1968

Automatic allocation of digital computer storage resources for time-sharing

Frank Gerald Soltis
Iowa State University

69-9896

SOLTIS, Frank Gerald, 1940-
AUTOMATIC ALLOCATION OF DIGITAL COMPUTER STORAGE RESOURCES FOR TIME-SHARING.

Iowa State University, Ph.D., 1968
Engineering, electrical

AUTOMATIC ALLOCATION OF DIGITAL COMPUTER STORAGE RESOURCES FOR TIME-SHARING

by

Frank Gerald Soltis

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of
The Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Major Subject: Electrical Engineering

Approved:

In Charge of Major Work

Head of Major Department

Dean of Graduate College

Iowa State University
Ames, Iowa

1968

TABLE OF CONTENTS

INTRODUCTION

"Time-sharing" is the term used to describe a computer system in which each user in turn is given a certain amount of operating time (i.e., a time slice). In this way the system's resources can be effectively "shared" among many users. Most of the current time-sharing systems operate in a multiprogramming environment, wherein a number of programs reside in main core at the same time. A problem which soon becomes apparent is that, for a large number of users and large programs, not all programs can fit into main core at the same time. The solution to this problem is to provide the system with a large capacity auxiliary storage device (i.e., secondary storage) which may be considerably slower than main core but which is able to efficiently transfer blocks of information to and from main core. These blocks of information are called "pages".

The concept of paging was first introduced on the ATLAS computer, which is described in a paper by Kilburn _et_ _al_. (8). Paging can be defined as the transmission to and from secondary storage, the relocation, and the execution of programs on a paged basis. Relocation refers to the translation of a program's addresses into the actual locations in main core. This is needed because it is not possible to put a page into the same main core location every time it is read in. Whenever a program refers to some address which is not in main core, it is necessary for the system to locate the page in secondary storage which contains that address and read in that page. It may be necessary to write out to secondary storage some page in main core in order to make room for the page being read in. Thus, pages are constantly being transferred to and from secondary storage in a paged time-sharing computer system.

Since the paging structure is unique to a time-sharing system, a number of design decisions have been encountered which are not present in the design of a conventional computer system. The selection and organization of secondary storage; the means of accomplishing the relocation operation; the development of algorithms for page replacement, page reading, and page writing; and numerous other decisions must be made in order to design the paging structure.

Because of the large number and complexity of these many design decisions, it is not surprising that many different paging structures for time-sharing systems have been proposed. The performance of some of these proposed structures has been very good; others have not fared as well. The causes of the poor system performance of some systems and remedies for this poor performance have generated a great deal of interest among system designers. While it is obvious that the many design decisions do affect the performance of the time-sharing system, very little is known about how much of an effect any particular decision has on the total system performance.

This study will examine the behavior of a paged time-sharing system with the objective of determining those areas of the paging structure which offer the most opportunity for improvements in system performance. Because there is no universally accepted design for a paging structure, it will be necessary to select some particular system whose design is representative of current time-sharing philosophy and to analyze this system thoroughly. Modifications to the paging structure which improve the total system performance will be proposed and the generalized results found for this system will be extended to other systems.

Some means must be found to analyze the operation of a time-sharing system. The second chapter describes some of the analysis techniques which have previously been developed. The system selected to be studied and a description of the analysis technique chosen are also presented in this chapter.

In order to provide valid results for the system study, very careful consideration must be given to the selection of a test case. The determination of the test cases and the reasons behind their selection are given in the third chapter.

In the fourth chapter the effect of the paging devices on system performance is examined. The paging drums are shown to be able to deliver pages to the system at a rate which can maintain satisfactory system performance. Some of the major causes of poor system performance are pointed out in this chapter.

Those areas of the paging structure which degrade the system performance are studied in the fifth chapter. The proposed changes to these areas are given in detail and the effects of each change on the system behavior are presented. Each change is carefully examined on its own merits to determine how significant an improvement in system performance can be obtained.

The sixth chapter summarizes the results which have been obtained by modifications to the paging structure. These results are then generalized to determine which areas of the paging structure can offer the most opportunity for improvements in system performance. Additional areas for study are proposed and some concluding comments about time-sharing studies are given.

ANALYSIS OF TIME-SHARING COMPUTER SYSTEMS

This chapter is concerned with the selection of an analysis technique which can be used to investigate the operation of a time-sharing computer system. A survey of some of the studies that have been made is first presented, followed by a more detailed description of the technique selected. Basically, there are two means of analyzing the behavior of a time-sharing system: mathematical analysis and simulation.

Mathematical Analysis

In order to develop a mathematical model of a system, some structure must be assumed. A common model of a time-sharing system is taken to be an interconnection of queues and processes with a known stochastic flow of tasks between them. An arithmetic processor, a stored program, a data channel, and even a user are considered to be processes. On the other hand, a queue is regarded as a list of uncompleted tasks whose routes between processes are indistinguishable. Associated with each process is a service time, representing the length of time a task will occupy that process. Other factors affecting the flow between processes, such as arrival times, priority assignments, etc., must also be represented in the model. The state of the system at any time is described by the number of entries in the various queues.

The structure assumed is a fairly good representation of general purpose, time-sharing system and is the model derived from a Markov chain. Taken together with the assumed probability distributions for arrival and service times, this model can be analyzed by means of Markov process theory to predict system behavior. The objective of the analysis is to evaluate the probability distributions of the states. Once these distributions are known, other probablistic measures of interest, such queue lengths, processor utilization times, and through-put rates, can be determined. For a discussion of how these distributions are obtained for a Markov model, see the paper by Wallace and Rosenberg (18).

There have been many studies conducted to analyze computer systems using Markov models. One of the best studies was made by Smith (17). In his paper Smith describes a paged, time-sharing structure for which a queuing model was derived. The model did represent the system operations fairly well, but the requirement that the model remain a Markov process severely restricted its flexibility. However, this is one of the few studies which have attempted to analyze a paged structure.

Not all analyses concern themselves with the performance of the entire system. A paper by Fife (2) describes a study of job scheduling techniques used in a time-sharing system. Krishnamoorthi and Wood (9) explored methods which would aid in the selections of time slice intervals.

The papers described above, as well as many others, represent attempts to analyze a general purpose, time-sharing system using mathematical techniques. While none of them have completely achieved this goal, a number of them have made significant contributions. All suffer from a general lack of flexibility. Unfortunately, the assumptions made by many in proposing models, render their results totally useless for any practical system.

Simulation

The second means of analyzing the performance of time-sharing computer systems is simulation. An excellent review of computer system simulations is given in a paper by Huesmann and Goldberg (6). Some of the more important simulation studies are discussed in the following paragraphs.

Scherr (16) developed a very good simulation of the MAC system at MIT. The description of the system is through a series of statements and subroutines written in MAD (Michigan Algorithm Decoder) which allow the hardware configuration and operating system to be specified in as much or as little detail as desired. The job mix is generated according to desired statistics which are inputs to the simulator. The program also uses statistics gathering routines to obtain data on the job mix which outputs along with response times, through-put rates, queue lengths, overhead times, etc.

Fine and McIsaac (4) discuss a simulation of the System Development Corporation (SDC) Q-32 time-sharing system. Most of the important characteristics of the simulation, such as hardware configuration and operating system information, are described within the simulation program with only a few of them being input parameters. The job mix is specified statistically with probability distributions read in, and the simulator generates jobs according to these distributions. The output of the program is in the form of response times, overhead times, swap times, etc. The simulation did reflect the behavior of the SDC computer fairly well.

Nielsen (13) extended the results of Scherr and of Fine and McIsaac in order to produce a more general time-sharing simulator. His simulation program successfully models the IBM System/360 Model 67, but it requires some reprogramming to simulate most other systems.

Special simulation languages, such as GPSS and SIMSCRIPT, have been used to simulate various time-sharing systems, although these languages are not specifically designed to simulate only computer systems. There are simulation languages that have been designed specifically to model computer systems. IBM's CSS (Computer System Simulator) and LOMUSS (Lockheed's Multipurpose System Simulator) are two such languages.

The simulation efforts just described were undertaken in an attempt to simulate one particular time-sharing system. Ideally, a simulator would require only the input parameters to describe any system. Unfortunately, no such simulator exists or is likely to exist for some time due to the differences in hardware and operating systems of the various time-sharing systems. In spite of its drawbacks, a simulation study appears to be the most promising means of analysis of a time-sharing system.

Selection of an Analysis Technique

To analyze only the paging structure of a time-sharing system, it may be possible to use strictly mathematical techniques. However, the object of this study is to investigate the behavior of the entire system with respect to changes in the paging structure. For this reason, it is necessary to select some analysis technique which can be used to model the entire time-sharing system. Since simulation offers the most promising results, it will be used.

The selection of a system to analyze also presents some interesting problems. Selecting a general time-sharing system, which in itself is very difficult to define, raises the question of the validity of the simulation results with respect to any particular system. Also, the most successful simulations have been designed to simulate some particular system. Therefore, it appears that the most practical approach is to select some particular system which is representative of current time-sharing systems and to extend the results to other systems.

Ideally, the system selected should reflect the design used in most current time-sharing computer systems. It should be a well documented system with these documents readily available. And finally, there should be as many of these systems in use as possible, so that the behavior of the actual system is known. A system which meets all of these requirements is the IBM System/360 Model 67. Consequently, the Model 67 will be the system used.

Detailed descriptions of the Model 67 are given in papers by Gibson (5) and Comfort (1) as well as in an IBM publication (7). A brief description of this system is also given in the Appendix.

As a bonus for selecting the Model 67, the simulation by Nielsen, which has been shown to accurately reflect the behavior of this system, was available. A copy of this simulation program was obtained from Nielsen at the Stanford University Computation Center.

The simulation program finally used in this study was not the same as the one originally received. The original simulation apparently was designed to be as general as possible in order to simulate any paged, time-sharing system. To make the program simulate any particular system required some reprogramming. In order to simulate the structure desired for this study, a number of modifications had to be made to the original simulation program. A copy of the final version used is available at the Iowa State University Computation Center Library.

A Description of the Simulation Program

In this section a brief description of some of the salient features of the simulation program is presented. For a more detailed description see the original work by Nielsen (13) or his two other papers (14) and (15) which also describe his original work.

Levels of simulation studies

An important consideration for any simulation is to determine what level of system activity is to be simulated. Four levels of simulation studies are given in Table 1.

Table 1. Levels of simulation studies

Level	Size unit	Time unit
1	job	seconds
2	page	milliseconds
3	instruction	microseconds
4	bit	nanoseconds

The first level is the one which is of most interest to the system's user. Although the user is interested in statistics in terms of jobs and seconds, these units are too large to produce a meaningful simulation. The system designer is interested in simulations at the second level of system study. The simulation by Nielsen is at this level with basic units of a page and 0.1 milliseconds. The third level is of some interest in a study of the paging structure, but any simulation of system behavior using this level's basic units becomes very long and time consuming. Fortunately, independent studies, both analytical and simulated, can be made at this level; and the results obtained can be used as inputs to the second level simulation with very good results. The fourth level is of no particular interest in a system study.

Representation of jobs

One of the most important aspects of the simulation program has to do with the methods used to represent jobs in the system. The validity of the simulation depends in many ways on the means used to represent jobs.

A common technique used to represent jobs in a system is to input to the simulation the distributions for the length of time a job executes in any page and the next page a job will reference. Since the characteristics of a job can change as the job progresses, it is desirable to have different distributions for the various stages of a job (e.g. compile, link-edit and execute).

Nielsen represents each section of a job as a sequence of page references and supervisor services appropriately spaced by execution times. Several of these sequences can be linked together and/or repeated in any desired manner to represent an entire job. For example, the syntax checking and table building phase of compiling a one hundred statement job could be handled by describing the sequence of operations for two statements and then repeating this sequence fifty times.

These sequences must be generated by the system's user for every desired job type in the system and are then input to the simulation program as master sequences. In this way the master sequences represent prototype jobs for each different job type. While the construction of these prototype jobs can be an extremely involved process, it is a once only operation for the simulation's user.

The simulation program uses the prototype jobs to construct the sequences for each job in the system. At the time that each of these sequences is constructed from a prototype, the simulation program determines according to appropriate random distributions the number of repetitions to be used, the terminal user's "think" time for each terminal interaction, the next master sequence to be used, etc. Because of the random nature used to construct each job sequence, no two jobs constructed from the same prototype are exactly the same. Two jobs of the same type will have the same characteristics, but they will not be carbon copies of one another.

A special job description language was developed to describe the job sequences. This language consists of a set of fourteen instruction types. Eight of these instruction types appear in both the master sequence and the specific job sequences. These instruction types determine the behavior of a job during its simulated execution time. They are used to specify the execution time before a particular operation is to occur and to control page accessing, terminal interactions, I/O operations, etc. The six other instruction types are used only in the master sequence. They are used to build the specific job sequences from the master sequence.

The description of the job representation presented here has been necessarily brief. For a more detailed discussion, the reader is referred to the paper by Nielson (14) which has been almost entirely devoted to a description of the representation of jobs used in this simulation program.

Implementation

Nielsen implemented the simulation program in Fortran because he felt that there were no special simulation languages widely available. The source program consists of about 7000 statements organized into 31 subroutines each concerned with one particular aspect of the simulation. This modularity was used so that changes necessary to simulate other systems could be made as easily as possible.

In the simulation program one word is used to keep track of each page in the system whether it is a physical memory page or a virtual memory page. The word representing each page indicates the status of the page, the location of the page, and the task associated with the page. Queues in the system are represented by lists of words. To simulate a page on one of the queues in system, such as a page read queue, its word is attached to the list representing that queue.

Scheduling of events in the system is accomplished by the event calender queue. Entries are removed from this queue according to the earliest time of occurrence. A master clock is used to keep track of the system time in units of 0.1 milliseconds (i.e., the basic time unit). Statistics are gathered periodically by various routines during the simulation and are summarized at the end of the simulation to produce the output results.

Simulation inputs

Approximately 1500 data cards are used as inputs to the simulation program. The first set of data cards describe the monitor parameters. The monitor parameters include such things as the length of a time slice and the overhead times required to perform the many system functions. The next section is concerned with the equipment configuration. The names, quantities, capacities, transmission rates, rotational delays, seek times, etc. of the various devices in the system are given in this section. The next two sections contain the simulation run parameters and the data analysis parameters which are used to control the simulation and gather the statistics for the simulation's output. The terminal characteristics indicate which jobs are used in the system. Following this are the list of the prototype jobs and the list of instructions for the prototype jobs.

Simulation results

This section presents a description of the information which will be given in the table of results for each simulation run made for this study. It should be kept in mind that the simulation program produces much more information than can be included in the tables. Thus, the tables will present a concise listing of the most important simulation results.

Simulation times Two simulation times are given in the table of results. The first is the length of the initialization time in seconds. The simulation resets the statistics gathering procedure at the end of the initialization period. In this way, the problems of the startup and the initial transient of the model can be ignored. The second is the length of the simulation run in seconds after the statistics gathering begins.

CPU data The percentages of CPU time used for execution, overhead, and idle are given in the tables of results. Execution time is the total amount of system time during which one task in the system is executing instructions. Since only one task can be executing at a time, all other tasks must be idle. Overhead time is the total amount of system time used by the monitor to perform such functions as I/O scheduling, interrupt handling, etc. While the system is performing overhead functions, all tasks are idle. Idle time is the total amount of system time during which no work is being done by the system (i.e., all tasks are idle and no overhead functions are taking place). It is important not to confuse system idle time and task idle time. The idle time given in the tables is system idle time. The distribution of execution, overhead, and idle times indicates how much useful work is being performed by the system.

Response times The response time is the time between two consecutive time slices for a task. It is an indication of the amount of time required by the system to respond to a request by a user at a terminal. The average response time in seconds is given in the table of results along with the range of response times. This range gives an indication of the best and the worst response times for the system.

Paging rates The average number of pages per second transmitted to and from (i.e., write and read, respectively) secondary storage is given in the tables. Also, the average number of pages per second retrieved from core queues before being replaced is given. These values indicate the amount of paging activity taking place in the system.

Queue data   The average number of requests in the drum read and drum write queues is given for each simulation run. These values give some indication of the paging activity of the drums.

Device utilization   The percentage of total time during which the drums and the nonpaging disks are used is given for each simulation run.

SELECTION OF A TEST CASE

In order to simulate the operation of any proposed time-sharing system, it is necessary to carefully define the equipment configuration and job mix that will be used as inputs to the simulation. The simulation studies cited in the previous chapter have shown that the performance of a time-sharing system is very sensitive to the details of this definition; therefore, some rational means must be employed in the selection of test cases which will be used to compare the performance of various systems.

Clearly, it is neither possible nor desirable to exhaustively test every combination. An arbitrary selection of a "typical" configuration and job mix leaves the not-so-small problem of defining just what is typical. There are, however, certain guidelines which can be used to make a reasonable selection.

The selection of an equipment configuration is based more on economic considerations than on equipment availability. Although system performance is very much dependent upon this configuration, there are fewer undefined variables involved than there are with the job mix selection; thus a reasonable equipment configuration can be readily determined.

Selection of a job mix to be used as a test case for a time-sharing system simulation is a different matter. What may be a "typical" job mix for one installation may be totally foreign to another. It soon becomes evident that there is no such thing as a general job mix. Any particular job mix selected must be justified solely on its applicability to the environment at hand. Some guidelines which can be used to make a selection of a job mix are:

1. The job mix selected for the simulation should resemble a job mix found in the "real world". That is, the distribution of job types and lengths should be determined from a study of some actual installation.
2. The job mix should be stable with respect to order of jobs input to the simulation, starting time for each job, and length of the simulation runs. This means that after a suitable initialization period, the simulation process is time-stationary. Also, the simulation results should be insensitive to minor variations in the job mix.
3. More than one job mix may be needed to reflect the total system performance. Perhaps one mix is used for prime time operation (heavy conversational and light batch), and another mix is used for overnight operation (light conversational and heavy batch). Extremes in the job mixes may be one way to obtain the total performance.

After a stable job mix has been obtained, it is possible to use it with a number of simulations of proposed time-sharing systems and be reasonably certain that any variations in performance among the systems are significant. In other words, any variation is due to the system structure and not to the job mix.

Equipment Configuration

The equipment configuration selected for the test case will be made up of the hardware components available for the IBM System/360 Model 67. A study of an appropriate configuration for the Stanford University

Computation Center was used as a guide. The configuration selected is listed below.

1 CPU — The processing unit is initially the 2067, the CPU of the 360/67. It has the computational power of the model 65 computer with the hardware capability of dynamic address relocation.

3 Core Storage Units (2365) — Each unit has 262,144 bytes with a cycle time of 750 nanoseconds for an eight byte reference. Three of these units provide 166 nonsystem pages (1 page = 4096 bytes) and 26 system pages for the Model 67.

2 Paging Drums (2301) — Each drum has a capacity of 4,090,000 bytes or 900 pages (4½ pages per track). Transfer rate is 1,200,000 bytes per second and the rotational period is 17.2 milliseconds.

1 Disk (2314) — There are 8 modules/disk, each with a separate access mechanism. Each disk has a total of 207,014,000 bytes and a transfer rate of 312,000 bytes per second. The rotational period is 25 milliseconds, and the seek time is from 30 to 140 milliseconds.

4 Magnetic Tape Units — These are nine track units with a transfer rate of 60,000 bytes/second.

3 Line Printers (1403) — Each unit prints approximately 1100 lines/minute.

2 Card Reader — Punches (2540) - Each unit reads 1000 cards/minute and punches 300 cards/minute.

3 Selector Channels (2860) — High speed devices (drums and disks) are attached to the selector channels.

1 Multiplexor Channel — The slower I/O devices are attached to this channel.

Transmission Control Units — Each unit can control up to 31 remote terminals.

Terminals — As many terminals (teletypes) as required for the job load are attached to the system.

Job Mix Selection

This study involves the selection of the job mix (or mixes) to be used for the test cases. The guidelines previously presented will be followed, and both analytical and simulation methods will be employed to obtain a reasonable job mix.

The first guideline is that the job mix should resemble one found in the "real world". The "real world" in this case will be the Iowa State University Computation Center. A distribution of job run times from June 15, 1967 to June 15, 1968 was obtained and is shown in Figure 1.

From this distribution it is possible to place the jobs according to run time into four main classifications:

Short	0-15 sec	41%
Med. Short	15-60 sec	28%
Med. Long	60-300 sec	24%
Long	over 300 sec	7%

Within each of these classes there would be a distribution according to job types. In general, conversational jobs would tend to be shorter

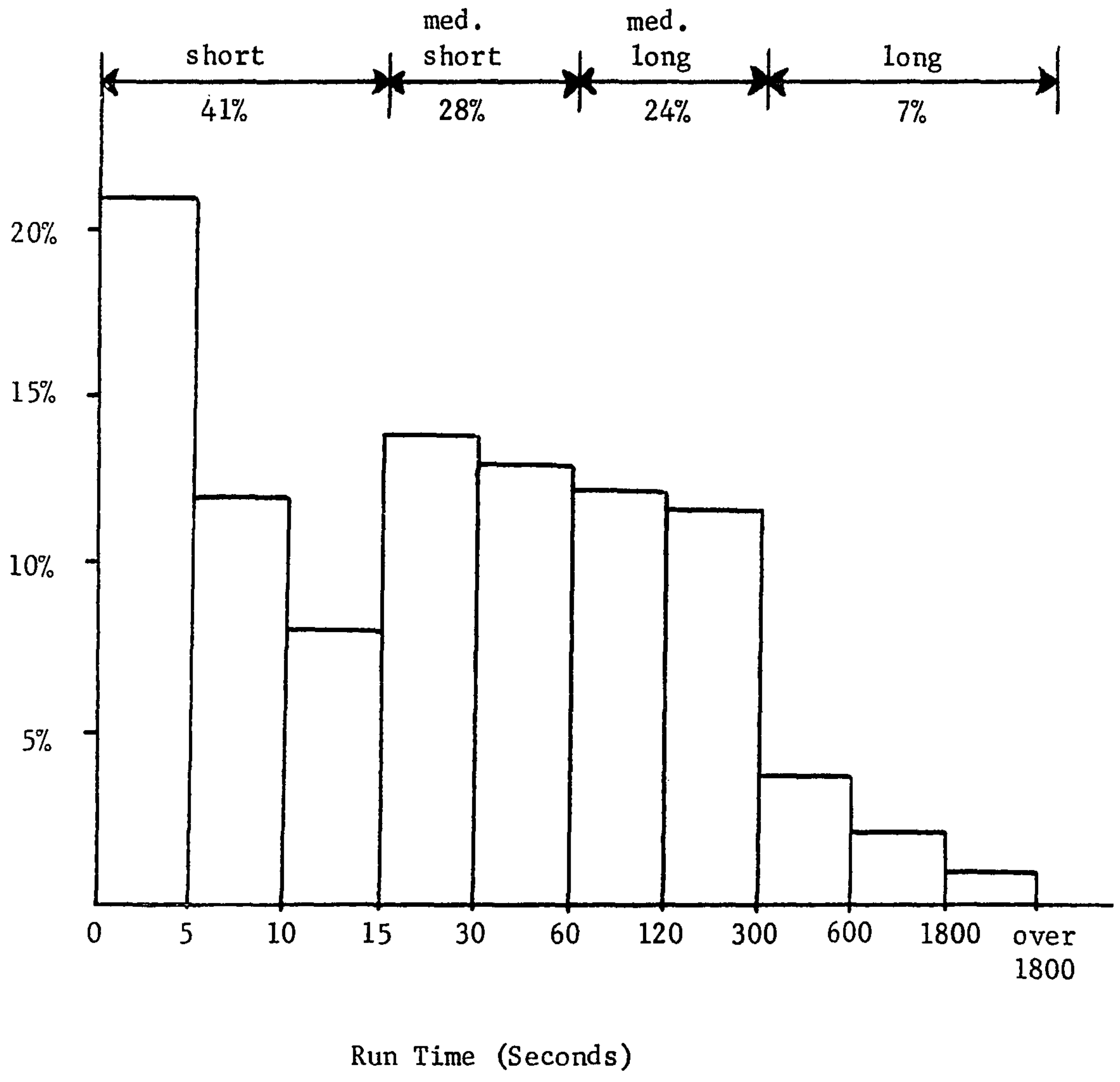


Figure 1. Distribution of job run times

than batch jobs. The disnistribution of run times for conversational jobs would tend to be skewed td toward the shorter run times. The batch jobs would tend to be skewed td toward the longer run times with respect to the total job mix.

Job list for simulation

Stanford University ty developed a list of 35 job types for the simulation along with the approropriate instruction list for each job type. This list, which required thrieree man-months to assemble, reflects the general types of jobs likely to bi be found in a time sharing system. No claim is made that a job type reprepresents the exact behavior of any real job. However, the job types do hahave the same characteristics of a real job (e.g., a list processing job wilwill do a lot of paging).

The job mix is obtaiained by selecting job types from this list. Any job type on the list may ty be used as many times as needed to form the desired distribution. InIn addition the starting time and the waiting time before a completed job rereinitializes itself must be specified for each job.

The list of job typeypes along with their priorities and run time classifications is given in Tabable 2.

Simulation Series I II was designed to aid in the selection of a job mix and to determine the he test cases which could be used to compare the performance of later seririees with that of the Model 67 as simulated in this series. The results of th this series of simulations are given in Table 3.

Test mix 1 — totally coronversational

In order to eliminatuate as many variables as possible in the selection of a stable job mix, the he first test mix is totally conversational.

Table 2. Simulation job list characteristics

Job	Description	Priority[a]	Run Time Classification
1	1000 statement conv.	2	medium short
2	1000 statement conv.	2	medium short
3	1000 statement conv.	2	medium short
4	1000 statement batch	3	medium short
5	1000 statement batch	3	medium short
6	1000 statement batch	3	medium short
7	150 statement conv.	2	short
8	150 statement conv.	2	short
9	150 statement conv.	2	short
10	150 statement batch	3	short
11	150 statement batch	3	short
12	150 statement batch	3	short
13	50 statement conv.	2	short
14	50 statement conv.	2	short
15	50 statement conv.	2	short
16	file maintenance	3	medium long
17	desk calculator	2	short
18	game	2	medium short
19	heavy cmpt. conv.	2	medium long
20	heavy cmpt. batch	3	medium long
21	tape to file	1	medium short
22	file to tape	1	medium short
23	disk to print	1	medium long
24	card to disk	1	medium long
25	list proc. med. conv.	2	medium long
26	list proc. med. batch	3	medium long
27	list proc. large conv.	2	long
28	list proc. large batch	3	long
29	prog. ckout system	2	short
30	production med. conv.	2	medium long
31	production med. batch	3	medium long
32	production large conv.	2	long
33	production large batch	3	long
34	short execute	2	medium short
35	never finish	2	long

[a]Priority level 1 - I/O jobs
Priority level 2 - Conversational jobs
Priority level 3 - Batch jobs

Table 3. Results of Simulation Series I

Run	1a	1b	1c	1d	1e	2a	2b
Initialization time	30	30	30	60	60	30	60
Run time	180	180	180	180	300	180	180
CPU data							
Execution	33.6%	33.7%	33.8%	34.8%	34.8%	40.8%	38.9%
Overhead	33.2%	32.3%	32.7%	33.1%	33.2%	33.2%	33.7%
Idle	33.2%	34.0%	33.4%	32.1%	32.0%	26.0%	27.4%
Response time							
Average	1.57	1.57	1.57	1.64	1.64	3.10	3.11
Range	1-3	1-3	1-3	1-3	1-3	1-6	1-6
Paging rates							
Read	27	26	27	28	28	30	32
Write	52	48	48	50	49	36	37
Pages retrieved	43	38	40	41	40	17	17
Drum queue data							
Read - average	2.0	2.0	2.0	2.0	2.1	2.2	2.2
Write - average	1.8	1.9	1.8	1.9	1.8	1.7	1.6
Device utilization							
Drums	15%	13-15%	12-16%	12-17%	13-16%	12-13%	12-13%
Nonpaging disks	2-38%	2-37%	2-41%	1-43%	1-42%	3-41%	2-44%

Table 3 (Continued)

Run	2c	3a	3b	3c	4
Initialization time	60	30	60	60	60
Run time	300	180	180	300	180
CPU data					
Execution	40.1%	59.5%	59.2%	59.7%	42.2%
Overhead	33.7%	31.8%	32.2%	31.8%	41.1%
Idle	26.2%	8.7%	8.6%	8.5%	16.7%
Response time					
Average	3.09	3.82	3.82	3.81	3.04
Range	1-6	3-5	3-5	3-5	2-5
Paging rates					
Read	32	22	23	22	60
Write	37	29	29	28	56
Pages retrieved	16	16	15	15	13
Drum queue data					
Read - average	2.2	1.9	1.9	1.8	3.4
Write - average	1.6	1.6	1.6	1.5	1.8
Device utilization					
Drums	13%	9-10%	9-10%	9-10%	22%
Nonpaging disks	2-43%	2-33%	1-35%	1-35%	1-45%

After a stable, conversational job mix has been determined, background jobs will be added to produce a combined mix.

Twenty five jobs are used with job types which were selected to fit the distribution previously determined. Since all jobs are conversational in nature, the distribution was skewed slightly toward the short jobs.

Simulation runs (Series I - Run 1) were used to determine the stability of the job mix with respect to order of jobs input to the simulation, starting time for each job, and length of the simulation runs.

Run 1a was made with the order of input jobs as given above, and all jobs started at the same time. Run 1b was made with the order of the jobs completely reversed, all jobs starting at the same time; and Run 1c was made with different starting times for the various jobs. A comparison of the results for these runs shows very little variation. Note that due to the random methods used to produce jobs in the simulation, the job mixes produced by re-ordering the input sequence are not identical. The job types are the same, but the individual jobs are not necessarily the same. For this reason a slight amount of variation in the results should be expected.

Increasing the initialization period (Run 1d) only produces a slight variation in the results. An initialization period of 60 seconds (Run 1d) appears to be adequate to produce consistent results.

A longer run (Run 1e) shows virtually no change in results with respect to the shorter run (Run 1d). Therefore, Run 1d was selected as the representative run for this test mix.

These simulations show that the job mix selected is stable. The results of Run 1d indicate that for this mix the CPU time is just about evenly distributed between execution time, overhead time and idle time; and the average response time is 1.6 seconds.

Test mix 2 — background jobs added

Having determined that the conversational job mix selected is stable, it was then possible to add some batch jobs to make a combined mix. This combined job mix now fits the desired "real world" distribution. The distribution is shown in Figure 2.

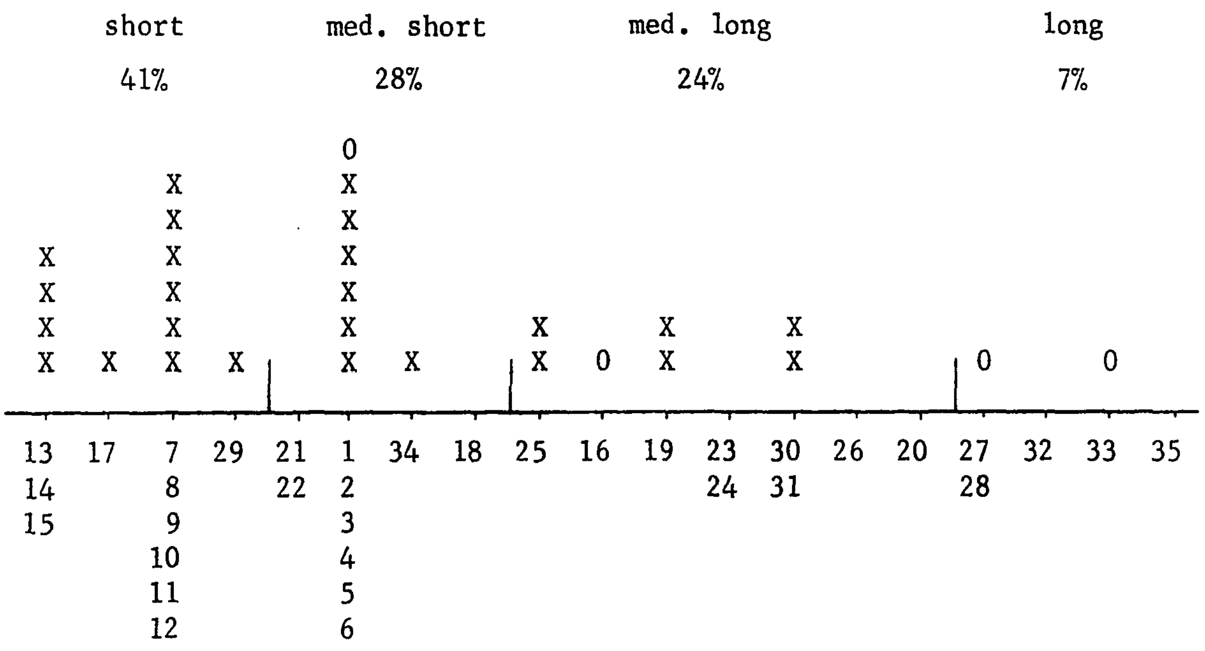
The performance results for this heavy conversational-light batch job mix were given by Simulation Series I — Run 2.

Runs 2a, 2b, and 2c were made with different values of initialization periods and run times. The only difference in the results of these three runs is a small variation in CPU utilization times. Run 2b was selected as the representative run for this job mix.

A comparison of the results of Runs 1 and 2 shows that adding some batch jobs to the conversational mix shifts about 4-5% of the total CPU time from idle time to execution time, while the overhead remains about the same. The response time increased from 1.6 seconds to 3.1 seconds. This is due to the fact that the operational cycle time (OCT) for the system is set to 3 seconds. Also note that the total paging activity has been reduced by adding the batch jobs.

Test mix 3 — heavy batch jobs

In order to obtain the results for a light conversational — heavy batch job mix, a third test mix was developed. Once again the "real-world" distribution was used to produce an appropriate mix.



The 35 job types (ordered by run time)

X = Conv. jobs
0 = Batch jobs

Figure 2. Job mix distribution for test mix 2

Twenty batch jobs and nine conversational jobs are used to make this mix. The distribution of these jobs is shown in Figure 3.

Simulation Series I — Run 3 was used to provide the performance results for the light conversational — heavy batch job mix.

Runs 3a, 3b, and 3c were made with different values of initialization periods and run times. No significant variation exists between the three runs; so Run 3b was selected as the representative run for this job mix.

The results show that execution time has increased significantly, idle time has decreased by the same amount, and overhead time is about the same as for Runs 1 and 2. Average response time is up to 3.8 seconds, and the total paging activity is further reduced.

Validity of test cases

The test mixes developed were designed to represent the types of jobs found in a time-sharing system and to fit the run time distribution of an actual system. One question still remains: How well do the test mixes reflect the paging behavior of jobs found in a time-sharing environment?

System Development Corporation (SDC) investigated the paging behavior of several programs that might be typical in a time-sharing system. The results of this investigation are presented in a paper by Fine et al. (3). Figure 4 shows the number of pages demanded by a job as a function of the job execute time. The assumption is made that the job starts with no pages and each page is brought in on demand. The demand paging rate is very high at the beginning of the time slice, but it decreases after a sufficient number of pages has been made available. During a 100 milli-second time slice (the same length used for the simulation runs) a job

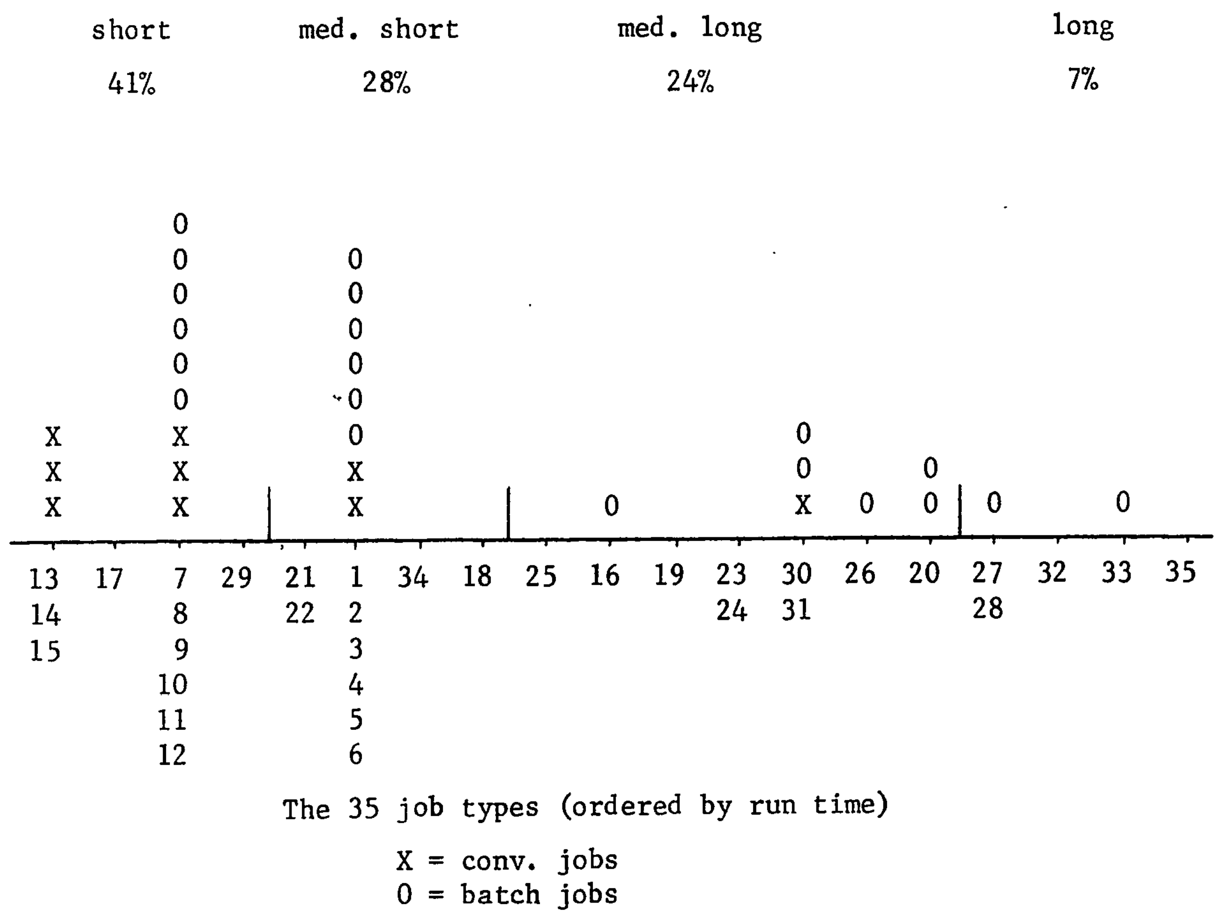


Figure 3. Job mix distribution for test mix 3

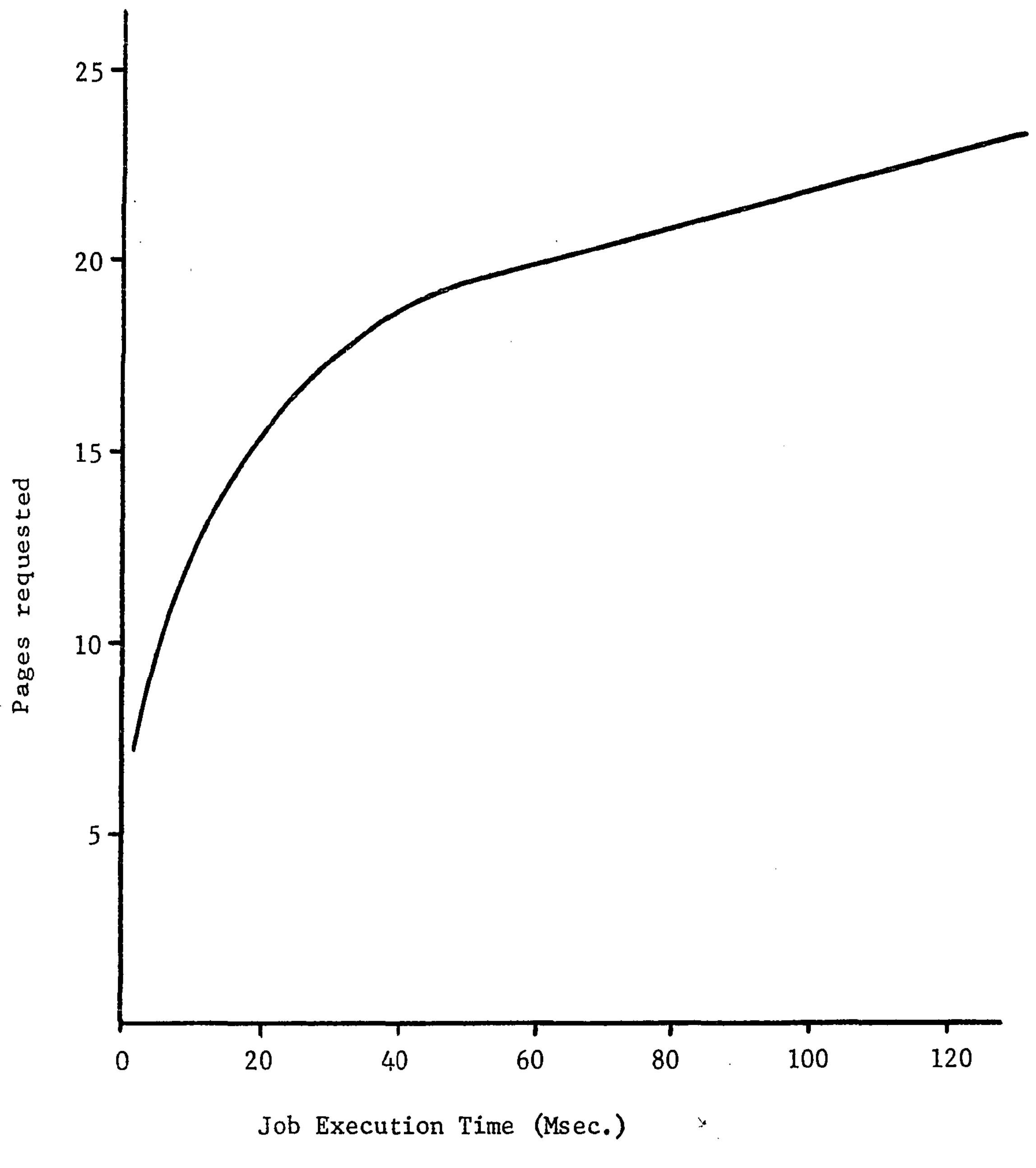


Figure 4. Paging behavior of time-sharing jobs

references approximately 22 pages. This means that during its execution time, a job requests pages at the average rate of about 220 pages per second.

The simulation program produces the total number of pages requested by the job mix for each run. The results of the simulation runs show that during their execution time, job mixes 1, 2, and 3 request pages at the average rates of 216, 154, and 90 pages per second, respectively. Thus, job mix 1 has essentially the same demand paging rate as the jobs studied by SDC. Job mixes 2 and 3 have somewhat lower rates because of their background jobs which require less paging.

In order to obtain the total performance of the time-sharing systems being studied, all three job mixes will be used. These test mixes were designed to represent extremes in the job load. If only one mix had to be selected, it would most likely be somewhere between job mixes 2 and 3. A single job mix which can be used to evaluate the total system performance is difficult to define. For this reason extremes are used.

Also note that none of the job mixes contain any bulk I/O jobs (e.g., card to disk, disk to print, etc.). The simulation runs are so short that any bulk I/O jobs tend to dominate the system activity, and the simulation results reflect this fact. Since it is the paging structure that is being investigated, the decision to eliminate the bulk I/O jobs is justifiable. As a consequence of this decision, the tape units, the card reader-punches, and the line printers are not used during the simulation runs.

Comparison of results

Since the same equipment configuration and job length distribution were used in each run, a comparison of the results obtained by varying job types can be made. Some generalizations that can be drawn from a comparison of Runs 1d, 2b, and 3b are given below.

CPU utilization   As the job types progress from totally conversational to totally batch, the system idle time is exchanged for execution time with the overhead time remaining fairly constant. This is as expected since the purpose of having background jobs is to use the idle time.

Response time   The price paid for reducing the idle time by adding background jobs is an increase in response time. The response time is lowest with all conversation jobs. Addition of a few background jobs causes the response time to rise sharply and then level off, rising only slightly, as more background jobs are added. The value at which this limiting action takes place is set by the operational cycle time (OCT) of the system. For this configuration the OCT is equal to 3 seconds. Increasing the OCT would give more CPU cycles to the background jobs, while decreasing it would give less.

Paging rate   As the percentage of batch jobs increases, the paging rate decreases. Batch jobs tend to have longer run times and execute longer without interruption than do conversational jobs. Therefore, the paging rate (pages per second) is generally less for a batch job.

Special test mix — heavy paging

The performance results with job mix 1 show that the system is idle a good deal of the time. This, together with the fact that a large number of pages are retrieved from the core allocation queues, indicates that the system can handle a heavier job load. An additional simulation run (Run 4) was made with job mix 1 doubled in size to provide the system with 50 jobs having the same characteristics as mix 1.

Comparing the results of Run 4 with those of Run 1d shows that much of the idle time has been reduced, the response time has doubled, and the paging rate has been increased. This double job mix will be used as a test mix when runs with heavy paging requirements are desired.

PAGING STRUCTURE OF A TIME-SHARING COMPUTER SYSTEM

The results in the previous chapter indicate that the CPU is either idle or is performing overhead functions much of the time. Neither of these is directly beneficial to the progress of a particular user's problem. This study will investigate some of the causes of these high idle and overhead times, and methods to improve the performance of the system will be proposed.

The paging structure is primarily responsible for the poor system performance. To demonstrate this, it will be necessary to examine the paging structure philosophy used in many current time-sharing computer systems of which the IBM System/360 Model 67 is a representative example.

Demand Paging

A fixed-sized page is brought into main memory only after some location in that page has been referenced. This concept is known as demand paging. After a task has referenced a new page, that task is placed in a page wait state, and none of its pages in main memory can be used until the referenced page is brought in. During this wait, control is given to another user, assuming that there is another user. In this way page fetching is overlapped with processing. It is possible, and indeed quite probable, that all tasks occupying main memory are in a page wait state. This condition results in CPU idle time. In addition heavy paging demands can also cause congestion at the paging devices which may result in poor user response.

In an attempt to minimize some of the problems just described, alternatives to demand paging have been suggested. The concept of affinity paging involves bringing into main memory groups of pages which have an affinity for each other. Whenever a single page is referenced, the monitor is to recognize that certain other pages are likely to be required and brings in these pages along with the referenced page. Of course, this concept assumes the ability of the monitor to recognize page affinities. To provide this ability, the user would most likely have to supply these page relationships or else organize programs so that they might be easily segmented. This approach of fitting the jobs to the system instead of vice versa does not appear to be realistic and for this has not been universally accepted. Because of a lack of a suitable alternative, current time-sharing philosophy seems to be firmly committed to the demand paging concept.

Paging devices must be able to deliver pages at the rate required by the system. Three types of paging devices currently used in time-sharing computer systems are described below.

Paging disks

The configuration of the Model 67 allows disks to be used in conjunction with a drum as the paging devices. The slower disks are used primarily as back-up devices for the drum. In his original simulations Nielson (13) demonstrated that the disks seriously degraded the system performance because of their inability to deliver pages at the required rate and suggested that the paging disks be replaced by an additional drum. For this reason no paging disks are included in the equipment configuration used to develop the test cases.

Paging drums

Drums are the most common paging devices used in current time-sharing systems. Some investigators have suggested that paging drums are not able to deliver pages at the required rate and should not be used when heavy paging demands are involved. A detailed discussion on paging drums is deferred until a later section.

Large capacity core storage (LCS)

LCS provides the most rapid (and most expensive) means of delivering pages to requesting tasks. There are two ways in which LCS can be used. The first is to use LCS in a manner analogous to the use of paging drums. Whenever a page is requested by some task, that page is located in LCS and brought into main memory. The second way is to execute directly from the slower speed LCS, thereby eliminating a page transfer. This second method can be used very effectively if the number of references to a page is small or if time-sharing is only a small part of the total system function. It has been suggested by Lauer (10) that a combination of these two methods can be used. That is, when a large number of references is to be made to a page, that page should be transferred from LCS to main storage; if the number of page references is small, LCS can be accessed directly. Unfortunately, there is presently no way for the system to determine before the fact how many references will be made to a page, and because of this, the combination method does not seem to be a practical solution.

Drum Paging Structure

Lauer also analyzed the drum paging structure used for a time-sharing system. The drum that he considered has one read-write head per track, but only one head can be connected to the channel at any one time. There are p pages per track with sufficient space between pages to permit head switching. Thus, on one drum revolution p pages may be read and/or written. This is equivalent to one head passing p slots per revolution. The maximum rate at which pages can be transferred to or from a drum is $\frac{p}{T}$ pages per second where T is the time required for one drum revolution. Page writes are necessary for those pages which have been changed during a task's time slice and now must be paged out to make room for a subsequent task's pages. If f is the probability that a page must be written out, then $\frac{p}{(1 + f)\ T}$ pages per second is the maximum average rate at which pages can be read from the drum.

Only one page can be read from a slot per revolution. A slot conflict occurs when more than one page is requested from the same slot at the same time. Under this condition only one request can be serviced on the current revolution; the rest will have to be delayed until they can be serviced on some future revolutions of the drum. Because of slot conflicts, the maximum average rate at which pages can be read from a drum can only be approached. Page writes do not generate slot conflicts. When the channel program is set up to control the drum operation during the next revolution, the monitor first schedules as many read requests as possible and then schedules page writes for the unused slots.

Lauer states that the average rate at which pages can be read is given by the results of the probability exercise known as the Urn-model Occupancy problem. Figure 5 shows the average page read rate as a function of f and k, the number of requests in the page read queue. The case p = 9, T = 34.4 milliseconds is the one used for the test cases. For this

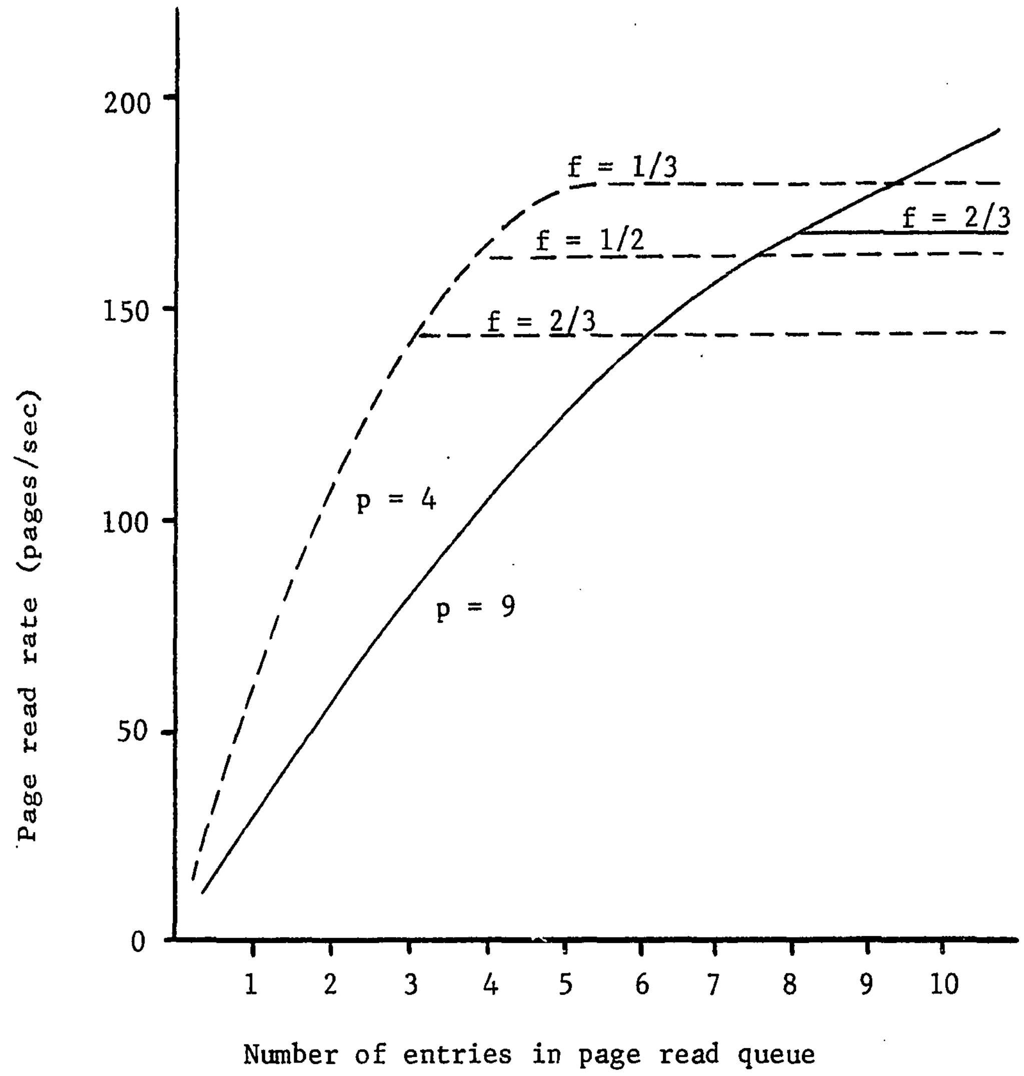


Figure 5. Average page read rate

configuration $4\frac{1}{2}$ pages are placed on each track so that 9 pages can be referenced during a double drum revolution. The case $p = 4$, $T = 17.2$ milliseconds is the same drum reorganized so that 4 pages can be referenced during each drum revolution.

From the figure it appears that a high page read rate is only possible with a long read queue. Each entry in the read queue represents one task which cannot continue executing because it has been placed in a page-wait state. Lauer has concluded that the paging device must deliver pages at the rate requested by the job mix in order to minimize the CPU idle time. Therefore, if drums are used, long read queues must result and a large amount of main storage must be available to support the large number of tasks in a page-wait state. He has further concluded that drums are not well suited as paging devices and should be replaced by LCS.

It was previously shown that job mix 1 requests pages at an average rate of about 216 pages per second. If the drums must deliver pages at this rate, the conclusions drawn by Lauer may be correct. However, it can be demonstrated that the actual rate at which the drums must deliver pages is considerably less.

There are two major reasons why the rate at which the drums must deliver pages is less than the rate requested by the job mix. First, and most important, heavy page requests are only generated during a task's execution time. No pages are requested during system idle time, and only a few, if any, are likely to be requested by the system during overhead time. Therefore, the average rate at which pages are requested is 216 pages per second only if the system execution time is 100% of the CPU time, a situation which is highly unlikely to even be approached.

The second reason is that not all page requests produce page reads. A page request can be satisfied in one of four ways. In addition to being read from a paging device, the requested page may have been read during a previous time slice and is still available to be retrieved; it may be read from an I/O device; or finally it may be a shared page which is in main memory being used by some other task.

As an example, consider Simulation Series I — Run 4. The results of this simulation show that the system execution time only accounts for 42% of the total CPU time. This means that the tasks are generating page requests only 42% of the time or at an average rate of only 90 pages per second. Of the 90 pages per second requested, an average of 60 pages per second are read, 13 pages per second are retrieved, and the remaining 17 pages per second are either shared pages or are obtained from I/O devices. While it is recognized that the requested rate may at times exceed the rate at which the drums can deliver pages, the important point is that the drums can keep up with the requested rate by delivering an average of only 60 pages per second and not the 216 pages per second as originally assumed. For the average read queue length of 3.4 requests, Figure 5 indicates that each drum can easily deliver 60 pages per second.

The size of the read queue also indicates the number of tasks currently in a page-wait state. For the example cited the average number of tasks in main memory at any one time is approximately six. The average size of the read queue indicates that about half of these tasks are in a page-wait state; the rest are ready to execute. The fact that the drums only deliver pages at an average rate of 60 pages per second has not resulted in excessive system idle time caused by many tasks waiting for page reads.

There is an explanation for the discrepancies between the results given in Lauer's paper and the results of this study. Lauer has assumed that the worst case situation is the normal mode of operation in a time-sharing system. The results presented here indicate that his assumption is not valid.

IMPROVEMENT OF SYSTEM PERFORMANCE

An effective way to improve the performance of the time-sharing computer system is to increase the amount of useful work performed by the system. To accomplish this, it is necessary to increase the fraction of time devoted to CPU execution by reducing the unproductive system time. Because service to the user is of paramount importance in any time-sharing system, care must be taken to insure that a reasonable response time is provided. In the following sections, modifications to the paging structure of the drum-oriented system will be proposed in an attempt to achieve the goal of improved hardware efficiency. The detailed evaluation of all of the proposed changes to the paging structure is postponed until the next chapter.

There are two main causes of CPU idle time in a time-sharing system. One is a lack of work for the system to perform, and the other is the inability to deliver pages to waiting tasks. Idle time caused by a lack of work can be reduced by providing background jobs which execute when no other work is available and which relinquish control when a higher priority job (e.g., conversational job) is ready. The results of Simulation Series I have shown how the addition of background jobs has significantly reduced system idle time.

The second cause is not so easy to resolve. CPU idle time results when all tasks in main memory are in a waiting condition. If the rate at which the drums can deliver pages could be increased, the average number of tasks in a page-wait state, along with system idle time, should be reduced. Figure 5 indicates that if the drums are reorganized with 4 pages per track rather than the original 4½ pages per track, the rate at which pages can be delivered is greatly increased. The cost of reorganizing the drums is an 11% reduction in drum capacity.

Simulation Series II — Runs 1a, 2a, 3a, and 4a were used to simulate the original system with the reorganized drums. The results are given in Table 4. A comparison of these results with those of the original system (Series I — Runs 1d, 2b, 3b, and 4) shows that, as predicted, the system idle time and the size of the drum read queues have been reduced. It is interesting to note that the percentage of CPU time used for execution is practically unchanged. Overhead time has increased because of the additional number of interrupts necessary to schedule the drums for every single revolution rather than for every double revolution as was originally the case. Even though the reorganized drums are capable of delivering pages at nearly twice their original rate, the average number of pages read per second is essentially the same. Thus, it appears that this reduction in system idle time has not significantly improved the performance of the system. It should also be quite clear that even if all of the CPU idle time could be eliminated, a considerable amount of time is still devoted to overhead. While it is important to be able to reduce the system idle time, something must be done about overhead time before any major improvement in performance will be achieved.

Overhead times cannot be reduced in the same manner as were system idle times. Overhead is the amount of CPU time needed by the monitor to perform such functions as interrupt handling and I/O scheduling and is, therefore, inherent in any system. The paging structure in a time-sharing system is chiefly responsible for the high overhead times. The remainder

Table 4. Results of Simulation Series II

Run	1a	1b	1c	2a	2b	2c
Initialization time	60	60	60	60	60	60
Run time	180	180	180	180	180	180
CPU data						
Execution	37.4%	39.3%	41.2%	39.1%	46.1%	48.0%
Overhead	43.2%	35.9%	31.5%	42.4%	34.7%	29.5%
Idle	19.4%	24.8%	27.3%	18.5%	19.2%	22.4%
Response time						
Average	1.55	1.43	1.39	2.92	2.85	2.91
Range	1-3	1-3	1-3	1-5	1-5	1-5
Paging rates						
Read	31	29	27	33	34	33
Write	56	57	61	39	40	39
Pages retrieved	47	52	58	20	21	20
Drum queue data						
Read - average	1.7	1.7	1.5	1.8	1.7	1.8
Write - average	1.9	2.0	2.1	1.8	1.9	1.8
Device utilization						
Drums	18-20%	17-21%	18-20%	15-17%	16-17%	14-18%
Nonpaging disks	1-44%	1-48%	1-49%	2-41%	2-38%	2-42%

Table 4 (Continued)

Run	3a	3b	3c	4a	4b	4c
Initialization time	60	60	60	60	60	60
Run time	180	180	180	180	180	180
CPU data						
Execution	56.5%	65.0%	68.1%	42.2%	43.9%	45.1%
Overhead	40.5%	31.3%	26.9%	49.4%	43.9%	39.4%
Idle	3.0%	3.7%	5.0%	8.4%	12.2%	15.5%
Response time						
Average	3.54	3.55	3.56	3.03	2.88	2.75
Range	3-5	3-5	3-5	2-5	2-5	2-5
Paging rates						
Read	23	23	24	59	61	64
Write	31	32	31	56	60	62
Pages retrieved	23	23	21	17	21	19
Drum queue data						
Read - average	1.5	1.4	1.6	2.2	2.4	2.4
Write - average	1.6	1.9	2.0	2.1	2.7	2.3
Device utilization						
Drums	11-13%	11-13%	11-13%	25-26%	26-27%	27-28%
Nonpaging disks	1-38%	1-29%	1-32%	1-45%	1-47%	1-50%

of this chapter will examine some of the causes of high overhead and will evaluate changes to the paging structure which are proposed to reduce overhead times.

Scheduling of the Paging Drums

Each drum operates independently under the control of its own channel program. A new channel program must be set up for each drum·revolution. To accomplish this, the CPU is interrupted before the start of the next revolution, and pages are scheduled to be read or written during the next revolution. Another interrupt at the start of the revolution gives the channel program control of the drum. This operation is performed for every single revolution, whether or not there are pages transferred to or from the drum. Each drum revolves approximately 60 times a second, causing about 120 channel programs to be set up per second for the two drums. This amounts to a great deal of unnecessary overhead time if pages are not transferred during every revolution.

Previous results have demonstrated that the drums are not busy at all times. This is especially true for the job mixes which have very low paging requirements. For this reason the decision was made to eliminate some of these unnecessary interrupts in order to reduce the overhead time. The procedure used is particularly simple. When no pages are being read or written on the current revolution, the next regularly scheduled drum interrupt is skipped. If the drums are used during every revolution, then this procedure has no effect. On the other hand, when very little paging activity is involved, up to half of the originally scheduled interrupts are eliminated.

The fact that no pages are being transferred on the current revolution indicates that no pages are currently pending in a read or write queue. Any request entering one of these queues after the decision has been made to skip the next revolution will be delayed longer than usual. This may result in a slight increase in the size of the read or write queues. The overall system performance, however, should be increased due to the decrease in overhead time.

Simulation Series II — Runs 1b, 2b, 3b, and 4b were used to simulate the system with the reorganized drums and the drum scheduling procedure just described. The results of this set of runs indicate a definite improvement in performance. This can be seen by comparing the simulation results with those of Runs 1a, 2a, 3a, and 4a. Overhead has been substantially reduced causing an improvement in both execution and reponse times. The drum scheduling procedure appears to be a very beneficial addition to the system.

Multiple Sets of Registers

The drum scheduling procedure has reduced the overhead by eliminating some of the interrupts required by the system. This procedure is most effective when the drums are not being used to their fullest extent but has little or no effect when high paging rates are involved. Certainly it is equally important, if not more so, to improve the performance of the system with high paging rates. The previous discussion and results indicate that there are gains to be made by using a more efficient interrupt structure.

An efficient interrupt structure is an especially important asset for a real-time computer system. The Scientific Data Systems (SDS) SIGMA 7 time-sharing computer system was designed primarily to solve the problem of achieving true real-time response. A set of 16 general registers is provided in the system. As an option, multiple sets of these registers can be included. In this way, when an interrupt occurs, it is not necessary to store the contents of the set of registers in order to preserve the condition of the system before the interrupt. Instead, a new set of registers can be used to service the interrupt. The need to restore the registers at the end of the interrupt to their condition prior to the interrupt has also been eliminated. Hence, this technique saves the time needed to store and load the set of general registers for every interrupt. The only time required is about 6 microseconds to switch control from one set of registers to another. A detailed description of the design features for the SIGMA 7 is presented in a paper by Mendelson and England (12).

This idea of adding sets of general registers to increase the efficiency of the interrupt structure should greatly improve the system performance. A set of simulations was designed to determine how much of a gain could be achieved with the addition of five sets of register (one set for each of the five classes of interrupts in the System/360). The results of Simulation Series II — Runs 1c, 2c, 3c, and 4c show that in all cases over 4% of the total CPU time has been removed from the overhead time and has been distributed between execution and idle times.

The results in this and the previous section indicate that much can be gained through modification of the interrupt structure. A detailed evaluation of these performance improvements will be made after all other changes to the paging structure have been presented.

Hardware Paging Structure

Dynamic relocation is achieved in the Model 67 by providing each task with its own set of relocation tables. These tables provide a map between the logical addresses of the task and the physical memory addresses. Whenever a task is given a time-slice, its relocation tables must be in main memory. Address translation is accomplished by using the task's logical addresses to access its relocation tables which contain the physical addresses of the pages belonging to the task.

It is obvious that the relocation tables of a task must at all times contain the current location of the task's pages. Whenever a page is read into or written out of main memory, it is necessary to modify the relocation tables to reflect this change. For this software structure, a considerable amount of overhead is required for each page read or written.

It is interesting to consider the amount of overhead that would be required for the Model 67 to adjust the relocation tables for the extremely high paging rates that were discussed in the previous chapter. Assume for the moment that it is necessary to read pages at the rate of 220 pages per second. Also assume that the rate of pages written out is approximately two-thirds of the read rate, or about 140 pages per second. This last assumption is in line with the results of the simulation runs previously presented and also agrees with the findings of Fine _et al_. (3). A study carried out by the Stanford Computation Center and reported in the paper by Nielson (13) showed that the Model 67 requires approximately 785 microseconds to adjust the tables for a page just read in and approximately 1023 microseconds for a page just written out. Using these values and the paging rates assumed, the overhead involved only to adjust tables is

315,920 microseconds per second, or in other words, 31% of the total CPU time is spent in adjusting tables. These figures do not begin to reflect any of the time required to release pages at the end of a time slice, to select the core page to be replaced when a page is read in, or to perform any other system overhead functions. This example is simply another indication that even if the paging device could deliver pages at a very high rate, the system would not be able to make use of this capability because of the fantastically high system overhead involved.

Since paging is basic to the operation of a time-sharing system, it would seem beneficial to implement the paging structure with hardware in order to provide greater efficiency. The software structure just described is used primarily because it can be developed from a conventional processor with very few modifications. The Model 67 CPU is basically a Model 65 processor with only a few changes. While this may be the easiest structure to implement, it may not be in the best interests of time-sharing. Consequently, some of the advantages that can be gained by using a hardware paging structure will be explored.

Instead of relocation tables the hardware paging structure uses one word for each page in main core. These words contain the logical addresses currently associated with the memory pages and the information bits which are used to indicate the current status of a memory page. Address translation is accomplished by interrogating these words to determine if the desired page is in main core, and if it is, where it is located.

Information bits

The information bits associated with each page of main core are used primarily for page replacement. The information bits for the proposed structure are described below.

Use bit This bit is set for a page when that page is accessed. All use bits are reset only after each use bit has been set.

Change bit This bit is set whenever a page is written into and reset only after the page has been written out to secondary storage.

Activity bit This bit is set whenever the change bit is set, but it is reset at the time slice end of the current task. These bits are used at the end of a task's time slice to determine which pages are to be written out. Note that the change bit remains set until the page is actually written out.

Transmit/lock bit This bit is set whenever a page is in the process of being read into or written out of main core. Under no circumstances can a page be referenced or replaced when this bit is set. After a channel program has been set up, there is no way to change it. Hence, this bit is necessary to insure that the pages being read or written are not tampered with. This bit may also be used as a lock bit to prevent the referencing of some page which is unusable (e.g., a memory page in which some fault has been detected).

Protection bits These bits provide the read/write protection for each page. They may be stored with the other information bits for each page or with each page in core as in the Model 67.

This structure requires four information bits plus the protection bits for each page. The ability to locate a core page which has associated with it any combination of information bits is necessary. The status of any core page at any time can be determined from the configuration of the four information bits. Table 5 lists all possible bit configurations along with the appropriate core page status.

Translation algorithm

The translation algorithm for the proposed paging structure is given in Figure 6. This algorithm is only concerned with the translation of the logical page address into the actual page address. The byte portion of the logical address and the actual address are the same.

Page read algorithm

When a desired page is not in main core, it is necessary to locate that page in secondary storage and read it into some predetermined location in main core. The location in main core into which the page is to be read is selected by the replacement algorithm.

For the hardware structure proposed, the replacement algorithm is very straight-forward. The information bits for each core page are interrogated to find a page with the bit configuration:

use	change	activity	transmit/lock
0	0	0	0

All unused pages or pages which have not been used recently will meet this condition. Hardware must also be provided to select the first core page which meets this condition in the event of a multiple match.

If this condition cannot be satisfied, then no page replacement can take place. Note that the change bit is set during the time a page is on the write queue and reset when it is written out. If the condition for

Table 5. Information bit configurations

Information bits				
use	change	activity	transmit/lock	Status of associated core page[a]
0	0	0	0	not recently used (NRU)
0	0	0	1	NRU, being read in
0	0	1	0	impossible combination
0	0	1	1	impossible combination
0	1	0	0	NRU, on write-out queue
0	1	0	1	NRU, being written-out
0	1	1	0	NRU, changed during current time slice
0	1	1	1	impossible combination
1	0	0	0	recently used (RU)
1	0	0	1	RU, being read in
1	0	1	0	impossible combination
1	0	1	1	impossible combination
1	1	0	0	RU, on write-out queue
1	1	0	1	RU, being written out
1	1	1	0	RU, changed during current time slice
1	1	1	1	impossible combination

[a]Any combination with transmit/lock = 1 may exist when the page is locked

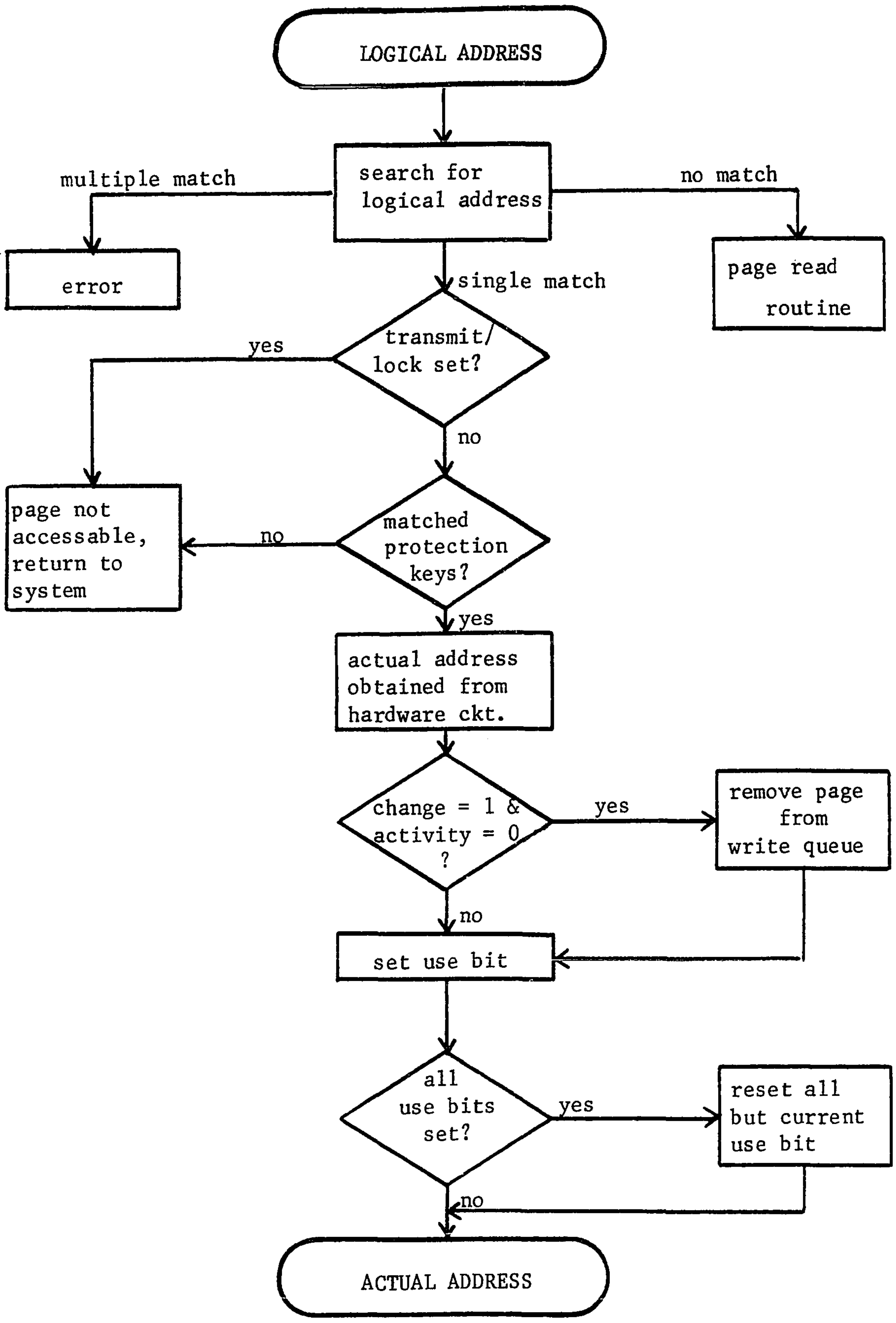

LOGICAL ADDRESS
search for logical address
multiple match
no match
error
page read routine
single match
transmit/ lock set?
yes
no
page not accessable, return to system
matched protection keys?
no
yes
actual address obtained from hardware ckt.
change = 1 & activity = 0 ?
yes
remove page from write queue
no
set use bit
all use bits set?
yes
reset all but current use bit
no
ACTUAL ADDRESS

Figure 6. Translation algorithm

replacement could not be met, it would most likely be due to an excessively long write queue. In time, the size of this queue would be reduced, and the condition for replacement would be met. When it is not possible to read in a needed page for some task, an end of time slice is called for that task and the next task is started. The algorithm for the page read operation is given in Figure 7.

Page write algorithm

To insure that an updated copy of every page exists in secondary storage, all pages which have been changed during a task's time slice are written out at the end of the time slice. The activity bits are used to indicate which pages were changed during a time slice. The page write algorithm is shown in Figure 8.

If a page which has been accessed is found to be pending on a write queue, it is necessary to remove that page from the queue. This situation can be recognized by observing that any page on the write queue meets the condition, change bit = 1 and activity bit = 0. Note that the activity bit of a page removed from the write queue must be set to insure that the page will be written out at the end of the current time slice.

Hardware implementation

In order to evaluate the performance of the hardware paging structure, some actual implementation will have to be selected. The proposed paging structure requires that the words associated with each core page be addressed on the basis of content. Associative registers are ideally suited for this type of structure.

The model 67 uses eight associative registers to provide rapid address

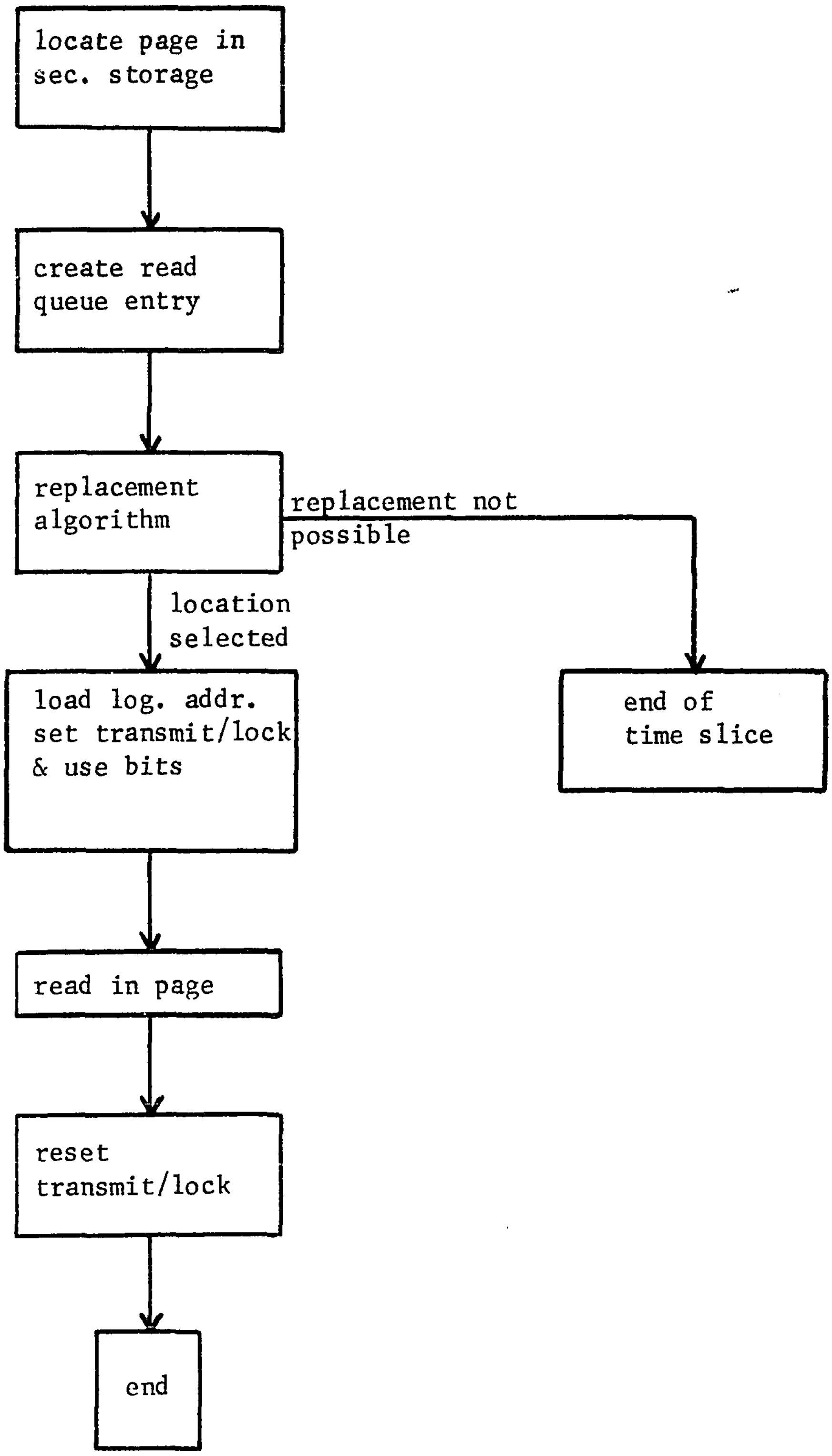


Figure 7. Page read algorithm

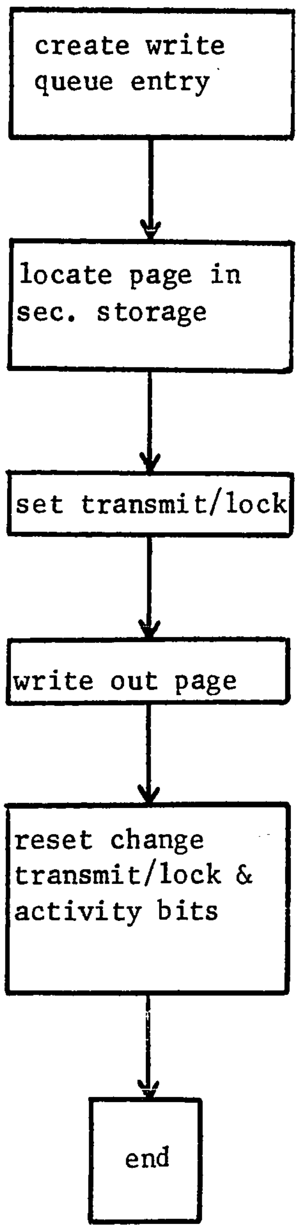

create write queue entry
locate page in sec. storage
set transmit/lock
write out page
reset change transmit/lock & activity bits
end

Figure 8. Page write algorithm

translation. Numerous studies of dynamic relocation have also used associative registers with good results. One such study by Lindquist _et_ _al_. (11) investigated the operation of an experimental 360/40 time-sharing system which used an associative memory for dynamic storage allocation. The associative register configuration used to implement the proposed paging structure will be similar to that used by Lindquist.

The associative register structure is by no means the only implementation that could have been selected. The major requirement for any memory chosen is that it is possible to interrogate its contents. If some search memory is used which cannot be interrogated as rapidly as an associative array, it may be necessary to include a few current address registers. These registers would be used to provide the actual addresses of the most recently accessed pages in much the same way as the associative registers in the Model 67. In this way, a much slower search memory could be used quite effectively. The object of this study is not to propose any particular hardware implementation but rather to evaluate the performance of the time-sharing system with a hardware paging structure.

The translation hardware is shown in Figure 9. Whenever a job addresses a memory location, the desired address is placed in the logical address register. The translation hardware translates this logical address into the actual address (i.e., the physical location in core). Since the low order bits (i.e., byte address) of the logical and actual addresses are the same, only the page portion of the logical address must be translated.

The page portion of the logical address register is simultaneously compared with the contents of each associative register to determine which

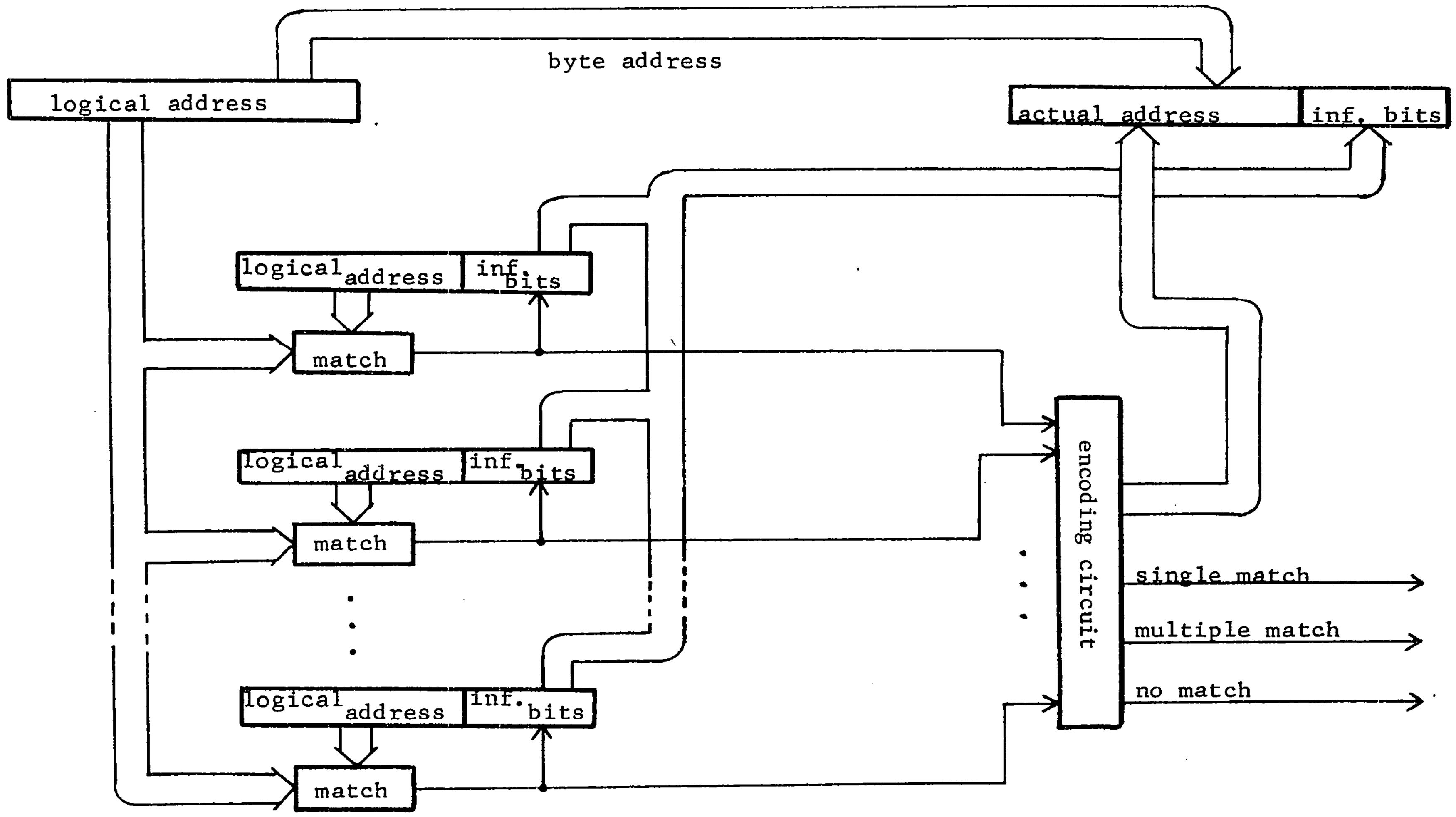


Figure 9. Associative register translation hardware

register contains the desired logical address. Because there is one register for each page in core, encoding hardware is used to form the page portion of the actual address once the appropriate register has been identified. If no register contains the desired logical address (i.e., no match), then the desired page is not in core, and a page read request must be generated to read in the desired page from secondary storage. If more than one register contains the desired logical address (i.e., multiple match), then an error condition exists which must be handled by the monitor. In addition to the translation hardware shown, this structure has the ability to interrogate the information bits and to change all or part of any register.

The translation hardware requires about 10 circuit logic levels to translate a logical address into an actual address. Because of the symmetry involved and the large potential market, the translation hardware seems well suited to be implemented with Large Scale Integration (LSI).

Simulation Series III was designed to evaluate the performance of the hardware paging structure. This series is capable of simulating the associative register implementation, and it reflects the reduced overhead parameters determined in a separate study for this configuration. The equipment configuration and job mixes that were originally developed as test cases are used in this simulation series. In order to evaluate the hardware paging structure on its own merits, the original drum structure has been restored. That being the case, the behavior of this system can be compared directly with the behavior of the original system, the Model 67. The three test mixes were used to make Runs 1, 2, and 3, respectively. The results are given in Table 6.

Table 6. Results of Simulation Series III

Run	1a	1b	1c	2a	2b
Initialization time	60	60	60	60	60
Run time	180	180	180	180	180
CPU data					
Execution	38.8%	37.2%	36.2%	43.9%	42.4%
Overhead	26.1%	26.0%	26.3%	26.7%	27.5%
Idle	35.1%	36.8%	37.5%	29.3%	30.1%
Response time					
Average	1.48	1.52	1.58	3.03	3.02
Range	1-3	1-3	1-3	1-5	1-5
Paging rates					
Read	25	25	28	28	31
Write	57	51	51	38	38
Pages retrieved	55	45	43	22	19
Drum queue data					
Read - average	1.9	2.0	2.1	2.0	2.1
Write - average	2.0	1.9	1.7	1.7	1.8
Device utilization					
Drums	13-17%	13-15%	15%	12-13%	13%
Nonpaging disks	1-48%	1-46%	1-44%	2-47%	2-45%

Table 6 (Continued)

Run	3a	3b	3c
Initialization time	60	60	60
Run time	180	180	180
CPU data			
Execution	60.8%	62.6%	63.8%
Overhead	26.8%	27.5%	27.9%
Idle	12.4%	9.9%	8.3%
Response time			
Average	3.87	3.83	3.75
Range	3-5	3-5	3-5
Paging rates			
Read	24	26	25
Write	28	31	35
Pages retrieved	13	14	19
Drum queue data			
Read - average	1.9	1.9	2.1
Write - average	1.5	1.8	1.7
Device utilization			
Drums	9-10%	10%	10-11%
Nonpaging disks	1-36%	1-35%	1-35%

Runs 1a, 2a, and 3a simulate the operation of the system with hardware paging as it has been proposed. The results show that the overhead remains constant at about 26% while idle time is exchanged for execution time as more batch jobs are added. A comparison of these results with those for the original structure shows that the hardware paging structure has removed 5 to 7% of the total CPU time from overhead and distributed it between execution and idle times. The response times for the hardware structure have also been reduced slightly. The reduced overhead reinforces the idea that the hardware structure is more efficient.

A further examination of the results shows that although the system with the hardware structure performs more useful work than the original system for the same period of time and same job mix, the number of pages read per second has been reduced for runs 1a and 2a. The number of pages retrieved per second, however, has been increased. This indicates that the hardware structure allows a greater number of pages to be retrieved while they are still in core, thus eliminating unnecessary page reads.

In the Model 67 the pages released by a task at the end of a time slice are placed at the end of one of three core allocation queues. The first queue contains those pages which were unchanged during the time slice; the second queue contains those pages which were changed and have been written out; and the third queue contains the priority pages. The priority pages consist of the page containing the last instruction executed before the end of the time slice and the pages containing the task's relocation tables. When space in core is required for a page being read in or created by a task, some page in core must be replaced. The assignment of the page to be replaced is made from the top of the first queue until it is exhausted, then from the second queue, and finally from the third queue when the second is exhausted. This insures that the pages which a task must have in core in order to begin execution in its next time slice have the least likelihood of being replaced. Whenever a page read is requested by a task, the core allocation queues are checked to see if the requested page is in core. If it is, the page is retrieved from the queue, and an unnecessary page read is avoided.

For the hardware paging structure no allocation queues are needed. Page replacement is made on the basis of the use bits associated with each core page. This configuration is analogous to a one queue structure since the page selected to be replaced is the first one (lowest order) whose use bit is not set. In this way the most recently used pages are the last to be replaced.

In order to investigate the effects of multiple core allocation queues on the hardware paging structure, a second set of runs was developed which used two queues. Hardware implementation of the two queue arrangement can be accomplished by adding a priority bit for each page in core. This bit is set for the page containing the last instruction executed at the end of a time slice and reset at the start of the task's next time slice. The replacement algorithm first attempts to locate a page with the bit configuration:

use	change	activity	transmit/lock	priority
0	0	0	0	0

If no core page can be found to meet this first condition, then an attempt is made to select a page with the bit configuration:

use	change	activity	transmit/lock	priority
0	0	0	0	1

Thus, the priority pages have the least likelihood of being replaced.

The results of the simulations of this two queue structure are given by Runs 1b, 2b, and 3b. These results indicate that the execution times for job mixes 1 and 2 have been decreased; while for job mix 3, it has increased.

To further check this trend, Runs 1c and 3c were made to simulate the three queue structure used in the Model 67. As expected, the execution time decreases for Run 1c and increases for Run 3c. Another indication of this trend is that as the number of queues increases, the number of pages retrieved per second decreases for Runs 1 and 2 and increases for Run 3.

In order to explain the variation in performance caused by changing the number of queues, it is necessary to look at the average number of pages available for replacement (the length of the allocation queues). At the end of a task's time slice, all pages in core associated with that task are made available to be replaced. In addition, all pages in core associated with a conversational task are made available for replacement at the start of a terminal wait for that task. Therefore, the average number of pages which are available for replacement at any time is likely to be greater for a heavy conversational job mix than for a heavy batch job mix. This series of simulations indicates that a one queue structure provides better system performance than a multiple queue structure does when a large number of pages is available for replacement, and vice versa.

These results also give some indication of the amount of work being performed by the system. The number of core pages not being used, is, on the average, higher for job mixes 1 and 2 than it is for job mix 3. This indicates that the system is capable of handling more work than is provided by job mixes 1 and 2. This fact was previously verified by Simulation Series I — Run 4 when it was shown that the system was capable of handling a job load double the size of job mix 1.

Queue Structures

The results of the study on hardware paging indicated that the structure of the core allocation queues had an influence on the performance of the system. This section reports the results of an investigation of the effects of queue organizations and queue lengths on system performance.

The hardware structure improved the performance of the system by reducing the overhead required for each page read or written. Another way to reduce the overhead is to reduce the number of pages which must be read or written. The only way to eliminate a page read is to insure that the requested page is in main core. This situation only occurs when a page has been previously read into main core and has not been replaced before it is again addressed. Similarly, a page write can be eliminated only if the page is addressed by some task before it is written out. By selecting an appropriate queue organization and by lengthening the allocation and write queues, it may be possible to retrieve more pages from these queues. In this way unnecessary page reads and page writes would be eliminated, and the performance of the system may be improved.

It was demonstrated previously that when a large number of core pages is available for replacement, a single core allocation queue is more efficient than a three queue structure. Conversely, when a small number of core pages is available for replacement, the three queue structure is more efficient than a single queue. A dynamic core allocation queue structure which uses the number of available pages as a decision parameter can be developed. If the number of available pages exceeds a set threshold level, all pages released are placed on the first queue. If the number is less than the threshold level, all pages released are distributed among the three queues as was done in the original system. Thus, a single queue is used when it is most efficient, and a three queue structure is used when it is the most efficient. Since the number of available pages is constantly changing, the final organization is a compromise between the one and the three queue structures.

Simulation Series IV was developed to show the behavior of the original system with the addition of the dynamic core allocation queue structure. The results of this series are shown in Table 7. Studies indicated that the optimum threshold level for the test cases used is about 75 pages (i.e., about half of the total number of core pages). Runs 1a, 2, and 3a give the results for job mixes 1, 2, and 3. A comparison of these results with those of the original system (Series I — Runs 1d, 2b, and 3b) shows that the use of the dynamic queue structure has increased the fraction of time available for CPU execution and reduced the response times. Although these are not drastic improvements, they do indicate that the dynamic structure tends to be more efficient than the original structure. The threshold level may have to be changed to provide optimum performance for some other job mix distribution.

By increasing the length of the core allocation queues, more pages are

Table 7. Results of Simulation Series IV

Run	1a	1b	1c	1d	2
Initialization time	60	60	60	60	60
Run time	180	180	180	180	180
CPU data					
Execution	36.1%	36.0%	35.7%	35.0%	39.5%
Overhead	32.9%	33.4%	32.7%	32.1%	34.0%
Idle	31.0%	30.6%	31.6%	32.9%	26.5%
Response time					
Average	1.57	1.59	1.61	1.62	3.07
Range	1-3	1-3	1-3	1-3	1-5
Paging rates					
Read	27	27	27	28	35
Write	51	54	49	44	39
Pages retrieved	43	46	41	41	17
Drum queue data					
Read - average	2.0	1.9	2.0	2.0	2.4
Write - average	1.8	2.0	1.6	9.9	1.7
Device utilization					
Drums	12-18%	12-18%	12-17%	13-14%	14-15%
Nonpaging disks	1-44%	1-44%	1-44%	1-43%	2-44%

Table 7 (Continued)

Run	3a	3b	3c	3d
Initialization time	60	60	60	60
Run time	180	180	180	180
CPU data				
Execution	60.3%	59.1%	60.7%	59.0%
Overhead	32.3%	31.6%	33.1%	32.6%
Idle	7.4%	9.3%	6.2%	8.4%
Response time				
Average	3.75	3.77	3.72	3.81
Range	3-5	3-5	3-5	3-5
Paging rates				
Read	25	23	25	25
Write	33	29	34	30
Pages retrieved	18	17	18	18
Drum queue data				
Read - average	1.8	1.8	1.8	2.0
Write - average	1.6	1.5	1.7	11.3
Device utilization				
Drums	10-11%	8-11%	11%	9-11%
Nonpaging disks	1-33%	1-41%	1-35%	1-42%

available to be retrieved. The low core parameter of the system sets a lower limit on the number of core pages which must be available before additional tasks are given time slices. Consequently, an increase in the value of the low core parameter effectively lengthens the core allocation queues. It should be kept in mind that any increase in the value of the low core parameter reduces the total number of tasks that can be in the system at any one time.

Simulation Runs 1b and 3b give the results for test mixes 1 and 3 with the low core parameter set to 60 pages (double the original value). Since the average number of available pages for test mix 1 is high, increasing the minimum value had almost no effect. Increasing the length of the allocation queues did reduce the overhead for test mix 3; however, the increased idle time caused by the reduction in the number of tasks in the system more than offset any gain produced by the reduced overhead. Runs 1c and 3c were made with the low core parameter set to 15 pages. The performance of the system with test mix 1 was degraded slightly due to an increase in the number of time slices which were ended because of core shortage. The system with test mix 3 was able to make use of the extra pages, but this only amounted to a very small change in performance.

Lengthening the page write queue was accomplished by incorporating into the simulation the ability to specify the minimum length of the queue. Runs 1d and 3d were made with the minimum length of the write queue set to 10 pages. Even though more pages were retrieved from the write queue so that less page writes had to be made, the increased idle time produced a decrease in the performance of the system.

This series of simulations has demonstrated that although the structure of the core allocation queues has an effect on the performance of the system, the benefits to be gained are marginal. No significant improvement in system performance can be made by lengthening the queues.

EVALUATIONS AND CONCLUSIONS

The first part of this chapter is devoted to an objective evaluation of the changes to the paging structure which were proposed in the previous chapter. The conclusions that can be drawn about the paging structure of time-sharing systems in general are discussed in the second part of this chapter. Since this discussion is not concerned with any particular system, it must be somewhat subjective. The conclusions in this chapter should also generate some profitable areas for future investigations.

Evaluation of System Performance

This section is concerned with summarizing and evaluating the proposed changes to the system. Each change is evaluated independently on its own merits, and then the total system performance, with all changes incorporated, is examined.

Reorganized drums

The system idle time was drastically reduced when the two paging drums were reorganized with 4 pages per track instead of the original 4½ pages per track. While it was previously pointed out that this reduction in idle time has little effect on the fraction of CPU time used for execution, there are still notable improvements in system performance. In all cases the average response times have been reduced, thereby improving service to the users. The average size of the drum read queues has also been reduced which indicates that, on the average, fewer tasks are in a page-wait state.

These improvements have not been made totally without cost. Restructuring the drums has decreased the storage capacity of each drum from 900 to 800 pages. The paging drums are used to store pages only for those jobs which are currently in the system. After a job has been completed, its page locations on the drum are released. In this way, the drums act as an extension of main core rather than as permanent storage devices. The total drum capacity must be larger than the average number of pages associated with the jobs in the system so that some space is available to accommodate a large number of jobs. If the total drum capacity should be exceeded, the excess pages are placed on the disk. The simulation results for the test mixes indicated that the average number of pages associated with the jobs in the system is about 1200. Thus, reducing the total capacity of the two drums from 1800 to 1600 pages has not caused any serious problems. Although the number of pages held in reserve for a large number of jobs has been reduced, the gains in system performance are significant enough to justify the drum reorganization.

Drum interrupt scheduling

The recommendation to eliminate some of the interrupts used to set up unnecessary channel programs for the drums was the first of two proposed changes designed to improve the interrupt structure of the system. Because of the reduction in the overhead times, the simulation results for this drum scheduling procedure demonstrated a marked improvement in system performance. The price paid for these improvements is a very slight increase in the size of the drum read and write queues for some job mixes. The increased percentage of CPU time devoted to execution and the reduced response times make it quite clear that the drum scheduling procedure is a beneficial addition to the system.

The simulation results also show that the greatest improvement occurs when low paging rates are involved. It was for this reason that the second change to the interrupt structure was proposed.

Multiple sets of registers

The use of multiple sets of general registers can improve the system performance by reducing the amount of time required to service an interrupt. The drum scheduling procedure reduced the total system overhead by completely eliminating some of the I/O interrupts. The multiple register concept reduces the total system overhead by reducing the overhead for every type of interrupt. System/360 has five types of interrupts. Consequently, the use of multiple sets of general registers produces a more uniform improvement in system performance because it is not so dependent on the paging rate as is the drum scheduling procedure.

Scientific Data Systems Company has been able to supply these multiple sets of general registers in their SIGMA 7 for about $2500 per set including all necessary hardware. The cost of adding five sets of these registers seems rather small considering that over 4% of the total CPU time can be removed from overhead. This performance improvement appears to be sufficient enough to warrant the use of the multiple register structure.

Hardware paging structure

The paging operation in a time-sharing environment is defined well enough to allow the paging structure to be implemented in hardware. The cost of implementing a structure such as the associative register configuration is by no means trivial, but then neither are the benefits that can be gained by using such a structure. The simulation results verify that the hardware structure substantially reduces the overhead required to handle each page read in or written out. In the previous chapter, the percentage of CPU time required by the original system to adjust relocation tables was determined for an extremely high paging rate. For comparison purposes assume once again that pages are read in at a rate of 220 pages per second and written out at a rate of 140 pages per second. The associative register structure requires approximately 340 microseconds for each page just read in and approximately 370 microseconds for each page just written out. Using these values, the overhead required by the associative register structure is 126,600 microseconds per second, or 12% of the total CPU time. This is a major improvement when compared to the 31% required by the original system. Of course, this is an extreme case, but it does demonstrate the potential of the hardware paging structure.

In the original system, each task has associated with it at least one page which contains only the relocation tables for that task. That page must be in main core whenever the task is active. Since there are about six tasks active in main core at any one time, the original structure requires at least six pages of relocation tables to be in core. The hardware paging structure has eliminated the need for relocation tables, thereby increasing the amount of usable main core by about six pages. Thus, in addition to reducing the system overhead, the hardware structure has effectively increased the size of main core by about 4%.

The increased performance due to the overhead reduction and the increased amount of usable core seems to justify the use of the hardware paging structure for the system studied. But more important, if the capabilities of the time-sharing system are to be improved by increasing the paging rate, some alternative to the software structure of the original system must be found. Hardware paging appears to offer such an alternative.

Queue structures

The study on queue structures and queue lengths concluded that there are no major improvements to be made by changes in this area. Some improvement in system performance can be caused by varying the number of allocation queues for the different job mixes. The dynamic queue structure provides an effective way to equalize this improvement across the entire spectrum of job mixes.

Since the priority pages which contain the relocation tables are not used by the hardware paging structure, only two queues are needed to provide a system performance which is nearly identical to that of three queues. Priority pages can be placed on the second queue with the pages which have been changed. Hence, a dynamic two queue structure is used with hardware paging.

Total system performance

To determine the total effect of all the proposed changes on the performance of the system, a final series of simulations was developed. The results of Simulation Series V are given in Table 8. To provide some basis for comparison, the results of the simulations of the original system are repeated in this table.

The improvements in system performance are quite obvious. In all cases the system overhead has been reduced by about one-third. A striking

Table 8. Results of Simulation Series V

Run	1	Series I 1d	2	Series I 2b	3	Series I 3b
Initialization time	60	60	60	60	60	60
Run time	180	180	180	180	180	180
CPU data						
Execution	43.4%	34.8%	49.7%	38.9%	70.9%	59.2%
Overhead	23.1%	33.1%	23.4%	33.7%	22.3%	32.2%
Idle	33.5%	32.1%	26.9%	27.4%	6.8%	8.6%
Response time						
Average	1.30	1.64	2.73	3.11	3.54	3.82
Range	1-3	1-3	1-4	1-6	3-5	3-5
Paging rates						
Read	28	28	35	32	24	23
Write	60	50	42	37	31	29
Pages retrieved	56	41	19	17	22	15
Drum queue data						
Read - average	1.5	2.0	1.8	2.2	1.5	1.9
Write - average	2.1	1.9	2.1	1.6	1.9	1.6
Device utilization						
Drums	18-19%	12-17%	16-17%	12-13%	11-12%	9-10%
Nonpaging disks	1-51%	1-43%	2.43%	2-44%	1-29%	1-35%

Table 8 (Continued)

Run	Series I 4	4
Initialization time	60	60
Run time	180	180
CPU data		
Execution	46.8%	42.2%
Overhead	27.8%	41.1%
Idle	25.4%	16.7%
Response time		
Average	2.57	3.04
Range	2-4	2-5
Paging rates		
Read	61	60
Write	62	56
Pages retrieved	21	13
Drum queue data		
Read - average	2.3	3.4
Write - average	2.2	1.8
Device utilization		
Drums	26-27%	22%
Nonpaging disks	2-53%	1-45%

improvement in the response times is also evident. The amount by which the percentage of CPU time devoted to execution has been increased is dependent on the job mix. The two totally conversational job mixes (Runs 1 and 4) have not been able to make use of all of the time released from overhead. As a result, the system idle time has increased for these two mixes. It is important to note that this increase in idle time is caused by a lack of work rather than by the inability of the system to deliver pages. This can be seen in the reduced size of the drum read queues, which indicates that fewer tasks are waiting for pages to be read. If the system were unable to deliver enough pages, the size of these queues would have increased.

The job mixes which include some batch jobs (Runs 2 and 3) have been able to use the time released from the overhead. The obvious conclusion is that any job mix should have some background jobs if the system is to be utilized to the fullest extent. It must be remembered that the job mixes used as test cases were designed to reflect the extremes in the job load, any real job mix would contain both conversational and batch jobs.

There are no sharp increases in the paging rates of the system. The drum utilization has also only increased slightly. These results indicate once again that the paging drums are capable of supplying pages as required.

All of the proposed hardware changes to the original system would add about 5% to the total system cost. The performance gains achieved by these changes should more then justify the increased cost.

Conclusions

The object of this entire study has not been to only propose specific changes for any particular time-sharing system. Rather, it has also been to illuminate those areas of the paging structure which appear to offer the largest potential gains in system performance. In the light of these objectives the following conclusions have been drawn.

The need to efficiently deliver pages from secondary storage to main memory is of primary importance in a time-sharing system. Contrary to the conclusions of some investigators, such as Lauer (10), this study concludes that the paging drums are able to maintain a satisfactory level of performance, and that a more efficient device, such as LCS, is not needed. While it is conceded that LCS can deliver pages at a higher rate than the paging drums, all evidence indicates that the additional cost of LCS is not warranted simply because the system could not make use of the increased capabilities.

The experimental findings of this study support the conclusion that there is much to be gained by changes to the scheduling and interrupt structures of the system. The improvements which can be made are not limited to just time-sharing systems but can also benefit the performance of conventional systems. That being the case, the scheduling and interrupt structures appear to offer very promising areas for investigation.

The hardware paging structure is felt to be the most significant improvement that can be made to the time-sharing system. If time-sharing is to be taken seriously, it would seem appropriate to design the system to fit the peculiar needs required for time-sharing, rather than to adjust the time-sharing philosophy to fit a conventional structure.

Perhaps a separate special-purpose processor could be designed to perform the paging and scheduling operations. Both of these operations are fairly well defined and could be implemented almost totally in hardware. The special processor would be used in conjunction with a fairly conventional processor to provide the system with time-sharing capabilities. This processor would have its own access to main memory and could be located on the main frame of the conventional structure. Because of its well-defined functions, it could be a fairly low cost processor. The use of such a structure would provide parallel operation of paging and scheduling with the other system functions.

The simulations have revealed that varying the structures and lengths of the many queues in the system does not greatly affect the system performance. Most of the analytical studies of time-sharing systems have concentrated on the queue structures. A number of other investigations have concerned themselves with determining an optimum replacement algorithm. While there is no doubt that there are benefits to be gained by modifications to these areas, this study raises the question of just how important these gains are with respect to the total system performance. It would appear that there may not be as much to be gained by changes to the queue structures and replacement algorithms as is thought in some circles.

There are a number of other important areas in the time-sharing philosophy. The scheduling or ordering of jobs to be processed is important for the efficient operation of a time-sharing structure. Assignment of job priorities is one area open for investigation. Perhaps a dynamic priority structure could be designed in which a program that has just been compiled is given a higher priority because its pages are in main core ready for immediate execution. Numerous variations on this design are possible.

Another area closely related to job scheduling is time slice allocation. Time slice allocation refers to the amount of processor time that should be given to any particular job when it is scheduled to receive CPU cycles. A few questions to be answered are: Should the allocation be made to insure the progress of the system at the expense of some jobs? Must the allocation be fair? Should all jobs in the same priority class be allowed to make equal progress? What is an appropriate time slice length?

It should be apparent that there are many areas still to be explored which may or may not significantly improve the system performance. The use of a simulation program has proved to be an extremely effective way to investigate the many aspects of a time-sharing computer system.

Concluding Remarks

In order to compare the performance of one system with the performance of another, some "figure of merit" would be desirable. The amount of CPU time devoted to execution, the system overhead time, the system idle time, the response time, the paging rates, etc. are all important factors in evaluating system performance. It soon becomes clear that no single measure can be found to evaluate the total performance of a time-sharing system. Therefore, no attempt has been made to define a "figure of merit". The concept of performance gained per dollar spent has also been avoided for much the same reason.

The validity of any simulation is always open to question. A great deal of care was taken in the selection of the test cases in order to insure valid results, and the simulation selected has been shown to accurately reflect the behavior of the Model 67. For these reasons, it is felt that the variations in system performance are valid and that the conclusions drawn from these results are correct.

BIBLIOGRAPHY

1. Comfort, Webb T. A computing system design for user service. Joint Computer Conference Proceedings, Fall 27: 619-626. 1965.

2. Fife, Dennis W. An optimization model for time-sharing. Joint Computer Conference Proceedings, Spring 28: 97-104. 1966.

3. Fine, Gerald H., Jackson, Calvin W. and McIsaac, Paul V. Dynamic program behavior under paging. ACM National Meeting Proceedings 1966: 223-228. 1966.

4. Fine, Gerald H. and McIsaac, Paul V. Simulation of a time-sharing system. Management Science 12: B180-B194. 1966.

5. Gibson, Charles T. Time-sharing in the IBM system/360: model 67. Joint Computer Conference Proceedings, Spring 28: 61-78. 1966.

6. Huesmann, L. R. and Goldberg, R. P. Evaluating computer systems through simulation. Computer Journal 10: 150-156. 1967.

7. International Business Machines. IBM system/360 model 67 functional characteristics. Form A27-2719. Place, IBM. 1966.

8. Kilburn, T., Edwards, D. B. G., Lanigan, M. J. and Sumner, F. H. One-level storage system. I.R.E. Trans. Electronic Computers 11: 223-235. 1962.

9. Krishnamoorthi, B. and Wood, Roger C. Time-shared computer operations with both interarrival and service times exponential. Journal of the ACM 13: 317-338. 1966.

10. Lauer, Hugh C. Bulk core in a 360/67 time-sharing system. Joint Computer Conference Proceedings, Fall 31: 601-609. 1967.

11. Lindquist, A. B., Seeber, R. R. and Comeau, L. W. A time-sharing system using an associative memory. I.E.E.E. Proceedings 54: 1774-1779. 1966.

12. Mendelson, Myron J. and England, A. W. The SDS sigma 7: a real time time-sharing computer. Joint Computer Conference Proceedings, Fall 29: 51-64. 1966.

13. Nielsen, Norman R. The analysis of general purpose computer time-sharing systems: doc. 40-10-1. Stanford, California, Stanford Computation Center. 1966.

14. Nielsen, Norman R. An approach to the simulation of a time-sharing system. Joint Computer Conference Proceedings, Fall 31: 419-428. 1967.

15. Nielsen, Norman R. The simulation of time-sharing systems. Communications of the ACM 10: 397-412. 1967.

16. Scherr, A. L. An analysis of the time-shared computer systems: MAC-TR-18. Cambridge, Massachusetts, Massachusetts Institute of Technology. 1966.

17. Smith, J. L. An analysis of time-sharing computer systems using Markov models. Joint Computer Conference Proceedings, Spring 28: 87-104. 1966.

18. Wallace, Victor L. and Rosenberg, Richard S. Markovian models and numerical analysis of computer system behavior. Joint Computer Conference Proceedings, Spring 28: 141-148. 1966.

ACKNOWLEDGMENTS

The author would like to thank his major professor, Dr. Robert M. Stewart, for his advice and encouragement during the preparation of this dissertation. The author also gratefully acknowledges the support given by the National Aeronautics and Space Administration in the form of a graduate fellowship and by IBM Corporation in the form of an educational leave of absence. Credit for typing this manuscript is given to Mrs. Margi Wibholm. In addition, the author owes a special debt of gratitude to his wife, Sandra, whose help and understanding made this undertaking possible.

APPENDIX

This appendix is designed to give the reader a brief description of the IBM System/360 Model 67 Time-Sharing System. Since the design of the Model 67 is similar to other systems in the 360 line, only those characteristics of the system which are unique to the time-sharing capabilities are discussed.

The basic mode of operation in the Model 67 is time-sliced multiprogramming. The basic unit of control in the system is a task. Every item of work to be performed by the system (e.g., programs, etc.) is set up as a task. Each task has its own virtual memory with its own set of relocation tables.

Dynamic relocation is defined as the method by which virtual storage is translated into actual storage. Virtual storage is all storage which can be reached by the logical address. The logical address is that address known to the program. For a 24 bit logical address, there are 2^{24} = 16,777,216 byte locations in the virtual storage of each task. The Model 67 also has provisions for 32 bit addressing. The actual address is the address presented to physical memory for the reference.

Relocation allows a program to be broken up in memory into pages; and it provides a means of moving programs into and out of main core, each time relocating the program pages at different physical locations. Each page in the Model 67 contains 4096 bytes. The low order 12 bits of the logical address are used to indicate the address of each byte within a page. Since relocation is only concerned with page addresses, the low order bits of the logical address are not translated. Therefore, the low order 12 bits of the logical address and of the actual address are the same.

A group of 256 pages is called a segment. The high order bits of the logical address indicate the segment number; the next 8 bits indicate the page number; and the 12 low order bits indicate the byte address.

The relocation operation is accomplished under program control with all relocation tables in main core. Whenever a task begins operation, the system's table register is loaded with the starting address and the length of the segment table associated with that task. To translate a logical address into the actual address, the segment part of the logical address is first used to locate the corresponding entry in the segment table. The entry in the segment table contains the starting address and length of a page table. The page part of the logical address is then used to locate the desired entry in the page table. The page table entry contains the actual address of the page in core. If the page is not in core, a bit in the page table entry indicates this fact.

Thus, it is necessary to make two references to main memory (i.e., one to obtain the segment table entry and the other to obtain the page table entry) for each address translation. The total time required for the translation operation is about 2.1 microseconds. Since this translation must occur for every instruction fetched, the amount of time required for this look-up procedure would be objectionable. For this reason an associative memory is used to reduce the translation time to an acceptable level. This high-speed associative memory consists of eight registers which contain the segment and page portions of the logical addresses of the most recently used pages along with their actual addresses. When an address is to be translated, the segment and page parts are compared simultaneously with the contents of all eight registers. If the proper page is found, its address is immediately available; if not, the previously described method of address translation must be followed. The associative memory is updated to contain only the eight page references most recently used. The time required to interrogate the associative memory is only 0.15 microseconds.

All tasks in the system are grouped into one of three priority levels. First level tasks (i.e., bulk I/O operations) have the highest priority and are given CPU time immediately when ready and for as long a time as necessary. Second level tasks (i.e., conversational jobs) are given a time slice every operational cycle time (OCT). The length of the OCT is a monitor parameter which together with the length of a time slice controls the task scheduling in the system. If the OCT is too short, it will be extended to give every second level task a time slice. Third level tasks (i.e., batch jobs) are given a time slice if all second level tasks have received their time slice and time still remains in the OCT.

To determine which tasks are eligible to receive CPU time, two pointers, called the front wall and the commutator, move from the beginning of the second level task list toward the end of the third level task list each OCT. Tasks between the two pointers are given CPU time. Whenever a task has received its time slice, it is placed in a time slice end status. When the commutator encounters a task in the time slice end status, it marks the task's pages available for replacement and advances to the next task.

Movement of the front wall is controlled by the low core parameter. Whenever a task is included in the wall, its estimated page requirements (i.e., pages used during its previous time slice) are deducted from the extra page count. Whenever the commutator advances, the number of pages released is added to the extra page count. In this way, the extra page count is the number of pages not expected to be used by the tasks between the commutator and the front wall. The low core parameter is the number of pages to be saved for a larger than expected need. The front wall will not move ahead if the estimate of extra pages is less than the low core parameter.

Detailed descriptions of the features just presented, as well as descriptions of the more conventional aspects of the Model 67, are given in the papers by Gibson (5), Comfort (1), and the IBM manual (7).