IS NOW BLOCKED - CHOOSE
                                                        A NEW JOB*/
 119     1     1        NEXTJOB = NULL;
 120     1     1        DO WHILE (NEXTJOB = NULL);


 121     1     2            IF (KPTR->STATUS.TC_IND = '1'B)&(KPTR->STATUS.ACT_IND = '1'B)
 122     1     2            THEN DO;                  /*NEXT JOB IS READY AND ACTIVE*/
 123     1     3               NEXTJOB = KPTR;        /*JOB CHOSEN*/
 124     1     3               NXTJOB = NEXTJOB->STATUS.JOB#;
                                                      /*DESIGNATE THIS AS NEXT JOB TO BE RUN*/
 125     1     3               TSLICE = 50000;        /*ASSIGN STANDARD TIME SLICE*/
 126     1     3               NEXTJOB->STATUS.BEG_TIME = TIME;
                                                      /*INITIALIZE BEGINNING OF RUN INTERVAL*/
 127     1     3            END;
 128     1     2            KPTR = KPTR->STATUS.NEXT;         /*UPDATE POINTER TO NEXT JOB*/
 129     1     2            IF KPTR->STATUS.JOB# = 0
 130     1     2            THEN IF ZERO_FLAG = '0'B
 131     1     2               THEN DO;
 132     1     3                  KPTR = KPTR->STATUS.NEXT;
                                                      /*SKIP JOB ZERO UNTIL ALL OTHER JOBS
                                                        HAVE BEEN TRIED.*/
 133     1     3                  ZERO_FLAG = '1'B;
                                                      /*SET FLAG ON SO JOB ZERO IS CHOSEN IF
                                                        SCAN COMES AROUND AGAIN.*/
 134     1     3               END;
 135     1     2         END;
 136     1     1         END;
 137     1           END SCHED;
```

```
STMT LEVEL NEST
                  /* *****              PREEMPTIVE SCHEDULER       '        ***** */
                  /******************************************************************/
   1              SCHED:PROCEDURE(JOBNUM,INDEX,TIME,SPACE,SJPTR,NXTJOB,TSLICE,IPTR,OPTR);

                  /* THIS SCHEDULER CHOOSES A JOB TO BE RUN FROM AMONG THE READY JOBS IN
                     CORE ACCORDING TO PRIORITY LEVEL.  JOBS ARE ACTIVATED ACCORDING TO
                     PRIORITY LEVEL AS SOON AS THERE IS ROOM FOR THEM, AND REMAIN IN
                     CORE UNTIL THEY TERMINATE. */

                  /* JOBNUM IS THE NUMBER OF THE JOB INVOLVED IN THE EVENT WHICH HAS
                     JUST OCCURRED IN THE SYSTEM, AND INDEX IDENTIFIES THIS EVENT.
                     INDEX VALUES HAVE MEANINGS AS FOLLOWS:
                        INDEX = -1  ---   INITIALIZATION
                                 0  ---   TERMINATION
                                 1  ---   PAGE REQUEST ISSUED
                                 2  ---   DISK OR TAPE REQUEST ISSUED
                                 4  ---   TIME SLICE RUNOUT
                                10  ---   JOB ARRIVAL
                                11  ---   PAGE REQUEST SATISFIED
                                12  ---   DISK OR TAPE REQUEST SATISFIED
                     SPACE GIVES THE TOTAL MEMORY SPACE AVAILABLE TO USER PROGRAMS, AND
                     TIME THE PRESENT TIME IN THE MODEL AS RECORDED ON THE SYSTEM CLOCK
                     IN THE MAIN ROUTINE.  SJPTR IS A POINTER TO THE FIRST ENTRY IN THE
                     JOB STREAM LIST MAINTAINED BY THE MAIN ROUTINE. NXTJOB IS THE
                     NUMBER OF THE JOB CHOSEN BY THE SCHEDULER TO BE PROCESSED NEXT,
                     AND TSLICE IS THE TIMESLICE ASSIGNED TO IT.  IPTR IS A POINTER TO
                     THE FIRST SWAPIN COMMAND ISSUED BY THE SCHEDULER AND OPTR IS A
                     POINTER TO THE FIRST SWAPOUT COMMAND. */

   2     1           DCL(JOBNUM,INDEX,SPACE) FIXED BIN(15),TIME FIXED BIN(31),SJPTR PTR;
   3     1           DCL (IPTR,OPTR)PTR, NXTJOB FIXED BIN(15), TSLICE FIXED BIN(31);
                                          /*ARGUMENTS*/

                  /* STRUCTURES FOR INDICATING SWAPIN AND SWAPOUT COMMANDS */

   4     1           DCL 1 SWAPOUT BASED(SOPTR),
                           2 JOB# FIXED BIN(15),
                           2 ATIME FIXED BIN(31),
                           2 NEXT POINTER;

   5     1           DCL 1 SWAPIN BASED(SIPTR),
                           2 JOB# FIXED BIN(15),
                           2 SIZE FIXED BIN(15),
                           2 NEXT POINTER;

                  /* VERSION OF JOB DESCRIPTIONS AVAILABLE TO SCHEDULER */

   6     1           DCL 1 SJOB BASED(SJPT),
                            2 JOB# FIXED BIN(15),        /*MATCHES UPPER PORTION OF*/
                            2 TYPE FIXED BIN(15),        /*JOB STREAM LIST ENTRIES*/
                            2 PRIORITY FIXED BIN(15),    /*USED BY MAIN ROUTINE*/
```

```
STMT LEVEL NEST

                          2 SIZE FIXED BIN(15),
                          2 NEXT PTR;

   7    1            DCL 1 STATUS BASED(STATPT),      /*STRUCTURE FOR KEEPING TRACK OF*/
                          2 JOB# FIXED BIN(15),       /*ACTIVE/INACTIVE AND TRAFFIC*/
                          2 ACT_IND BIT(1),           /*CONTROL STATUS, PARTITION SIZE*/
                          2 TC_IND BIT(1),            /*AND BEGINNING OF EACH RUN */
                          2 PART_SIZE FIXED BIN(15),  /*INTERVAL FOR EACH JOB*/
                          2 BEG_TIME FIXED BIN(31),
                          2 PRIORITY FIXED BIN(15),
                          2 NEXT PTR;

   8    1            DCL(FSTATPT,LSTATPT) POINTER STATIC;
                                          /*HOLD LOCATION OF INITIAL AND FINAL
                                            STATUS BLOCKS*/

   9    1            DCL(TPTR,DPTR) PTR, FOUND BIT(1), (NEXTJOB,KPTR) PTR STATIC;
  10    1            DCL MEMSPACE FIXED BIN(15) STATIC;   /*RUNNING RECORD OF FREE MEM.*/
  11    1            DCL ZERO_FLAG BIT(1) INITIAL('0'B);
  12    1            DCL RTIME FIXED BIN(31);
  13    1            DCL P_LEVEL FIXED BIN(15);   /*VARIABLE FOR KEEPING TRACK OF PRIOR-
                                                   ITY LEVELS FOR RUNNING AND ACTIVE
                                                   JOBS*/

                  /* PROCESS INPUT INFORMATION */

  14    1            IPTR = NULL;                 /*INITIALIZE SWAPIN POINTER TO NULL*/
  15    1            OPTR = NULL;                 /*INITIALIZE SWAPOUT POINTER TO NULL*/

  16    1            IF INDEX = -1
  17    1            THEN DO;                     /*FIRST CALL - INITIALIZE THINGS*/
  18    1     1         MEMSPACE = SPACE;         /*INITIALIZE RECORD OF FREE MEMORY*/
  19    1     1         TPTR = SJPTR;
  20    1     1         DPTR = NULL;
  21    1     1         DO WHILE (TPTR ¬= NULL);            /*CREATE A STATUS ENTRY*/
  22    1     2            ALLOCATE STATUS SET(STATPT);    /*FOR EACH JOB CURRENTLY*/
  23    1     2            STATUS.JOB# = TPTR->SJOB.JOB#;  /*IN THE SYSTEM*/
  24    1     2            STATUS.TC_IND = '1'B;      /*ALL JOBS INITIALLY READY*/
  25    1     2            STATUS.ACT_IND = '1'B;       /*ALL JOBS INITIALLY ACTIVE*/
  26    1     2            STATUS.PART_SIZE = (TPTR->SJOB.SIZE + 1)/2;
                                          /*INITIAL PARTITION SIZE IS HALF OF
                                            TOTAL SIZE*/
  27    1     2            STATUS.PRIORITY = TPTR->SJOB.PRIORITY;
                                          /*RECORD PRIORITY LEVEL FOR LATER USE*/
  28    1     2            MEMSPACE = MEMSPACE - STATUS.PART_SIZE;
                                          /*KEEP TRACK OF HOW MUCH MEMORY IS STILL
                                            FREE*/
  29    1     2            IF DPTR = NULL       /*PERFORM LINKING*/
  30    1     2            THEN FSTATPT = STATPT;
  31    1     2            ELSE DPTR->STATUS.NEXT = STATPT;
```

```
STMT LEVEL NEST

  32     1    2             DPTR = STATPT;
  33     1    2             TPTR = TPTR->SJOB.NEXT;
  34     1    2          END;
  35     1    1          LSTATPT = DPTR;           /*KEEP LOCATION OF FINAL BLOCK*/
  36     1    1          LSTATPT->STATUS.ACT_IND = '0'B;
                                                   /*LAST JOB GENERATED IS INACTIVE*/
  37     1    1          MEMSPACE = MEMSPACE + LSTATPT->STATUS.PART_SIZE;
                                                   /*CORRECT COUNT OF TOTAL FREE MEMORY
                                                     FOR INACTIVE JOB*/
  38     1    1          LSTATPT->STATUS.NEXT = FSTATPT;    /*MAKE LIST CIRCULAR*/
  39     1    1          KPTR = FSTATPT->STATUS.NEXT;  /*THIS POINTER HOLDS A PLACE IN
                                                             THE JOB CHAIN*/
  40     1    1          OPTR = NULL;              /*NO SWAPOUTS DONE BY THIS SCHEDULER*/
  41     1    1       END;

  42     1            ELSE IF INDEX = 0
  43     1            THEN DO;                     /*A JOB HAS TERMINATED*/
  44     1    1          TPTR = FSTATPT->STATUS.NEXT;
  45     1    1          DPTR = FSTATPT;
  46     1    1          FOUND = '0'B;
  47     1    1          DO WHILE (FOUND = '0'B);        /*SEARCH FOR STATUS ENTRY*/
  48     1    2             IF TPTR->STATUS.JOB# = JOBNUM
  49     1    2             THEN DO;               /*ENTRY FOUND*/
  50     1    3                MEMSPACE = MEMSPACE + TPTR->STATUS.PART_SIZE;
                                                   /*RETURN CORE TO FREE AREA*/
  51     1    3                DPTR->STATUS.NEXT = TPTR->STATUS.NEXT;
                                                   /*ADJUST LINKING OF STATUS CHAIN*/
  52     1    3                IF LSTATPT = TPTR
  53     1    3                THEN LSTATPT = DPTR;   /*RESET POINTER TO LAST ENTRY*/
  54     1    3                FREE TPTR->STATUS;     /*DELETE THIS ENTRY*/
  55     1    3                FOUND = '1'B;
  56     1    3             END;
  57     1    2             ELSE DO;               /*KEEP LOOKING FOR PROPER ENTRY*/
  58     1    3                DPTR = TPTR;
  59     1    3                TPTR = TPTR->STATUS.NEXT;
  60     1    3             END;
  61     1    2          END;
  62     1    1       END;

  63     1            ELSE IF (INDEX=1)|(INDEX=2)|(INDEX=11)|(INDEX=12)
  64     1            THEN DO;                     /*I/O REQUEST ISSUED OR SATISFIED*/
  65     1    1          TPTR = FSTATPT;
  66     1    1          FOUND = '0'B;
  67     1    1          DO WHILE (FOUND = '0'B);  /*SEARCH FOR DESCRIPTION OF JOB IN
                                                         QUESTION*/
  68     1    2             IF TPTR->STATUS.JOB# = JOBNUM
  69     1    2             THEN DO;               /*PROPER ENTRY FOUND*/
  70     1    3                IF INDEX >= 11
  71     1    3                THEN TPTR->STATUS.TC_IND = '1'B;
                                                   /*REQUEST SATISFIED - JOB IS READY*/
```

```
STMT LEVEL NEST

  72      1     3              ELSE TPTR->STATUS.TC_IND = '0'B;
                                                 /*REQUEST ISSUED - JOB IS BLOCKED*/
  73      1     3              FOUND = '1'B;
  74      1     3           END;
  75      1     2           ELSE TPTR = TPTR->STATUS.NEXT;    /*KEEP LOOKING*/
  76      1     2        END;
  77      1     1     END;

  78      1           ELSE IF INDEX = 10
  79      1           THEN DO;                   /*NEW ARRIVAL - GENERATE STATUS ENTRY*/
  80      1     1        ALLOCATE STATUS SET(STATPT);
  81      1     1        STATUS.JOB# = JOBNUM;    /*INITIALIZE VALUES FOR NEW ENTRY*/
  82      1     1        STATUS.ACT_IND = '0'B;
  83      1     1        STATUS.TC_IND = '1'B;
  84      1     1        FOUND = '0'B;
  85      1     1        TPTR = SJPTR;            /*INITIALIZE POINTER TO JOB STREAM LIST*/
  86      1     1        DO WHILE (FOUND = '0'B);        /*FIND JOB DESCR. FOR SIZE*/
  87      1     2           IF TPTR->SJOB.JOB# = JOBNUM
  88      1     2           THEN FOUND = '1'B;    /*PROPER ENTRY FOUND*/
  89      1     2           ELSE TPTR = TPTR->SJOB.NEXT;      /*KEEP LOOKING*/
  90      1     2        END;
  91      1     1        STATUS.PART_SIZE =(TPTR->SJOB.SIZE+1)/2;
  92      1     1        STATUS.PRIORITY = TPTR->SJOB.PRIORITY;
                                                 /*RECORD PRIORITY LEVEL FOR LATER USE*/
  93      1     1        LSTATPT->STATUS.NEXT = STATPT;  /*LINK NEW ENTRY INTO CHAIN*/
  94      1     1        LSTATPT = STATPT;
  95      1     1        STATUS.NEXT = FSTATPT;
  96      1     1     END;

                   /* SWAP AS MANY JOBS AS POSSIBLE INTO CORE*/

  97      1           DPTR= FSTATPT->STATUS.NEXT;     /*INITIALIZE POINTER TO STATUS LIST*/
  98      1           P_LEVEL = 11;
  99      1           DO WHILE (DPTR¬= FSTATPT);    /*FIND LOWEST PRIORITY LEVEL AMONG
                                                       ACTIVATABLE JOBS*/
 100      1     1        IF (DPTR->STATUS.ACT_IND = '0'B)&(MEMSPACE>= DPTR->STATUS.
                            PART_SIZE)&(DPTR->STATUS.TC_IND = '1'B)&(P_LEVEL >
                            DPTR->STATUS.PRIORITY)
 101      1     1           THEN P_LEVEL = DPTR->STATUS.PRIORITY;
 102      1     1           DPTR = DPTR->STATUS.NEXT;
 103      1     1     END;
 104      1           DO WHILE (P_LEVEL <= 10);     /*SWAP IN AS MANY JOBS AS POSSIBLE*/
 105      1     1        DPTR = FSTATPT->STATUS.NEXT;
 106      1     1        DO WHILE (DPTR ¬= FSTATPT);
 107      1     2           IF (DPTR->STATUS.ACT_IND = '0'B)&(MEMSPACE>=DPTR->STATUS.
                               PART_SIZE)&(DPTR->STATUS.TC_IND = '1'B)&(P_LEVEL = DPTR->
                               STATUS.PRIORITY)
 108      1     2           THEN DO;              /*SWAP IN THIS JOB*/
 109      1     3           ALLOCATE SWAPIN;
 110      1     3           IF IPTR = NULL
```

```
STMT LEVEL NEST

 111     1     3           THEN IPTR = SIPTR;    /*INITIALIZE POINTER TO FIRST SWAPIN*/
 112     1     3           ELSE IPTR->SWAPIN.NEXT = SIPTR;     /*LINK SWAPINS*/
 113     1     3           IPTR = SIPTR;         /*KEEP POINTER TO THIS ENTRY FOR
                                                   LINKING*/
 114     1     3           SWAPIN.JOB# = DPTR->STATUS.JOB#;        /*ENTER DATA IN */
 115     1     3           SWAPIN.SIZE = DPTR->STATUS.PART_SIZE;   /*SWAPIN ENTRY*/
 116     1     3           SWAPIN.NEXT = NULL;
 117     1     3           MEMSPACE = MEMSPACE - DPTR->STATUS.PART_SIZE;
 118     1     3           DPTR->STATUS.ACT_IND = '1'B;    /*JOB IS NOW ACTIVE*/
 119     1     3           DPTR->STATUS.TC_IND = '0'B;
                                                 /*JOB IS BLOCKED UNTIL FIRST PAGE IS
                                                   BROUGHT INTO CORE*/
 120     1     3           END;
 121     1     2           DPTR = DPTR->STATUS.NEXT;
 122     1     2        END;
 123     1     1        P_LEVEL = P_LEVEL + 1;
 124     1     1     END;

                  /* NOW CHOOSE JOB TO BE RUN - EITHER PREVIOUS JOB OR A NEW ONE */
 125     1           P_LEVEL = 11;
 126     1           DPTR = FSTATPT->STATUS.NEXT;
 127     1           DO WHILE (DPTR ¬= FSTATPT);      /*FIND LOWEST PRIORITY LEVEL AMONG
                                                        RUNNABLE JOBS*/
 128     1     1        IF (DPTR->STATUS.TC_IND = '1'B)&(DPTR->STATUS.ACT_IND = '1'B)&
                           (DPTR->STATUS.PRIORITY < P_LEVEL)
 129     1     1        THEN P_LEVEL = DPTR->STATUS.PRIORITY;
 130     1     1        DPTR = DPTR->STATUS.NEXT;
 131     1     1     END;
 132     1           IF (INDEX >= 10) & (INDEX <= 12) & (NXTJOB ¬= 0) & (P_LEVEL =
                        NEXTJOB->STATUS.PRIORITY)
 133     1           THEN DO;                      /*JOB RUN PREVIOUSLY IS STILL RUNNABLE -
                                                     AND IS STILL HIGHEST PRIORITY RUNNABLE
                                                     JOB - REASSIGN IT WITH REMAINING
                                                     TIMESLICE*/
 134     1     1        RTIME = TIME - NEXTJOB->STATUS.BEG_TIME;
 135     1     1        TSLICE = TSLICE - RTIME;
 136     1     1        NEXTJOB->STATUS.BEG_TIME = TIME;
                                                   /*RESET BEGINNING OF RUN INTERVAL*/
 137     1     1     END;
 138     1           ELSE DO;                      /*PREVIOUS JOB WAS GIVEN ITS FULL
                                                     ALLOTMENT OR IS NOW BLOCKED OR IS NO
                                                     LONGER HIGHEST PRIORITY RUNNABLE JOB -
                                                     CHOOSE A NEW JOB TO BE RUN*/
 139     1     1     IF P_LEVEL = 11
 140     1     1     THEN DO;                      /*NO JOBS ARE ELIGIBLE TO BE RUN - RUN
                                                     JOB ZERO*/
 141     1     2        NXTJOB = 0;
 142     1     2        TSLICE = 50000;
 143     1     2     END;
 144     1     1     ELSE DO;                      /*THERE IS AT LEAST ONE ELIGIBLE JOB -
```

```
STMT LEVEL NEST

                                                  CHOOSE ONE*/
 145    1    2        NEXTJOB = NULL;
 146    1    2        DO WHILE (NEXTJOB = NULL);

 147    1    3           IF (KPTR->STATUS.TC_IND = '1'B)&(KPTR->STATUS.ACT_IND = '1'B)
                            &(KPTR->STATUS.PRIORITY = P_LEVEL)
 148    1    3           THEN DO;               /*NEXT JOB IS READY AND ACTIVE AND
                                                  HIGHEST PRIORITY*/
 149    1    4              NEXTJOB = KPTR;      /*JOB CHOSEN*/
 150    1    4              NXTJOB = NEXTJOB->STATUS.JOB#;
                                                /*DESIGNATE THIS AS NEXT JOB TO BE RUN*/
 151    1    4              TSLICE = 50000;      /*ASSIGN STANDARD TIME SLICE*/
 152    1    4              NEXTJOB->STATUS.BEG_TIME = TIME;
                                                /*INITIALIZE BEGINNING OF RUN INTERVAL*/
 153    1    4           END;
 154    1    3           KPTR = KPTR->STATUS.NEXT;        /*UPDATE POINTER TO NEXT JOB*/
 155    1    3           IF KPTR->STATUS.JOB# = 0
 156    1    3           THEN KPTR = KPTR->STATUS.NEXT;     /*SKIP JOB ZERO*/
 157    1    3        END;
 158    1    2        END;
 159    1    1        END;
 160    1          END SCHED;
```

```
STMT LEVEL NEST
                    /* *****              DEACTIVATING SCHEDULER              ***** */
                    /*****************************************************************/
   1                SCHED:PROCEDURE(JOBNUM,INDEX,TIME,SPACE,SJPTR,NXTJOB,TSLICE,IPTR,OPTR);

                    /* THIS SCHEDULER CHOOSES A JOB TO BE RUN FROM AMONG THE READY JOBS IN
                       CORE IN A ROUND ROBIN FASHION.  JOBS ARE ACTIVATED IN THE ORDER IN
                       WHICH THEY ARRIVED AT THE SYSTEM AS SOON AS THERE IS ROOM FOR
                       THEM.  JOBS ARE DEACTIVATED WHENEVER THEY DO DISK OR TAPE I/O AND
                       ARE ELIGIBLE TO BE REACTIVATED WHEN THEIR I/O IS COMPLETE. */

                    /* JOBNUM IS THE NUMBER OF THE JOB WHICH HAS BEEN RUNNING JUST
                       PRIOR TO THIS CALL TO THE SCHEDULER.  INDEX INDICATES THE CAUSE
                       OF THE TERMINATION OF ITS PROCESSING.  INDEX VALUES HAVE
                       MEANINGS AS FOLLOWS:
                          INDEX = -1   ---   INITIALIZATION
                                   0   ---   TERMINATION
                                   1   ---   PAGE REQUEST ISSUED
                                   2   ---   DISK OR TAPE REQUEST ISSUED
                                   4   ---   TIME SLICE RUNOUT
                                  10   ---   JOB ARRIVAL
                                  11   ---   PAGE REQUEST SATISFIED
                                  12   ---   DISK OR TAPE REQUEST SATISFIED
                       SPACE GIVES THE TOTAL MEMORY SPACE AVAILABLE TO USER PROGRAMS, AND
                       TIME THE PRESENT TIME IN THE MODEL AS RECORDED ON THE SYSTEM CLOCK
                       IN THE MAIN ROUTINE.  SJPTR IS A POINTER TO THE FIRST ENTRY IN THE
                       JOB STREAM LIST MAINTAINED BY THE MAIN ROUTINE. NXTJOB IS THE
                       NUMBER OF THE JOB CHOSEN BY THE SCHEDULER TO BE PROCESSED NEXT,
                       AND TSLICE IS THE TIMESLICE ASSIGNED TO IT.  IPTR IS A POINTER TO
                       THE FIRST SWAPIN COMMAND ISSUED BY THE SCHEDULER AND OPTR IS A
                       POINTER TO THE FIRST SWAPOUT COMMAND. */

   2     1             DCL(JOBNUM,INDEX,SPACE) FIXED BIN(15),TIME FIXED BIN(31),SJPTR PTR;
   3     1             DCL (IPTR,OPTR)PTR, NXTJOB FIXED BIN(15), TSLICE FIXED BIN(31);
                                                 /*ARGUMENTS*/

                    /* STRUCTURES FOR INDICATING SWAPIN AND SWAPOUT COMMANDS */

   4     1             DCL 1 SWAPOUT BASED(SOPTR),
                             2 JOB# FIXED BIN(15),
                             2 ATIME FIXED BIN(31),
                             2 NEXT POINTER;

   5     1             DCL 1 SWAPIN BASED(SIPTR),
                             2 JOB# FIXED BIN(15),
                             2 SIZE FIXED BIN(15),
                             2 NEXT POINTER;

                    /* VERSION OF JOB DESCRIPTIONS AVAILABLE TO SCHEDULER */

   6     1             DCL 1 SJOB BASED(SJPT),
                              2 JOB# FIXED BIN(15),        /*MATCHES UPPER PORTION OF*/
```

```
STMT LEVEL NEST

                              2 TYPE FIXED BIN(15),         /*JOB STREAM LIST ENTRIES*/
                              2 PRIORITY FIXED BIN(15),     /*USED BY MAIN ROUTINE*/
                              2 SIZE FIXED BIN(15),
                              2 NEXT PTR;

   7     1              DCL 1 STATUS BASED(STATPT),        /*STRUCTURE FOR KEEPING TRACK OF*/
                              2 JOB# FIXED BIN(15),        /*ACTIVE/INACTIVE AND TRAFFIC*/
                              2 ACT_IND BIT(1),            /*CONTROL STATUS, PARTITION SIZE*/
                              2 TC_IND BIT(1),             /*AND BEGINNING OF EACH RUN */
                              2 PART_SIZE FIXED BIN(15),   /*INTERVAL FOR EACH JOB*/
                              2 BEG_TIME FIXED BIN(31),
                              2 NEXT PTR;

   8     1              DCL(FSTATPT,LSTATPT) POINTER STATIC;
                                                /*HOLD LOCATION OF INITIAL AND FINAL
                                                  STATUS BLOCKS*/

   9     1              DCL(TPTR,DPTR) PTR, FOUND BIT(1), (NEXTJOB,KPTR) PTR STATIC;
  10     1              DCL MEMSPACE FIXED BIN(15) STATIC;   /*RUNNING RECORD OF FREE MEM.*/
  11     1              DCL ZERO_FLAG BIT(1) INITIAL('0'B);
  12     1              DCL PTIME FIXED BIN(31);

                      /* PROCESS INPUT INFORMATION */

  13     1              IPTR = NULL;                    /*INITIALIZE SWAPIN POINTER TO NULL*/
  14     1              OPTR = NULL;                    /*INITIALIZE SWAPOUT POINTER TO NULL*/

  15     1              IF INDEX = -1
  16     1              THEN DO;                        /*FIRST CALL - INITIALIZE THINGS*/
  17     1      1          MEMSPACE = SPACE;            /*INITIALIZE RECORD OF FREE MEMORY*/
  18     1      1          TPTR = SJPTR;
  19     1      1          DPTR = NULL;
  20     1      1          DO WHILE (TPTR ¬= NULL);               /*CREATE A STATUS ENTRY*/
  21     1      2             ALLOCATE STATUS SET(STATPT);        /*FOR EACH JOB CURRENTLY*/
  22     1      2             STATUS.JOB# = TPTR->SJOB.JOB#;      /*IN THE SYSTEM*/
  23     1      2             STATUS.TC_IND = '1'B;          /*ALL JOBS INITIALLY READY*/
  24     1      2             STATUS.ACT_IND = '1'B;          /*ALL JOBS INITIALLY ACTIVE*/
  25     1      2             STATUS.PART_SIZE = (TPTR->SJOB.SIZE + 1)/2;
                                                /*INITIAL PARTITION SIZE IS HALF OF
                                                  TOTAL SIZE*/
  26     1      2             MEMSPACE = MEMSPACE - STATUS.PART_SIZE;
                                                /*KEEP TRACK OF HOW MUCH MEMORY IS STILL
                                                  FREE*/
  27     1      2             IF DPTR = NULL          /*PERFORM LINKING*/
  28     1      2             THEN FSTATPT = STATPT;
  29     1      2             ELSE DPTR->STATUS.NEXT = STATPT;
  30     1      2             DPTR = STATPT;
  31     1      2             TPTR = TPTR->SJOB.NEXT;
  32     1      2          END;
  33     1      1          LSTATPT = DPTR;            /*KEEP LOCATION OF FINAL BLOCK*/
```

```
STMT LEVEL NEST

 34    1    1          LSTATPT->STATUS.ACT_IND = '0'B;
                                             /*LAST JOB GENERATED IS INACTIVE*/
 35    1    1          MEMSPACE = MEMSPACE + LSTATPT->STATUS.PART_SIZE;
                                             /*CORRECT COUNT OF TOTAL FREE MEMORY
                                               FOR INACTIVE JOB*/
 36    1    1          LSTATPT->STATUS.NEXT = FSTATPT;   /*MAKE LIST CIRCULAR*/
 37    1    1          KPTR = FSTATPT->STATUS.NEXT;  /*THIS POINTER HOLDS A PLACE IN
                                                         THE JOB CHAIN*/
 38    1    1          DPTR = NULL;              /*NO SWAPOUTS DONE BY THIS SCHEDULER*/
 39    1    1       END;

 40    1            ELSE IF INDEX = 0
 41    1            THEN DO;                     /*A JOB HAS TERMINATED*/
 42    1    1          TPTR = FSTATPT->STATUS.NEXT;
 43    1    1          DPTR = FSTATPT;
 44    1    1          FOUND = '0'B;
 45    1    1          DO WHILE (FOUND = '0'B);        /*SEARCH FOR STATUS ENTRY*/
 46    1    2             IF TPTR->STATUS.JOB# = JOBNUM
 47    1    2             THEN DO;               /*ENTRY FOUND*/
 48    1    3                MEMSPACE = MEMSPACE + TPTR->STATUS.PART_SIZE;
                                             /*RETURN CORE TO FREE AREA*/
 49    1    3                DPTR->STATUS.NEXT = TPTR->STATUS.NEXT;
                                             /*ADJUST LINKING OF STATUS CHAIN*/
 50    1    3                IF LSTATPT = TPTR
 51    1    3                THEN LSTATPT = DPTR;  /*RESET POINTER TO LAST ENTRY*/
 52    1    3                FREE TPTR->STATUS;    /*DELETE THIS ENTRY*/
 53    1    3                FOUND = '1'B;
 54    1    3             END;
 55    1    2             ELSE DO;               /*KEEP LOOKING FOR PROPER ENTRY*/
 56    1    3                DPTR = TPTR;
 57    1    3                TPTR = TPTR->STATUS.NEXT;
 58    1    3             END;
 59    1    2          END;
 60    1    1       END;

 61    1            ELSE IF (INDEX=1)|(INDEX=2)|(INDEX=11)|(INDEX=12)
 62    1            THEN DO;                     /*I/O REQUEST ISSUED OR SATISFIED*/
 63    1    1          TPTR = FSTATPT;
 64    1    1          FOUND = '0'B;
 65    1    1          DO WHILE (FOUND = '0'B);  /*SEARCH FOR DESCRIPTION OF JOB IN
                                                   QUESTION*/
 66    1    2             IF TPTR->STATUS.JOB# = JOBNUM
 67    1    2             THEN DO;               /*PROPER ENTRY FOUND*/
 68    1    3                IF INDEX >= 11
 69    1    3                THEN TPTR->STATUS.TC_IND = '1'B;
                                             /*REQUEST SATISFIED - JOB IS READY*/
 70    1    3                ELSE TPTR->STATUS.TC_IND = '0'B;
                                             /*REQUEST ISSUED - JOB IS BLOCKED*/
 71    1    3                FOUND = '1'B;
 72    1    3             END;
```

```
STMT LEVEL NEST

  73     1     2              ELSE TPTR = TPTR->STATUS.NEXT;    /*KEEP LOOKING*/
  74     1     2           END;
  75     1     1           IF INDEX = 2
  76     1     1           THEN DO;                      /*JOB HAS ISSUED A PERIPHERAL I/O
                                                           REQUEST - SWAP IT OUT*/
  77     1     2              ALLOCATE SWAPOUT;
  78     1     2              OPTR = SOPTR;              /*AT MOST ONE SWAPOUT ON ANY CALL*/
  79     1     2              SWAPOUT.JOB# = JOBNUM;
  80     1     2              SWAPOUT.NEXT = NULL;
  81     1     2              TPTR->STATUS.ACT_IND = '0'B;
                                                         /*NOTE THAT THIS JOB IS NO LONGER IN
                                                           CORE*/
  82     1     2              MEMSPACE = MEMSPACE + TPTR->STATUS.PART_SIZE;
                                                         /*ADJUST AMOUNT OF FREE MEMORY TO
                                                           REFLECT DEACTIVATION*/
  83     1     2           END;
  84     1     1        END;

  85     1              ELSE IF INDEX = 10
  86     1              THEN DO;                         /*NEW ARRIVAL - GENERATE STATUS ENTRY*/
  87     1     1           ALLOCATE STATUS SET(STATPT);
  88     1     1           STATUS.JOB# = JOBNUM;      /*INITIALIZE VALUES FOR NEW ENTRY*/
  89     1     1           STATUS.ACT_IND = '0'B;
  90     1     1           STATUS.TC_IND = '1'B;
  91     1     1           FOUND = '0'B;
  92     1     1           TPTR = SJPTR;                  /*INITIALIZE POINTER TO JOB STREAM LIST*/
  93     1     1           DO WHILE (FOUND = '0'B);          /*FIND JOB DESCR. FOR SIZE*/
  94     1     2              IF TPTR->SJOB.JOB# = JOBNUM
  95     1     2              THEN FOUND = '1'B;     /*PROPER ENTRY FOUND*/
  96     1     2              ELSE TPTR = TPTR->SJOB.NEXT;      /*KEEP LOOKING*/
  97     1     2           END;
  98     1     1           STATUS.PART_SIZE =(TPTR->SJOB.SIZE+1)/2;
  99     1     1           LSTATPT->STATUS.NEXT = STATPT;  /*LINK NEW ENTRY INTO CHAIN*/
 100     1     1           LSTATPT = STATPT;
 101     1     1           STATUS.NEXT = FSTATPT;
 102     1     1        END;

                    /* SWAP AS MANY JOBS AS POSSIBLE INTO CORE*/

 103     1              DPTR= FSTATPT->STATUS.NEXT;      /*INITIALIZE POINTER TO STATUS LIST*/
 104     1              DO WHILE (DPTR¬= FSTATPT);
                                                         /*SEARCH THROUGH THE JOB LIST FOR
                                                           INACTIVE JOBS THAT WILL FIT INTO FREE
                                                           CORE*/
 105     1     1           IF (DPTR->STATUS.ACT_IND = '0'B)&(MEMSPACE>= DPTR->STATUS.
                              PART_SIZE)&(DPTR->STATUS.TC_IND = '1'B)
 106     1     1           THEN DO;                      /*NEXT JOB IS READY AND WILL FIT - MAKE
                                                           ENTRIES TO SWAP IT IN*/
 107     1     2              ALLOCATE SWAPIN;
 108     1     2              IF IPTR = NULL
```

```
STMT LEVEL NEST

 109     1    2              THEN IPTR = SIPTR;   /*INITIALIZE POINTER TO FIRST SWAPIN*/
 110     1    2              ELSE IPTR->SWAPIN.NEXT = SIPTR;    /*LINK SWAPINS*/
 111     1    2              IPTR = SIPTR;        /*KEEP POINTER TO THIS ENTRY FOR
                                                    LINKING*/
 112     1    2              SWAPIN.JOB# = DPTR->STATUS.JOB#;        /*ENTER DATA IN */
 113     1    2              SWAPIN.SIZE = DPTR->STATUS.PART_SIZE;  /*SWAPIN ENTRY*/
 114     1    2              SWAPIN.NEXT = NULL;
 115     1    2              MEMSPACE = MEMSPACE - DPTR->STATUS.PART_SIZE;
 116     1    2              DPTR->STATUS.ACT_IND = '1'B;   /*JOB IS NOW ACTIVE*/
 117     1    2              DPTR->STATUS.TC_IND = '0'B;
                                                  /*JOB IS BLOCKED UNTIL FIRST PAGE IS
                                                    BROUGHT INTO CORE*/
 118     1    2           END;
 119     1    1           DPTR = DPTR->STATUS.NEXT;
 120     1    1        END;

                    /* NOW CHOOSE JOB TO BE RUN - EITHER PREVIOUS JOB OR A NEW ONE */
 121     1             IF (INDEX >= 10) & (INDEX <= 12) & (NXTJOB ¬= 0)
 122     1             THEN DO;               /*JOB RUN PREVIOUSLY IS STILL RUNNABLE -
                                                    REASSIGN IT WITH REMAINING TIMESLICE*/
 123     1    1           RTIME = TIME - NEXTJOB->STATUS.BEG_TIME;
 124     1    1           TSLICE = TSLICE - RTIME;
 125     1    1           NEXTJOB->STATUS.BEG_TIME = TIME;
                                                  /*RESET BEGINNING OF RUN INTERVAL*/
 126     1    1        END;
 127     1             ELSE DO;               /*PREVIOUS JOB WAS GIVEN ITS FULL
                                                    ALLOTMENT OR IS NOW BLOCKED - CHOOSE
                                                    A NEW JOB*/
 128     1    1        NEXTJOB = NULL;
 129     1    1        DO WHILE (NEXTJOB = NULL);


 130     1    2           IF (KPTR->STATUS.TC_IND = '1'B)&(KPTR->STATUS.ACT_IND = '1'B)
 131     1    2           THEN DO;               /*NEXT JOB IS READY AND ACTIVE*/
 132     1    3              NEXTJOB = KPTR;      /*JOB CHOSEN*/
 133     1    3              NXTJOB = NEXTJOB->STATUS.JOB#;
                                                  /*DESIGNATE THIS AS NEXT JOB TO BE RUN*/
 134     1    3              TSLICE = 50000;      /*ASSIGN STANDARD TIME SLICE*/
 135     1    3              NEXTJOB->STATUS.BEG_TIME = TIME;
                                                  /*INITIALIZE BEGINNING OF RUN INTERVAL*/
 136     1    3           END;
 137     1    2           KPTR = KPTR->STATUS.NEXT;        /*UPDATE POINTER TO NEXT JOB*/
 138     1    2           IF KPTR->STATUS.JOB# = 0
 139     1    2           THEN IF ZERO_FLAG = '0'B
 140     1    2              THEN DO;
 141     1    3                 KPTR = KPTR->STATUS.NEXT;
                                                  /*SKIP JOB ZERO UNTIL ALL OTHER JOBS
                                                    HAVE BEEN TRIED.*/
 142     1    3                 ZERO_FLAG = '1'B;
                                                  /*SET FLAG ON SO JOB ZERO IS CHOSEN IF
```

```
                                                   SCAN COMES AROUND AGAIN.*/
        3                END;
        2            END;
        1            END;
                 END SCHED;
```

APPENDIX C

SAMPLE OUTPUT PRODUCED BY THE MODEL

The sample output provided in this appendix was produced by the test run described in chapter three which used the preemptive scheduler and fifty pages of user memory. The output shown enclosed in starred boxes beginning on page 202 was produced by the TRACE routine. The boxes are used to set off this information from any output produced by the DEBUG routine and from any diagnostic print put out by the scheduler being used. The TRACE output shown on pages 202 through 205 was produced between the simulated times of 1,000,000 microseconds and 1,500,000 microseconds in this run, i.e. it covers the simulated time between 1.0 and 1.5 seconds.

The TRACE module is called each time an event occurs in the simulated system. When called, it prints out the current simulated time (in microseconds), the event which has occurred and the job involved in this event, and the response of the scheduler to this event. The scheduler response includes commands issued to indicate the job to be processed next and any jobs to be activated or deactivated. For instance, the first call to TRACE was issued at time 1,023,500 microseconds. At this time job #1 has its

peripheral I/O request completed and is now ready to be run again. The scheduler responds to this information by selecting job #1 to be assigned to the processor.

A study of a sequence of TRACE reports provides a fairly clear picture of the successive events occurring in the simulated system during model operation. Though no information about the jobs not directly involved in the various events is provided, one can make some inferences about the state of the system from the observed pattern of events and scheduler commands. For instance, the second TRACE report shows that the scheduler has selected job #0 to be processed, i.e. the system is to remain idle. This indicates that none of the active jobs are in a runnable state at this time. At time 1,080,840 microseconds a new job, job #6, arrives at the system, and this job is not immediately ordered to be activated. This indicates that there are enough jobs already in main memory to make it fairly full, not leaving enough room to activate this additional job. Consideration of the overall TRACE report shows that processing alternates between jobs #1 and #4 during the time covered by this scan. Since neither job #2 or #3 appears we can conclude that these two jobs have already terminated and have left the system. We also note that job #0 is assigned to the processor a good deal of the time, due to both jobs #1 and

#4 being blocked. Neither job #1 nor job #4 terminates during the period covered by the TRACE scan, and no other jobs can be activated while these jobs are in core.

The output produced by the TRACE routine provides an overview of the microscopic operation of the model. This represents a compromise between the summary figures produced by the accounting routine and the copiously detailed information provided by DEBUG. It is useful for such tasks as exploring the operation of the model to corroborate or investigate summary figures which seem counterintuitive. It produces enough output to enable the user to follow the course of the simulation without forcing him to wade through a great deal of possibly irrelevant information. In cases where the TRACE routine points up an apparent error or anomaly the DEBUG routine may be called to print out more detailed information about the system during the time interval in question.

Only a single DEBUG snapshot is reproduced here (pages 206 through 208) due to the large amount of output produced on each call to this module. The information represented by the sample output shown includes the following facts. On the most recent call to the scheduler job #9 was selected to be processed, and was assigned a timeslice of 50,000 microseconds. The System Event List shows the future

events scheduled in the system at the time job #9's processing was halted. It shows that job #9 will have a peripheral I/O request satisfied (event type 12) at time 5,107,177 microseconds. This indicates that the halt in the processing of this job was due to its issuing a peripheral I/O request. Other events in the System Event List are the completion of a peripheral I/O request for job #17 which will occur at time 5,123,767 microseconds, and the arrival (event type 10) of job #22 at time 5,287,906 microseconds.

The job descriptions shown indicate the state of the job stream in the simulated system after the halt in the processing of job #9. Each job presently in the system is represented in this list, and all the characteristics describing its present state are shown. For instance, we see that job #2 is of type one and priority level two. Its total size (SIZE) is fifty-three pages, and its working set size (WSS) is twenty-one pages. It will require a total of 338,539 microseconds of processing time (CPUTIME). Its entries for partition size, timeslice and number of pages in core are all zero, indicating that it has not yet been activated. This is not surprising since the scheduler being used here takes priority level into account in choosing jobs to be activated, and this job is not a high priority job. The value of -1 for its active indicator (ACTIVE_IND) denotes the fact that it

is inactive at the present time. Its traffic control status (T_C_STATUS) is READY. CPU time until next page fault (PAGETIME) and CPU time until next peripheral I/O request (DSTPTIME) are given values when a job is activated; they have value zero here since this job has never been active. The CPU time still required to complete the job's processing (TERMTIME) is 338,539 microseconds, the same as the value of CPUTIME. This is as it should be since no processing has yet been done on this job to cause its time-remaining figure to be decreased.

Now consider the description of job #9, the job which has just been processed. It is of type one and priority level two. This is the same priority level as job #2, but job #9 is considerably smaller than job #2 in total size. Its working set size is larger than that of job #2, but since the scheduler has only the total size figures to use in making its scheduling decisions the relative working set sizes have no effect on activation choices. Job #9's total CPU time requirement is 377,755 microseconds. It has been assigned a partition size of twenty-three pages, and a timeslice of 50,000 microseconds. It has eleven pages in core at present (#_PAGES_IN_CORE). Its ACTIVE_IND value of 46,675 indicates that it is now active and has received 46,675 microseconds of processing time since its last

activation. Its traffic control status is BLOCKED because the job has just issued a peripheral I/O request and is now waiting for it to be completed. The processing time until it issues its next pagefault is 3,197 microseconds, and it will issue another peripheral I/O request after 11,597 microseconds of processing. It requires another 331,080 microseconds of processing for completion.

The descriptions of the other jobs may be interpreted in the same manner as the two described above. An interesting thing to note about the job stream shown here is the absence of descriptions for some of the jobs. The list is maintained and displayed in order of increasing job number. Running down the list, we find that jobs #1, #4, #7 and #11 do not appear. This indicates that these jobs have already terminated and left the system. The last item of information provided on each call to the DEBUG module is the simulated time at which the snapshot was taken, which in this case was 5,076,508 microseconds.

Careful study of the output provided by the DEBUG module provides a clear and comprehensive picture of the state of the simulated system at a given instant of time. Comparison of a sequence of DEBUG reports shows the microscopic processes occurring in the simulated system in great detail. Such information is invaluable in locating errors

in the logic of the model or exploring in detail unexpected behavior observed at the macroscopic level. The sheer volume of information generated, however, makes it expensive and time-consuming to perform this kind of detailed inspection.

The summary data output by the accounting routine (ACCNT) for this run is shown on pages 209 through 212. Since this data has already been discussed in chapter three it will not be analyzed in detail here. Most of the statistics produced are self-explanatory. The report is divided into four sections, each printed on a separate page. The first section gives overall figures on system activity and performance. The second describes the average characteristics of the jobs generated on this run, and the third gives data on the various activities of the scheduler used. The last section provides data on the behavior of the jobs under this scheduling scheme and the response of the simulated system to the demands of these jobs.

```
********************************************************************************
* AT TIME  1023500 JOB    1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).   *
* SCHEDULER SELECTS JOB    1 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1036123 JOB    1 ISSUES A PERIPHERAL I/O REQUEST.                             *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1055978 JOB    4 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).   *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1058931 JOB    4 INCURS A PAGE FAULT.                                         *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1066428 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).              *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1073344 JOB    4 INCURS A PAGE FAULT.                                         *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1080840 JOB    6 ENTERS THE SYSTEM.                                           *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1080974 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).              *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1088444 JOB    4 INCURS A PAGE FAULT.                                         *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                             *
********************************************************************************

********************************************************************************
* AT TIME  1090123 JOB    1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).   *
* SCHEDULER SELECTS JOB    1 TO BE RUN NEXT.                                             *
********************************************************************************
```

```
*******************************************************************************************
* AT TIME  1094827 JOB     4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).              *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1104112 JOB     1 ISSUES A PERIPHERAL I/O REQUEST.                             *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1109925 JOB     4 ISSUES A PERIPHERAL I/O REQUEST.                             *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1154925 JOB     0 RUNS OUT ITS TIMESLICE.                                      *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1168825 JOB     4 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).    *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1175233 JOB     4 INCURS A PAGE FAULT.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1183994 JOB     4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).              *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1189379 JOB     1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).    *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1191808 JOB     4 ISSUES A PERIPHERAL I/O REQUEST.                             *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1200833 JOB     1 INCURS A PAGE FAULT.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                             *
*******************************************************************************************

*******************************************************************************************
* AT TIME  1207850 JOB     1 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).              *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                             *
*******************************************************************************************
```

```
********************************************************************************
* AT TIME  1215256 JOB    1 ISSUES A PERIPHERAL I/O REQUEST.                    *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1265034 JOB    4 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).  *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1265215 JOB    4 INCURS A PAGE FAULT.                                *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1271966 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).     *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1281799 JOB    4 INCURS A PAGE FAULT.                                *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1289849 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).     *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1293493 JOB    4 ISSUES A PERIPHERAL I/O REQUEST.                    *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1323312 JOB    4 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).  *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1330661 JOB    4 INCURS A PAGE FAULT.                                *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1338397 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).     *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************

********************************************************************************
* AT TIME  1345941 JOB    1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).  *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                    *
********************************************************************************
```

```
* AT TIME  1346[illegible]12 JOB    4 ISSUES A PERIPHERAL I/O REQUEST.                                *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                                        *

* AT TIME  1364015 JOB    1 ISSUES A PERIPHERAL I/O REQUEST.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                                        *

* AT TIME  1397794 JOB    1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).               *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                                        *

* AT TIME  1409977 JOB    1 ISSUES A PERIPHERAL I/O REQUEST.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                                        *

* AT TIME  1425131 JOB    4 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).               *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                                        *

* AT TIME  1429978 JOB    4 INCURS A PAGE FAULT.                                                     *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                                        *

* AT TIME  1436566 JOB    4 ENTERS THE READY STATE (PAGE REQUEST COMPLETE).                          *
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                                        *

* AT TIME  1446119 JOB    4 ISSUES A PERIPHERAL I/O REQUEST.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                                        *

* AT TIME  1487962 JOB    1 ENTERS THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).               *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                                        *
```

```
**********          DEBUG OUTPUT          **********

SCHEDULER COMMANDS FOR THIS ITERATION:

CURRENT JOB =    9     TIMESLICE =  50000



SYSTEM EVENT LIST (AFTER RUN INTERVAL):

    JOB NUMBER              TYPE                 TIME
        9                    12                5107177
       17                    12                5123767
       22                    10                5287906


JOB DESCRIPTIONS (AFTER RUN INTERVAL):
    JOB#                     0
    TYPE                     0   PRIORITY              0   SIZE                0   WSS               0
    CPUTIME          999999999   PARTITION SIZE        0   TIMESLICE       50000   #_CORE_PAGES      0
    ACTIVE_IND         3281505   T_C_STATUS        READY
    PAGETIME         996718494   DSTPTIME      996718494   TERMTIME    996718494

    JOB#                     2
    TYPE                     1   PRIORITY              2   SIZE               53   WSS              21
    CPUTIME             338539   PARTITION SIZE        0   TIMESLICE           0   #_CORE_PAGES      0
    ACTIVE_IND              -1   T_C_STATUS        READY
    PAGETIME                 0   DSTPTIME              0   TERMTIME       338539

    JOB#                     3
    TYPE                     1   PRIORITY              3   SIZE               55   WSS              23
    CPUTIME             377755   PARTITION SIZE        0   TIMESLICE           0   #_CORE_PAGES      0
    ACTIVE_IND              -1   T_C_STATUS        READY
    PAGETIME                 0   DSTPTIME              0   TERMTIME       377755

    JOB#                     5
    TYPE                     1   PRIORITY              2   SIZE               51   WSS              30
    CPUTIME             519295   PARTITION SIZE        0   TIMESLICE           0   #_CORE_PAGES      0
    ACTIVE_IND              -1   T_C_STATUS        READY
    PAGETIME                 0   DSTPTIME              0   TERMTIME       519295

    JOB#                     6
    TYPE                     1   PRIORITY              3   SIZE               45   WSS              30
    CPUTIME             519295   PARTITION SIZE        0   TIMESLICE           0   #_CORE_PAGES      0
    ACTIVE_IND              -1   T_C_STATUS        READY
    PAGETIME                 0   DSTPTIME              0   TERMTIME       519295
```

JOB#	8						
TYPE	1	PRIORITY	3	SIZE	50	WSS	21
CPUTIME	338539	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	338539		

JOB#	9						
TYPE	1	PRIORITY	2	SIZE	45	WSS	23
CPUTIME	377755	PARTITION SIZE	23	TIMESLICE	50000	#_CORE_PAGES	11
ACTIVE_IND	46675	T_C_STATUS	BLOCKED				
PAGETIME	3197	DSTPTIME	11597	TERMTIME	331080		

JOB#	10						
TYPE	1	PRIORITY	1	SIZE	52	WSS	23
CPUTIME	377755	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	377755		

JOB#	12						
TYPE	1	PRIORITY	2	SIZE	50	WSS	22
CPUTIME	358071	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	358071		

JOB#	13						
TYPE	1	PRIORITY	2	SIZE	53	WSS	24
CPUTIME	397584	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	397584		

JOB#	14						
TYPE	1	PRIORITY	2	SIZE	53	WSS	30
CPUTIME	519295	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	519295		

JOB#	15						
TYPE	1	PRIORITY	1	SIZE	51	WSS	21
CPUTIME	338539	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	338539		

JOB#	16						
TYPE	1	PRIORITY	3	SIZE	47	WSS	27
CPUTIME	457882	PARTITION SIZE	0	TIMESLICE	0	#_CORE_PAGES	0
ACTIVE_IND	-1	T_C_STATUS	READY				
PAGETIME	0	DSTPTIME	0	TERMTIME	457882		

```
JOB#            17
TYPE             1    PRIORITY              1    SIZE             49    WSS            23
CPUTIME     377755    PARTITION SIZE       25    TIMESLICE     50000    #_CORE_PAGES   21
ACTIVE_IND  159184    T_C_STATUS      BLOCKED
PAGETIME     28051    DSTPTIME          15767    TERMTIME     218571

JOB#            18
TYPE             1    PRIORITY              2    SIZE             47    WSS            22
CPUTIME     358071    PARTITION SIZE        0    TIMESLICE         0    #_CORE_PAGES    0
ACTIVE_IND      -1    T_C_STATUS        READY
PAGETIME         0    DSTPTIME              0    TERMTIME     358071

JOB#            19
TYPE             1    PRIORITY              2    SIZE             51    WSS            21
CPUTIME     338539    PARTITION SIZE        0    TIMESLICE         0    #_CORE_PAGES    0
ACTIVE_IND      -1    T_C_STATUS        READY
PAGETIME         0    DSTPTIME              0    TERMTIME     338539

JOB#            20
TYPE             1    PRIORITY              3    SIZE             52    WSS            27
CPUTIME     457882    PARTITION SIZE        0    TIMESLICE         0    #_CORE_PAGES    0
ACTIVE_IND      -1    T_C_STATUS        READY
PAGETIME         0    DSTPTIME              0    TERMTIME     457882

JOB#            21
TYPE             1    PRIORITY              3    SIZE             45    WSS            22
CPUTIME     358071    PARTITION SIZE        0    TIMESLICE         0    #_CORE_PAGES    0
ACTIVE_IND      -1    T_C_STATUS        READY
PAGETIME         0    DSTPTIME              0    TERMTIME     358071


CLOCK =                    5076508

**************************************************
```

```
*************************************************************************
* SUMMARY STATISTICS DESCRIBING THE BEHAVIOR OF THE SIMULATED SYSTEM *
*************************************************************************
```

OVERALL STATISTICS:

AVERAGE NUMBER OF JOBS IN SYSTEM: 36.15

AVERAGE NUMBER OF JOBS IN CORE: 1.98

PAGE SIZE USED: 4096 MEMORY UNITS.

TOTAL AMOUNT OF CORE SPACE AVAILABLE TO USER PROGRAMS: 50 PAGES.

AVERAGE AMOUNT OF CORE ASSIGNED TO JOBS: 48.59 BLOCKS.

AVERAGE NUMBER OF PAGES ACTUALLY IN CORE: 34.70

CPU IDLE TIME: 62.49%

89 JOBS ARRIVED AT THE SYSTEM OVER A SCAN PERIOD OF 30027.520 MILLISECONDS, BEGINNING AT TIME 0.000 MILLISECONDS.

THESE JOBS WERE DESCRIBED BY THE FOLLOWING AVERAGE CHARACTERISTICS:

AVERAGE WORKING SET SIZE = 24.26 PAGES.

AVERAGE TOTAL SIZE = 49.48 PAGES.

AVERAGE CPUTIME REQUIRED = 403.399 MILLISECONDS.

TYPE DISTRIBUTION:

TYPE 1 -	100.00%
TYPE 2 -	0.00%
TYPE 3 -	0.00%
TYPE 4 -	0.00%
TYPE 5 -	0.00%
TYPE 6 -	0.00%

PRIORITY LEVEL DISTRIBUTION:

LEVEL 1 -	10.11%
LEVEL 2 -	35.96%
LEVEL 3 -	53.93%
LEVEL 4 -	0.00%
LEVEL 5 -	0.00%
LEVEL 6 -	0.00%
LEVEL 7 -	0.00%
LEVEL 8 -	0.00%
LEVEL 9 -	0.00%
LEVEL 10 -	0.00%

THE FOLLOWING FIGURES DESCRIBE THE BEHAVIOR OF THE SCHEDULER IN REGARD TO THESE JOBS:

DEACTIVATIONS: TOTAL NUMBER = 0

ACTIVATIONS: TOTAL NUMBER = 29

AVERAGE PARTITION SIZE ASSIGNED = 24.66 PAGES.

AVERAGE TIMESLICE ASSIGNED = 49.611 MILLISECONDS.

THE FOLLOWING FIGURES WERE COMPILED ON THE BEHAVIOR OF THESE JOBS:

THE AVERAGE PERCENTAGES OF TIME SPENT BY AN ACTIVE JOB IN EACH OF THE THREE TRAFFIC CONTROL STATES WAS AS FOLLOWS:

RUNNING:	18.95%
READY:	29.44%
BLOCKED:	51.61%

PAGEFAULTS:
TOTAL NUMBER = 721
AVERAGE PAGE WAIT = 7.636 MILLISECONDS.
AVERAGE TIME BETWEEN PAGE FAULTS = 41.647 MILLISECONDS.

DISK AND TAPE REQUESTS:
TOTAL NUMBER = 688
AVERAGE DISK OR TAPE WAIT = 59.398 MILLISECONDS.
AVERAGE TIME BETWEEN DISK OR TAPE REQUESTS = 43.645 MILLISECONDS.

TIMEOUTS:
TOTAL NUMBER = 70

TERMINATIONS:
TOTAL NUMBER = 27

AVERAGE TURNAROUND TIME: 7176.623 MILLISECONDS.

AVERAGE TURNAROUND TIME BY JOB TYPE:

TYPE 1 - 7176.621 MILLISECONDS.
TYPE 2 - --- MILLISECONDS.
TYPE 3 - --- MILLISECONDS.
TYPE 4 - --- MILLISECONDS.
TYPE 5 - --- MILLISECONDS.
TYPE 6 - --- MILLISECONDS.

AVERAGE TURNAROUND TIME BY PRIORITY LEVEL

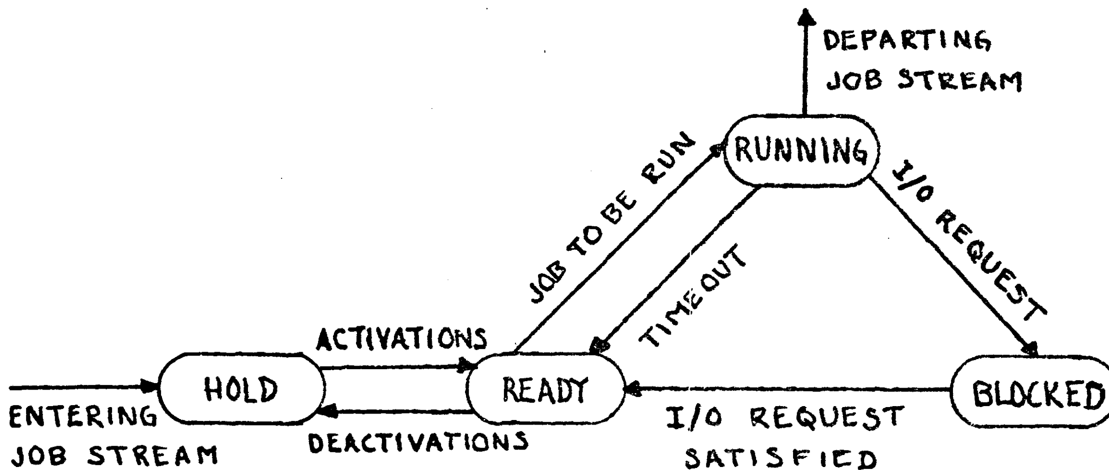
LEVEL 1 - 4064.740 MILLISECONDS.
LEVEL 2 - 8732.563 MILLISECONDS.
LEVEL 3 - --- MILLISECONDS.
LEVEL 4 - --- MILLISECONDS.
LEVEL 5 - --- MILLISECONDS.
LEVEL 6 - --- MILLISECONDS.
LEVEL 7 - --- MILLISECONDS.
LEVEL 8 - --- MILLISECONDS.
LEVEL 9 - --- MILLISECONDS.
LEVEL 10 - --- MILLISECONDS.

APPENDIX D

SAMPLE STUDENT ASSIGNMENT

The student is asked to write a process scheduler for a single processor demand paged computer system to meet certain specifications provided by the instructing staff (e.g. high overall throughput, fast response to high-priority jobs, etc.). This scheduler is to be coded in PL/1 and will be run under the supervision of a calling procedure which simulates the environment of a multiprogramming system, providing the scheduler with information about the characteristics and behavior of jobs in a simulated job stream running in the simulated environment. This information is described in detail below. Based on this information the scheduler issues commands which are carried out by the calling program. A report on the performance of the scheduler as measured by various statistics collected during the run is output at the end of each run. These statistics include figures such as average turnaround time and average system idle time as well as many more detailed figures.

The scheduler to be written is to perform the task of selecting jobs to be processed from among the jobs in the ready state as shown in the diagram below:



It also has the option of causing jobs to be moved back and forth between the hold and ready states via activate and deactivate commands. (Jobs in the ready state are resident in main memory; jobs in hold are not.) When a job is activated the scheduler must assign it a partition size. This represents the maximum number of 1K blocks of core which can be occupied at any one time by pages of that job. A job which is assigned a partition size of n blocks is assumed to occupy all of those blocks throughout the time it is in main memory. Thus the set of jobs which may be in memory at any one time is constrained by the limitation that the sum of the partition sizes of all such jobs must be less than or equal to the total amount of memory available to user jobs. It should be borne in mind that

activating and deactivating jobs incurs considerable overhead in terms of I/O resources and time needed to perform the transfer of data. These commands should not be used indiscriminately.

Description of the Simulator

The simulator which will call your scheduler starts out by setting up an initial list of jobs to be processed. These jobs are identified by number, and initially all but one of them (the one with the highest number) are in main memory and are partially processed. Each time it is called the scheduler must select a job to be processed, and it must assign this job a timeslice, which is the maximum amount of CPU time for which the job may be processed without being interrupted. In addition, the scheduler may issue commands to activate and deactivate jobs. Note that a job cannot be run if it is not in main memory. On some calls to the scheduler no jobs may be ready to be run. This situation may occur, for instance, when all active jobs (jobs in main memory) are blocked for I/O. When this is the case, or whenever the scheduler wishes for some other reason to let the processor remain idle, it selects job #0 as the next job to be processed. This is a dummy job which is run whenever no useful work is possible or desired. In

this case the timeslice assigned specifies the maximum length of time for which the processor is to remain idle without interruption.

When the scheduler is finished issuing commands it returns control to the simulator, which carries out those commands in the simulated environment. First any deactivations requested are performed, followed by any activations ordered. Then the job chosen to be processed next is run until one of the following events occurs:

- the job being processed terminates
- the job being processed goes blocked for I/O
- the job being processed runs out its timeslice
- a new job arrives at the system for processing
- another job enters the ready state (its I/O is complete and it is now ready to run)

When any of these events occurs processing is suspended and the scheduler is called to decide which job to process next. Note that this scheme allows for preemption of the current job in favor of any other job which has just become ready (i.e. was not ready at the time the current job was selected for processing). If preemption is not desired the current job is again assigned as the job to be processed. If the current job is no longer runnable (it is blocked or has

terminated), some other job must be chosen to be processed. This pattern is maintained for the duration of the run.

A note on activations: A job is considered to be in main memory and therefore runnable when one or more of its pages is in main memory. Activating a job is interpreted by the simulator simply as bringing its first page into core. Some amount of time is required for the I/O operations to perform this transfer of data. A job which the scheduler orders to be activated is then not immediately runnable, and cannot be selected as the next job to be processed at the same time it is chosen to be activated. When the page transfer is complete the activated job becomes runnable, and the scheduler is informed of this as described above (a job has entered the <u>ready</u> state).

<u>Description of the Scheduler</u>

The scheduler to be written must be called SCHED, and must <u>not</u> be declared with OPTIONS(MAIN). It is called with nine parameters which convey the following information:

1. The identification number of the job involved in the event which caused the halt in processing. (If the event was a time-out or a page request or peripheral I/O request issued by the running job,

this number is the number of the job which has been running; otherwise it is the number of some other job in the system. This is passed as a FIXED BIN(15) variable.

2. The cause of the immediately preceding halt in processing. This information is represented as a FIXED BIN(15) integer, and its values have meanings as follows:

   -1 - Initialization (This value is passed to the scheduler the first time it is called, when there is no current job. Variables in the scheduler which need initialization may be set up when this value is passed.)

   0 - The job being processed has terminated.

   1 - The job being processed has generated a page fault.

   2 - The job being processed has issued a disk or tape I/O request.

   4 - The job being processed has exceeded its time limit.

   10 - A new job has arrived at the system for processing.

   11 - Another job has had its page request satisfied and is now ready to be run.

12 - Another job has had its disk or tape I/O request satisfied and is now ready to be run.

3. The present time (in microseconds) in the simulated system as recorded on a clock maintained by the calling routine. This variable should be declared FIXED BIN(31).

4. The total main memory space available to user programs, expressed in pages. This variable has attributes FIXED BIN(15).

5. A pointer to the first description in a list containing descriptions of all jobs presently in the system. The format of these descriptions is outlined below. This parameter should be declared POINTER.

6. A variable in which the scheduler enters the identification number of the job to be processed next. This is a FIXED BIN(15) quantity.

7. A variable in which the scheduler enters the time-slice to be assigned to the job chosen to be processed next, in microseconds. This is a FIXED BIN(31) quantity.

8. A pointer to be set by the scheduler to point to the first entry in the chain of entries describing

activation commands. (The form of these entries is described below.) If there are no activations to be performed on a given call to the scheduler, no value need be assigned to this variable. This variable should be declared POINTER.

9. A pointer to be set by the scheduler to the first deactivation entry as described above for activations. This variable should be declared POINTER.

Job Descriptions

Job descriptions are stored as based structures which should be declared as follows:

```
DCL 1 SJOB BASED(SJPT),
      2 JOB# FIXED BIN(15),
      2 TYPE FIXED BIN(15),
      2 PRIORITY FIXED BIN(15),
      2 SIZE FIXED BIN(15),
      2 NEXT POINTER;
```

The first such description is accessed via the pointer to the job stream list which is passed as a parameter to the scheduler. Successive descriptions are linked together by the pointers in SJOB.NEXT. In the final description SJOB.NEXT has a value of NULL. On each call to the scheduler all

jobs currently in the system are represented in this list, which is maintained by the calling routine. The scheduler need not and should not make any changes to these descriptions.

The information contained in this structure is as follows:

JOB# - The identification number of the job being described. All jobs have nonnegative identification numbers, which are assigned in ascending order to the jobs as they enter the system.

TYPE - The type of the job (i.e. compilation, execution, file manipulation, etc.). Job type is represented as an integer between 1 and 6 with meanings of the various values as specified by the instructing staff.

PRIORITY - The priority level of the job, represented as an integer between 1 and 10, 1 being the highest priority.

SIZE - The total size of the job, in pages.

Command Structure

The structure in which the scheduler enters its activation commands should be declared as shown below:

```
DCL 1 ACTIVATION BASED(SIPTR),
      2 JOB# FIXED BIN(15),
      2 SIZE FIXED BIN(15),
      2 NEXT POINTER;
```

Each activation command issued by the scheduler is described by a separate copy of this structure. The use of each variable is as follows:

JOB# - The identification number of the job to be activated.

SIZE - The partition size to be assigned to this job when it is activated. This quantity is expressed as a number of pages.

NEXT - A pointer used to chain the activation commands issued on a given call together. It should point to the next copy of the structure in the chain, except in the case of the final entry, when it should have a value of NULL.

The corresponding structure for deactivations should be declared as:

```
DCL 1 DEACTIVATION BASED(SOPTR),
      2 JOB# FIXED BIN(15),
      2 ATIME FIXED BIN(31),
      2 NEXT POINTER;
```

The use of variables in this structure is described below:

JOB# - The identification number of the job to be deactivated.

ATIME - A variable used by the model routines but of no relevance to the scheduler. Its value should not be modified by the scheduler.

NEXT - A pointer for chaining deactivation entries together. Its use is analogous to that described above for NEXT in the activation structure.

<u>Data Compiled by the Scheduler</u>

In addition to the information explicitly provided by the calling routine, certain records should be kept by the scheduler itself if it is to operate in an efficient manner. For instance, the scheduler needs to know which jobs are in main memory at any given time, since only jobs which are in main memory may be chosen to be processed. Similarly, it should keep a record of which of the jobs in main memory are blocked and which are ready to run. It will also need to keep a record of how much memory space it has assigned to each job it has ordered to be activated so that when a job terminates it knows how much memory is available for bringing in new jobs.

Other information may also be of use in certain scheduling schemes, depending upon the aims of the particular scheduler in question. For example, it may be useful to keep track of the number of I/O requests issued by different jobs in order to determine which are I/O bound and which are compute bound. This information may then be used to maintain a balanced load of jobs in core. All information to be compiled by the scheduler must be deduced from the parameters passed to it by the calling routine.

In Case of Error

For each scheduler command there are certain values for the various command parameters which indicate legal commands and others which do not. If the identification number of a nonexistent job is entered in ACTIVATION.JOB# or DEACTIVATION.JOB# the command is ignored by the calling routine. If the partition size given in ACTIVATION.SIZE is nonpositive or is greater than the actual amount of free memory space remaining the command is again ignored. In each case a message is printed out explaining what has occurred. If the job number given as the next job to be processed specifies a job that is not in the system, is not in main memory, or is blocked, the command cannot be executed. In this case job #0 is

assigned to be processed and is given the same timeslice specified by the scheduler for the illegal job. A message is output indicating what was wrong with the choice of the job to be run. If a negative value is specified for the timeslice a default value of fifty milliseconds is used, and again a diagnostic message is provided.

Sample Scheduler Run

The following pages give a listing of a simple scheduler and the output produced when it is run under the simulator. Pages 227 through 230 show the code for the scheduler. This scheduler selects a new job to be run each time it is called. This choice is made in a simple round-robin manner. It does not perform any activations or deactivations, but simply operates on the set of jobs present in core at the beginning of the simulator run.

Pages 231 through 233 show the TRACE listing produced in the course of the run. Each starred box corresponds to one event occurring in the system. As described above, such an event may be the arrival or termination of a job, the issuing of a page request or a peripheral I/O request by the running job, the satisfaction of a page or peripheral I/O request issued by some other job, or the current job's exceeding its timeslice. Whenever such an event occurs the scheduler is

called. Each box identifies the event which has occurred, the time (in microseconds) at which it occurred, and the commands issued by the scheduler in response to this event. In the case of the scheduler used here the only command issued is the choice of the job to be processed next. In the case of a scheduler which orders activations and deactivations of jobs those commands are also shown.

The last four pages (234 through 237) show the summary information produced by the model. The total simulated time for this run was only 110 milliseconds, in order to make it feasible to reproduce the entire TRACE listing. For this reason the figures shown are not really representative of the behavior of this scheduler; however, they do illustrate the kinds of data produced by the model. Page 234 gives overall data and performance figures, and page 235 gives figures describing the jobs submitted to the simulated system. The number of jobs monitored as shown on the first line of this page represents the number of jobs which arrived for processing after the beginning of the run. Over a long simulation period this figure approximates the number of jobs which have passed through the system; for a run as short as this one the number is not valid. Page 236 gives a summary of the behavior of the scheduler being used, and page 237 summarizes the behavior of the jobs in the simulated system.

```
STMT LEVEL NEST
                      /*   SAMPLE SCHEDULER    */
  1                   SCHED:PROCEDURE(JOBNUM,INDEX,TIME,SPACE,SJPTR,NXTJOB,TSLICE,IPTR,OPTR);

                      /* SIMPLE ROUND ROBIN SCHEDULER - NO ACTIVATIONS OR DEACTIVATIONS*/

  2      1               DCL(JOBNUM,INDEX,SPACE) FIXED BIN(15),TIME FIXED BIN(31),SJPTR PTR;
  3      1               DCL (IPTR,OPTR)PTR, NXTJOB FIXED BIN(15), TSLICE FIXED BIN(31);
                                                   /*ARGUMENTS*/

                      /* STRUCTURES FOR INDICATING ACTIVATE AND DEACTIVATE COMMANDS (NOT USED
                         IN THIS SCHEDULER. */

  4      1               DCL 1 DEACTIVATION BASED(SOPTR),
                               2 JOB# FIXED BIN(15),
                               2 ATIME FIXED BIN(31),
                               2 NEXT POINTER;

  5      1               DCL 1 ACTIVATION BASED(SIPTR),
                               2 JOB# FIXED BIN(15),
                               2 SIZE FIXED BIN(15),
                               2 NEXT POINTER;

  6      1               DCL 1 SJOB BASED(SJPT),
                                2 JOB# FIXED BIN(15),          /*JOB DESCRIPTIONS*/
                                2 TYPE FIXED BIN(15),
                                2 PRIORITY FIXED BIN(15),
                                2 SIZE FIXED BIN(15),
                                2 NEXT PTR;

  7      1               DCL 1 STATUS BASED(STATPT),         /*STRUCTURE FOR KEEPING TRACK OF*/
                                2 JOB# FIXED BIN(15),        /*ACTIVE/INACTIVE AND TRAFFIC*/
                                2 ACT_IND BIT(1),            /*CONTROL STATUS, PARTITION SIZE*/
                                2 TC_IND BIT(1),
                                2 PART_SIZE FIXED BIN(15),
                                2 NEXT PTR;

  8      1               DCL FSTATPT POINTER STATIC;     /*HOLDS LOCATION OF INITIAL STATUS
                                                           BLOCK*/

  9      1               DCL(TPTR,DPTR) PTR, FOUND BIT(1), (NEXTJOB,KPTR) PTR STATIC;
 10      1               DCL MEMSPACE FIXED BIN(15) STATIC;
```

```
STMT LEVEL NEST

  11      1               DCL ZERO_FLAG BIT(1) INITIAL('0'B);
                                                    /*UTILTIY VARIABLES*/

                       /* PROCESS INPUT INFORMATION */

  12      1               IF INDEX = -1
  13      1               THEN DO:                  /*FIRST CALL - INITIALIZE THINGS*/
  14      1       1          MEMSPACE = SPACE:      /*INITIALIZE RECORD OF FREE MEMORY*/
  15      1       1          TPTR = SJPTR;
  16      1       1          OPTR = NULL;
  17      1       1          DO WHILE (TPTR ¬= NULL);              /*CREATE A STATUS ENTRY*/
  18      1       2             ALLOCATE STATUS SET(STATPT):       /*FOR EACH JOB CURRENTLY*/
  19      1       2             STATUS.JOB# = TPTR->SJOB.JOB#:     /*IN THE SYSTEM*/
  20      1       2             STATUS.TC_IND = '1'B;        /*ALL JOBS INITIALLY READY*/
  21      1       2             STATUS.ACT_IND = '1'B;         /*ALL JOBS INITIALLY ACTIVE*/
  22      1       2             STATUS.PART_SIZE = (TPTR->SJOB.SIZE + 1)/2:
                                                    /*INITIAL PARTITION SIZE IS HALF OF
                                                      TOTAL SIZE*/
  23      1       2             MEMSPACE = MEMSPACE - STATUS.PART_SIZE;
                                                    /*KEEP TRACK OF HOW MUCH MEMORY IS STILL
                                                      FREE*/
  24      1       2             IF OPTR = NULL      /*PERFORM LINKING*/
  25      1       2             THEN FSTATPT = STATPT:
  26      1       2             ELSE OPTR->STATUS.NEXT = STATPT;
  27      1       2             OPTR = STATPT:
  28      1       2             TPTR = TPTR->SJOB.NEXT;
  29      1       2          END:
  30      1       1          OPTR->STATUS.ACT_IND = '0'B;
                                                    /*LAST JOB GENERATED IS INACTIVE*/
  31      1       1          MEMSPACE = MEMSPACE + OPTR->STATUS.PART_SIZE:
                                                    /*CORRECT COUNT OF TOTAL FREE MEMORY
                                                      FOR INACTIVE JOB*/
  32      1       1          OPTR->STATUS.NEXT = FSTATPT;    /*MAKE LIST CIRCULAR*/
  33      1       1          KPTR = FSTATPT->STATUS.NEXT;  /*THIS POINTER HOLDS A PLACE IN
                                                              THE JOB CHAIN*/
  34      1       1          TPTR = NULL:           /*NO ACTIVATIONS DONE BY THIS SCHEDULER*/
  35      1       1          OPTR = NULL;           /*NO DEACTIVATIONS EITHER*/
  36      1       1       END;

  37      1               ELSE IF INDEX = 0
```

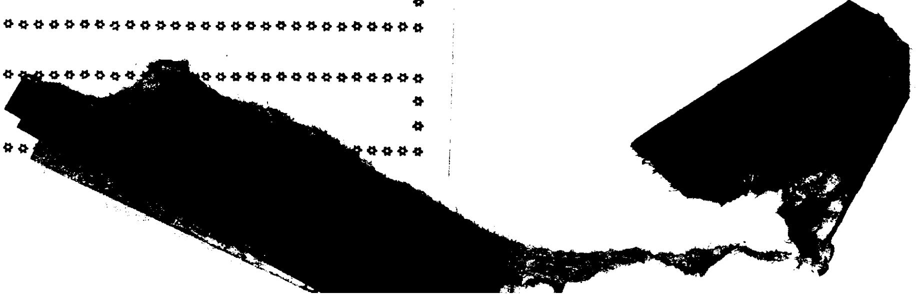
-228-

```
STMT LEVEL NEST

  38       1            THEN DO:                     /*A JOB HAS,TERMINATED*/
  39       1     1         TPTR = FSTATPT->STATUS.NEXT;
  40       1     1         OPTR = FSTATPT;
  41       1     1         FOUND = '0'B;
  42       1     1         DO WHILE (FOUND = '0'B);       /*SEARCH FOR STATUS ENTRY*/
  43       1     2            IF TPTR->STATUS.JOB# = JOBNUM
  44       1     2            THEN DO:                 /*ENTRY FOUND*/
  45       1     3               MEMSPACE = MEMSPACE + TPTR->STATUS.PART_SIZE:
                                                     /*RETURN CORE OCCUPIED BY THIS JOB TO
                                                       FREE CORE AREA*/
  46       1     3               OPTR->STATUS.NEXT = TPTR->STATUS.NEXT;
                                                     /*ADJUST LINKING OF STATUS CHAIN*/
  47       1     3               FREE TPTR->STATUS;     /*DELETE THIS ENTRY*/
  48       1     3               FOUND = '1'B;
  49       1     3            END;
  50       1     2            ELSE DO:                 /*KEEP LOOKING FOR PROPER ENTRY*/
  51       1     3               OPTR = TPTR:
  52       1     3               TPTR = TPTR->STATUS.NEXT:
  53       1     3            END:
  54       1     2         END:
  55       1     1      END:

  56       1            ELSE IF (INDEX=1)|(INDEX=2)|(INDEX=11)|(INDEX=12)
  57       1            THEN DO:                     /*I/O REQUEST ISSUED OR SATISFIED*/
  58       1     1         TPTR =,FSTATPT:
  59       1     1         FOUND = '0'B:
  60       1     1         DO WHILE (FOUND = '0'B);  /*SEARCH FOR DESCRIPTION OF JOB IN
                                                          QUESTION*/
  61       1     2            IF TPTR->STATUS.JOB# = JOBNUM
  62       1     2            THEN DO:                 /*PROPER ENTRY FOUND*/
  63       1     3               IF INDEX >= 11
  64       1     3               THEN TPTR->STATUS.TC_IND = '1'B:
                                                     /*REQUEST SATISFIED - JOB IS READY*/
  65       1     3               ELSE TPTR->STATUS.TC_IND = '0'B;
                                                     /*REQUEST ISSUED - JOB IS BLOCKED*/
  66       1     3               FOUND = '1'B;.
  67       1     3            END:
  68       1     2            ELSE TPTR = TPTR->STATUS.NEXT:    /*KEEP LOOKING*/
  69       1     2         END:
  70       1     1      END:
```

```
STMT LEVEL NEST

                         /* CHOOSE JOB TO BE PROCESSED NEXT */

 71     1                   NEXTJOB = NULL;
 72     1                   DO WHILE (NEXTJOB = NULL);


 73     1    1                 IF (KPTR->STATUS.TC_IND = '1'B)&(KPTR->STATUS.ACT_IND = '1'B)
 74     1    1                 THEN DO;                    /*NEXT JOB IS READY AND ACTIVE*/
 75     1    2                    NEXTJOB = KPTR;          /*JOB CHOSEN*/
 76     1    2                    NXTJOB = NEXTJOB->STATUS.JOB#;
                                                             /*DESIGNATE THIS AS NEXT JOB TO BE RUN*/
 77     1    2                    TSLICE = 50000;          /*ASSIGN STANDARD TIME SLICE*/
                                                             /*INITIALIZE BEGINNING OF RUN INTERVAL*/
 78     1    2                 END;
 79     1    1                 KPTR = KPTR->STATUS.NEXT;          /*UPDATE POINTER TO NEXT JOB*/
 80     1    1                 IF KPTR->STATUS.JOB# = 0
 81     1    1                 THEN IF ZERO_FLAG = '0'B
 82     1    1                    THEN DO;
 83     1    2                       KPTR = KPTR->STATUS.NEXT;
                                                             /*SKIP JOB ZERO UNTIL ALL OTHER JOBS
                                                               HAVE BEEN TRIED.*/
 84     1    2                       ZERO_FLAG = '1'B;
                                                             /*SET FLAG ON SO JOB ZERO IS CHOSEN IF
                                                               SCAN COMES AROUND AGAIN.*/
 85     1    2                    END;
 86     1    1              END;
 87     1                END SCHED;
```

```
*****************************************************************************************
* INITIAL ITERATION                                                                     *
* SCHEDULER SELECTS JOB    1 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME      2083 JOB    1 INCURS A PAGE FAULT.                                       *
* SCHEDULER SELECTS JOB    2 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME      2597 JOB    2 INCURS A PAGE FAULT.                                       *
* SCHEDULER SELECTS JOB    3 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME      3506 JOB    3 INCURS A PAGE FAULT.                                       *
* SCHEDULER SELECTS JOB    4 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME      6315 JOB    4 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB    5 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME      7713 JOB    5 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB    6 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME     13863 JOB    2 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).        *
* SCHEDULER SELECTS JOB    7 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME     15062 JOB    7 INCURS A PAGE FAULT.                                       *
* SCHEDULER SELECTS JOB    2 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME     15426 JOB    1 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).        *
* SCHEDULER SELECTS JOB    6 TO BE RUN NEXT.                                            *
*****************************************************************************************

*****************************************************************************************
* AT TIME     15659 JOB    3 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).        *
* SCHEDULER SELECTS JOB    1 TO BE RUN NEXT.                                            *
*****************************************************************************************
```

```
**********************************************************************
* AT TIME    19229 JOB    1 INCURS A PAGE FAULT.                     *
* SCHEDULER SELECTS JOB    2 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    26163 JOB    2 INCURS A PAGE FAULT.                     *
* SCHEDULER SELECTS JOB    3 TO BE RUN NEXT..                        *
**********************************************************************

**********************************************************************
* AT TIME    27270 JOB    7 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).    *
* SCHEDULER SELECTS JOB    6 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    28661 JOB    6 ISSUES A PERIPHERAL I/O REQUEST.         *
* SCHEDULER SELECTS JOB    7 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    31382 JOB    7 INCURS A PAGE FAULT.                     *
* SCHEDULER SELECTS JOB    3 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    32579 JOB    1 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).    *
* SCHEDULER SELECTS JOB    1 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    37181 JOB    1 INCURS A PAGE FAULT.                     *
* SCHEDULER SELECTS JOB    3 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    37458 JOB    3 INCURS A PAGE FAULT.                     *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    47111 JOB    2 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).    *
* SCHEDULER SELECTS JOB    2 TO BE RUN NEXT.                         *
**********************************************************************

**********************************************************************
* AT TIME    47215 JOB    2 ISSUES A PERIPHERAL I/O REQUEST.         *
* SCHEDULER SELECTS JOB    0 TO BE RUN NEXT.                         *
**********************************************************************
```



```
******************************************************************************************
* AT TIME     52189 JOB     1 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).       *
* SCHEDULER SELECTS JOB     1 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     54763 JOB     1 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     60783 JOB     7 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).       *
* SCHEDULER SELECTS JOB     7 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     61073 JOB     7 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     73567 JOB     3 RETURNS TO THE READY STATE (PAGE REQUEST COMPLETE).       *
* SCHEDULER SELECTS JOB     3 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     74320 JOB     3 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     86274 JOB     4 RETURNS TO THE READY STATE (PERIPHERAL I/O REQUEST COMPLETE).*
* SCHEDULER SELECTS JOB     4 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     93870 JOB     4 ISSUES A PERIPHERAL I/O REQUEST.                           *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME     99240 JOB     9 ENTERS THE SYSTEM.                                         *
* SCHEDULER SELECTS JOB     0 TO BE RUN NEXT.                                            *
******************************************************************************************

******************************************************************************************
* AT TIME    110650 JOB     6 ENTERS THE SYSTEM.                                         *
* RUN TERMINATES.                                                                        *
******************************************************************************************
```

```
**********************************************************************
* SUMMARY STATISTICS DESCRIBING THE BEHAVIOR OF THE SIMULATED SYSTEM *
**********************************************************************
```

OVERALL STATISTICS:

AVERAGE NUMBER OF JOBS IN SYSTEM: 8.07

AVERAGE NUMBER OF JOBS IN CORE: 7.00

PAGE SIZE USED: 1024 MEMORY UNITS.

TOTAL AMOUNT OF CORE SPACE AVAILABLE TO USER PROGRAMS: 100 PAGES.

AVERAGE AMOUNT OF CORE ASSIGNED TO JOBS: 96.00 BLOCKS.

AVERAGE NUMBER OF PAGES ACTUALLY IN CORE: 43.69

CPU IDLE TIME: 55.92%

THE FOLLOWING FIGURES WERE COMPILED ON THE BEHAVIOR OF THESE JOBS:

THE AVERAGE PERCENTAGES OF TIME SPENT BY A JOB IN EACH OF THE THREE TRAFFIC CONTROL STATES WAS AS FOLLOWS:

```
        RUNNING:    6.30%
        READY:     18.16%
        BLOCKED:   75.54%

PAGEFAULTS:         TOTAL NUMBER =     9
                    AVERAGE NUMBER PER JOB =     4.50
                    AVERAGE PAGE WAIT =  18.198 MILLISECONDS.
                    AVERAGE TIME BETWEEN PAGE FAULTS =  12.294 MILLISECONDS.

DISK AND
   TAPE REQUESTS:   TOTAL NUMBER =     8
                    AVERAGE NUMBER PER JOB =     4.00
                    AVERAGE DISK OR TAPE WAIT = 150.781 MILLISECONDS.
                    AVERAGE TIME BETWEEN DISK OR TAPE REQUESTS =  13.831 MILLISECONDS.

TIMEOUTS:           TOTAL NUMBER =     0
                    AVERAGE NUMBER PER JOB =   0.00
```

```
THE FOLLOWING FIGURES DESCRIBE THE BEHAVIOR OF THE SCHEDULER IN REGARD TO THESE JOBS:

DEACTIVATIONS:      TOTAL NUMBER =    0

ACTIVATIONS:        TOTAL NUMBER =    0

                    AVERAGE TIMESLICE ASSIGNED =  50.000 MILLISECONDS.
```

2 JOBS WERE MONITORED OVER A SCAN PERIOD OF 110.650 MILLISECONDS, BEGINNING AT TIME 0.000 MILLISECONDS.

THESE JOBS WERE DESCRIBED BY THE FOLLOWING AVERAGE CHARACTERISTICS:

AVERAGE WORKING SET SIZE = 17.00 PAGES.

AVERAGE TOTAL SIZE = 33.50 PAGES.

AVERAGE CPUTIME REQUIRED = 378.813 MILLISECONDS.

TYPE DISTRIBUTION:

TYPE 1 -	0.00%
TYPE 2 -	50.00%
TYPE 3 -	50.00%
TYPE 4 -	0.00%
TYPE 5 -	0.00%
TYPE 6 -	0.00%

PRIORITY LEVEL DISTRIBUTION:

LEVEL 1 -	0.00%
LEVEL 2 -	100.00%
LEVEL 3 -	0.00%
LEVEL 4 -	0.00%
LEVEL 5 -	0.00%
LEVEL 6 -	0.00%
LEVEL 7 -	0.00%
LEVEL 8 -	0.00%
LEVEL 9 -	0.00%
LEVEL10 -	0.00%

BIBLIOGRAPHY

1. Brinch Hansen, Per," Short-Term Scheduling in Multi-programming Systems", Proceedings Third ACM Symposium on Operating System Principles, Stanford University, October, 1971, pp. 101-105.

2. Browne, J.C., Lan, Jean and Baskett, Forest, "The Interaction of Multi-programming Job Scheduling and CPU Scheduling", Proceedings AFIPS Fall Joint Computer Conference, 1972, pp. 13-21.

3. Saltzer, Jerome H., "Traffic Control in a Multiplexed Computer System", MAC-TR-30 (Thesis), July, 1966.

4. Coffman, Edward G. Jr. and Kleinrock, Leonard, "Computer Scheduling Methods and Their Countermeasures", Proceedings AFIPS Spring Joint Computer Conference, 1968, pp. 11-21.

5. Oppenheimer, G. and Weizer, N., "Resource Management for a Medium Scale Time-Sharing Operating System", Communications of the ACM (11,5), May, 1968, pp.313-333.

6. Bryan, G. and Shemer, J., "The UTS Time-Sharing System: Performance Analysis and Instrumentation", Second ACM Symposium on Operating System Principles, October 20-22, 1969, Princeton University, pp. 147-158.

7. Arden, B. and Boettner, D., "Measurement and Performance of a Multiprogramming System", Second ACM Symposium on Operating System Principles, October 20-22, 1969, Princeton University, pp. 130-146.

8. Sherman, Stephen, Baskett, Forest and Browne, J.C., "Trace Driven Modeling and Analysis of CPU Scheduling in a Multi-programming System", ACM SIGOPS Workshop on Performance Evaluation, April 5-7, 1971, Harvard University, pp. 173-199.

9. DeMeis, W.M. and Weizer, N.,"Measurement and Analysis of a Demand Paging Time Sharing System", Proceedings ACM National Conference, 1969, pp. 201-216.

10. Losapio, N.S. and Bulgren, William G., "Simulation of Dispatching Algorithms in a Multiprogramming Environment", Proceedings ACM Annual Conference, 1972, pp. 903-913.

11. Calingaert, Peter, "System Performance Evaluation - Survey and Appraisal", Communications of the ACM (10,1), January, 1967, pp. 12-18.

12. Estrin, G., Muntz, R.R, and Uzgalis, R.C., "Modelling, Measurement and Computer Power", Proceedings AFIPS Spring Joint Computer Conference, 1972, pp. 725-738.

13. Lucas, Henry C., "Performance Evaluation and Monitoring", Computing Surveys (3,3), September, 1971, pp. 79-91.

14. McKinney, J.M., "A Survey of Analytical Time-Sharing Models", Computing Surveys (1,2), June, 1969, pp. 105-116.

15. Adiri, Igal, "A Note on Some Mathematical Models of Time-Sharing Systems", Journal of the ACM (18,4), October, 1971, pp. 611-615.

16. Gaver, D.P., "Probability Models for Multiprogramming Computer Systems", Journal of the ACM (14,3), July, 1967, pp. 423-438.

17. Kleinrock, Leonard, "Time-Shared Systems: A Theoretical Treatment", Journal of the ACM (14,2), April, 1967, pp. 212-261.

18. DeCegama, A., "A Methodology for Computer Model Building", Proceedings AFIPS Fall Joint Computer Conference, 1972, pp. 299-310.

19. Kimbleton, Stephen R., "Performance Evaluation - A Structured Approach", Proceedings AFIPS Spring Joint Computer Conference, 1972, pp. 411-416.

20. Fife, Dennis W., "An Optimization Model for Time-Sharing", Proceedings AFIPS Spring Joint Computer Conference, 1966, pp. 97-104.

21. Rasch, Philip, "A Queuing Theory Study of Round-Robin Scheduling of Time-Shared Computer Systems", Journal of the ACM (17,1), January, 1970, pp. 131-145.

22. Shedler, G.S., "A Cyclic-Queue Model of a Paging Machine", IBM Research, RC2814, March 25, 1970.

23. Shemer, Jack, "Some Mathematical Considerations of Time-Sharing Scheduling Algorithms", Journal of the ACM (14,2), April, 1967, pp. 262-272.

24. Shemer, Jack E. and Heying, Douglas W., "Performance Modelling and Empirical Measurement in a System Designed for Batch and Time-Sharing Users", Proceedings AFIPS Fall Joint Computer Conference, 1969, pp. 17-26.

25. Slutz, Donald R., "A Look at Paging Algorithms and Program Models", Proceedings Fifth Annual Princeton Conference on Information Sciences and Systems, 1971, pp. 432-436.

26. Blatny, J., Clark, S.R., and Rourke, T.A., "On the Optimization of Performance of Time-Sharing Systems by Simulation", Communications of the ACM (15,6), June, 1972, pp. 411-420.

27. Cheng, P.S., "Trace-Driven System Modeling", IBM Systems Journal (8,4), 1969, pp. 280-289.

28. MacDougall, M.H., "Computer System Simulation: An Introduction", Computing Surveys (2,3), September, 1970, pp. 191-209.

29. Nielsen, Norman R., "An Approach to the Simulation of a Time-Sharing System", Proceedings AFIPS Fall Joint Computer Conference, 1967, pp. 419-428.

30. Nielsen, Norman R., "The Simulation of Time Sharing Systems", Communications of the ACM (10,7), July, 1967, pp. 397-412.

31. Bell, Thomas E., "Objectives and Problems in Simulating Computers", Proceedings AFIPS Fall Joint Computer Conference, 1972, pp. 287-297.

32. Lynch, W.C., "Operating System Performance", Communications of the ACM (15,7), July, 1972, pp. 579-585.

33. Nutt, G.J., "Evaluation Nets for Computer System Performance Analysis", Proceedings AFIPS Fall Joint Computer Conference, 1972, pp. 279-286.

34. Seaman, P.H. and Soucy, R.C., "Simulating Operating Systems", IBM Systems Journal No. 4, 1969, pp. 264-279.

35. Boote, W.P., Clark, S.R., and Rourke, T.A., "Simulation of a Paging Computer System", The Computer Journal (15,1), February, 1972, pp. 51-57.

36. Fine, Gerald H. and McIsaac, Paul V., "Simulation of a Time-Sharing System", Management Science (12,6), February, 1966, pp. B-180-B-194.

37. Lehman, Meir M. and Rosenfeld, Jack L., "Performance of a Simulated Multiprogramming System", Proceedings AFIPS Fall Joint Computer Conference, 1968, pp. 1431-1442.

38. MacDougall, M.H., "Simulation of an ECS-Based Operating System", Proceedings AFIPS Spring Joint Computer Conference, 1967, pp. 735-741.

39. Morganstein, S.J., Winograd, J., and Herman, R., "SIM/61: A Simulation Measurement Tool for a Time-Shared, Demand Paging Operating System", ACM SIGOPS Workshop on Performance Evalution, April 5-7, 1971, Harvard University, pp. 142-172.

40. Noe, J.D. and Nutt, G.J., "Validation of a Trace-Driven CDC 6400 Simulation", Proceedings AFIPS Spring Joint Computer Conference, 1972, pp. 749-757.

41. Rehmann, Sandra L. and Gangwere, Sherbie G., "A Simulation Study of Resource Management in a Time-Sharing System", Proceedings AFIPS Fall Joint Computer Conference, 1968, pp. 1411-1430.

42. Scherr, Allan, "An Analysis of Time-Shared Computer Systems", Thesis, Massachusetts Institute of Technology, June, 1965.

43. Schwetman, H.D. and Brown, J.C., "An Experimental Study of Computer System Performance", Proceedings ACM Annual Conference, 1972, pp. 693-703.

44. Winograd, J., Morganstein, S.J., and Herman, R., "Simulation Studies of a Virtual Memory, Time-Shared, Demand Paging Operating System", Third ACM Symposium on Operating System Principles, October, 1971, Stanford University, pp. 149-155.

45. Denning, Peter, "A Statistical Model for Console Behavior in Multiuser Computers", Communications of the ACM (11,9), September, 1968, pp. 605-612.

46. Kimbleton, Stephen and Moore, Charles, "A Probabilistic Framework for System Performance Evaluation", ACM SIGOPS Workshop on Performance Evaluation, April 5-7, 1971, Harvard University, pp. 337-361.

47. Sussman, Joseph, 1.154 Course Notes, Department of Civil Engineering, Massachusetts Institute of Technology, February, 1973.

48. Fine, Gerald, Jackson, Calvin, and McIsaac, Paul, "Dynamic Program Behavior under Paging", Proceedings of the ACM National Meeting, 1966, pp. 223-228.

49. Denning, Peter, "The Working Set Model for Program Behavior", Communications of the ACM (11,5), May, 1968, pp. 323-333.

50. Coffman, E.G. and Wood, R.C., "Interarrival Statistics for Time Sharing Systems", Communications of the ACM (9,7), July, 1966, pp. 500-503.

51. Fuchs, E. and Jackson, P.E., "Estimates of Distributions of Random Variables for Certain Computer Communications Traffic Models", Communications of the ACM (13,12), December, 1970, pp. 752-757.

52. Martin, F.F., Computer Modeling and Simulation, Wiley, New York, copyright 1968.

53. Ryan, Leo, Private Communication, January, 1973.

54. Steinberg, Joseph R., Private Communication, January, 1973.

55. The Bulletin, Massachusetts Institute of Technology Information Processing Center, Number 107, October, 1972.

56. Control Program 67 - Cambridge Monitor System, January, 1970.

57. Baylis, M.H.J., Fletcher, D.G., and Howarth, D.J., "Paging Studies made on the I.C.T. ATLAS Computer" Proceedings IFIPS Conference, 1968, pp. 831-836.

58. Hatfield, D.J., "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance", IBM Journal of Research and Development, January, 1972, pp. 58-66.

59. Joseph, M., "An Analysis of Paging and Program Behaviour", The Computer Journal (13,1), February, 1970, pp. 48-54.

60. Coffman, E.G. and Varian, L.C., "Further Experimental Data on the Behavior of Programs in a Paging Environment", Communications of the ACM (11,7), July, 1968, pp. 471-474.

61. Madnick, Stuart and Donovan, John, Operating Systems (draft), copyright 1972.

62. Sekino, Akira, Private Communication, April, 1972.