PROGRAM AND JOB-STREAM CHARACTERISTICS IN THE MICHIGAN TERMINAL SYSTEM

by

KENNETH H. BOWLER

B. Eng., Carleton University, 1969

A thesis submitted in partial fulfilment of
the requirements for the degree of

MASTER OF SCIENCE

in the Department
of
COMPUTER SCIENCE

We accept this thesis as conforming to
the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

July, 1972

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the Head of my Department or by his representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver 8, Canada

Date Aug. 16, 1972

ABSTRACT

There has been little published about the characteristics of computer jobs running on modern time-sharing computer systems, due largely to the lack of appropriate programs and equipment necessary to measure the parameters involved. In this thesis, measures are presented for some of the important characteristics of jobs. The Data Collection Facility, which is part of the Michigan Terminal System, was used to this end. The Michigan Terminal System is a time-sharing operating system for the IBM 360/67 computer, and supports batch and terminal users simultaneously.

Chapter 1 gives an outline of the problem, and other work which has been done in this line. It also contains a reasonably detailed description of the Michigan Terminal System. In Chapter 2, measurements of requested CPU service, CPU service obtained, system and user response times, I/O delays, and page waiting times are given. Chapter 3 outlines the storage requirements of jobs, and gives a model which will generate profiles of storage required by jobs over their running times, which are very similar to profiles observed for actual jobs. Some discussion of the results is given in chapter 4, and also a simple model of the system is shown which might be used in a simulation study employing measurements taken in this study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I wish to express my thanks to Dr. D.A.R. Seeley for his helpful assistance during the preparation of this thesis, and to the National Research Council of Canada, and the Department of Computer Science for providing financial assistance during the course of my studies. I wish also to thank my wife, Joan, who provided much encouragement, and spent many hours keypunching and proof-reading for the preparation of this work.

CHAPTER 1 INTRODUCTION

1.1 Preamble

In the past, program behavior and the characteristics of the job mix running on a computer system were relatively easy things to observe. The machines were slow, and usually ran as single thread batch systems, so that one program was processed to completion before another was allowed on the system. It was possible on the early machines to time the periods of processor activity and waits for I/O with a stop watch since there was no overlap of processing and I/O, and the times of the events were of the order of seconds. On faster machines the measurements still could have been easily made using some sort of hardware device to do the measuring. With the advent of multiprogramming, measurements of program behavior became much more difficult. Hardware devices could still measure easily the processing and I/O intervals on the system, but, as stated by Drummond [19], data dependent characteristics such as job identification, data set identification, and the origins of requests are not readily available. In effect, what these devices measure is system behavior and not program behavior.

The characteristics of single jobs can be determined using a hardware device by running the jobs alone on the system one at a time, as was done by Cheng [12]. This method is tedious, and ties up the system for long periods of time, which makes it impractical to use in a busy environment.

Other complications for measurement also arise. Not all the I/O is associated with the jobs that are running. In an effort to improve the utilization of various resources of the

system, jobs can not be allowed to tie up certain devices for long periods of time. (In particular card readers and printers). If a job is allowed to possess a printer, it will tie it up for the duration of its running time on the system, (since the output for one job must be kept together) even though it prints only a few lines. Other jobs requiring the printer can not be dispatched. The solution is to have the job write its output to a file on disk (or tape), and after it is finished have the system take care of the printing. Similarly card input is read into a disk file, and the job can read the file as it progresses without tying up the card reader for long periods. So we have the system, as well as the jobs running, doing I/O. Time-sharing adds the additional complication that a job running on a CPU does not need to initiate an I/O operation to lose control of its CPU.

On a multiprocessing system more than one job can be running on the system at once and the task of associating events with the job involved is a very complex one for an external hardware device. (A good description of the type of information gathered by a hardware device may be found in [7]). For a software device, this task is much simpler, and various such devices (see [3,47] for examples) have been written for some computer systems. Software devices have such information, as job identification and the statistics that the system keeps about the jobs, easily available to them (the device may be the part of the system that collects these statistics), but require for their operation some part of the system's resources. This fact implies that they can affect the measurements that they are

taking, and this must always be considered when using such a device.

Two types of software measurement techniques have emerged in recent years. One approach is a sampling [19,47,48] form of measurement in which various statistics kept about the jobs are sampled periodically and the results recorded. (CPU time, I/O requests, page reads etc.) It is easy to see that a well devised program to do this sampling will not affect the measurements that it takes, as long as it does not sample too frequently. It might use only a few milliseconds of CPU time every few seconds, but there is a considerable loss of detailed information using this technique. Mean times for events may be obtained, but information about distributions of event times, and the sequences of events [19,40] will be lost.

The second type of device (program) monitors continuously, and every important event happening in the system can be recorded. (See [3,39] for examples.) In order that this technique does not completely overwhelm the measurements that it is taking, it must be a part of the supervisor program of the system, since the information is easily available at this level, and may be obtained at small expense. A program to do this type of monitoring, external to the supervisor, would virtually have to interpret every action that the supervisor takes, to see if some information needs saving. Such interpretation would cause considerable degradation of system performance, because of the processor time neccessary to accomplish it. This makes this technique much more difficult to implement than that of

sampling. The disadvantage of the continuous monitoring technique is that enormous quantities of raw data are produced, and the cost of reducing these data can be quite large. The sampling method on the other hand, produces manageable quantities of specific data which are more easily reduced.

Little data has been published on the characteristics of jobs in a time-sharing environment, other than their behavior with respect to paging, and paging strategies. Brawn and Gustavson [8] measured the effect of restricting real memory size on the running time of various programs, and they, as well as Hatfield [26], studied the effect of restructuring programs to localize their memory references, on the amount of paging required. Coffman and Varian [13] were interested in the times between inter-page references (as were Fine et al [22]), and also page residence times in main memory. Unfortunately their study was done using unrealistic restrictions on core size.

Freibergs [23] has measured the times taken between calls to the supervisor program for services, for a few types of programs (a FORTRAN compiler, a list processing program, etc.), and has measured a distribution for the maximum memory requirements of jobs on McGill's IBM 7044. Scherr [45] and Schwetman and DeLine [44] have measured distributions for user response time (think time) and system response time for two different systems. Scherr as well has obtained a distribution for the amount of processor time demanded by tasks between interactions on Project MAC's CTSS system. Baskett et al [5] have measured a distribution for disk I/O delays.

The problem of page transfer times has been well covered in the literature [14,39], but in the Michigan Terminal System (MTS), which is the operating system for the University of British Columbia's IBM 360/67 computer, a large portion of the page faults do not require page transfers from the drum. These faults result from the first reference to a new page. MTS does not allocate any real page to a job until after that page has been referenced. The first reference to a new page causes a page fault which can be dealt with by immediately allocating an available real page, and no drum transfer occurs. (See section 2.4.)

Parameters like the total processor time required by jobs, and job inter-arrival times are highly situation dependent, but can be easily obtained if the system keeps any accounting statistics.

Although models have been proposed for the real core requirements of computer jobs, notably Denning's [16], almost nothing exists concerning jobs' virtual memory requirements. By virtual memory requirements, we mean the total instantaneous storage requirements, which is a dynamic quantity in most time-sharing systems. Lehman and Rosenfeld [31] have modelled the dynamic quality of storage demands to some extent by breaking their model of a job into job steps, and giving each job step a different memory requirement. This is still not a complete enough model, however, since the storage requirement of a job in a time-sharing environment that allows dynamic program loading and storage allocation, may vary considerably over a single job

step.

Empirical measurements of program storage requirements over their running times, and models of these requirements are needed for a more thorough understanding of program behavior, and for simulation studies.

1.2 Description Of The Michigan Terminal System

This description is taken in large part from M. T. Alexander's paper "Time-sharing supervisor programs." [2]

1.2.1 The Michigan Terminal System (MTS)

MTS is a general purpose time-sharing system designed for the IBM 360/67 computer, and will support up to four processors running in parallel. The term MTS is used to refer to both the operating system as a whole, and to a re-entrant job program in the supervisor program, UMMPS, which gives the user the capability to run programs and to manipulate files from remote terminals or from batch.

MTS provides:

1) An easy to use command language to cause the running and monitoring of programs and to create, destroy and otherwise manipulate files, and for other communication with the system.

2) Two types of files, sequential and line files. Line files may be accessed randomly by line number, but sequential files can only be accessed sequentially. Line files reside on disk while sequential files may either be on disk or datacell .

3) A dynamic program loader which allows a program to load another during execution.

4) Extensive subroutine libraries and the capability to load programs selectively from either user defined or system libraries to resolve undefined external symbol references in a loaded object module.

5) Many language processors such as FORTRAN, PL/1, APL, ALGOL, WATFIV etc.

Together MTS and UMMPS form an easy to use but powerful time sharing system.

1.2.2 University Of Michigan Multiprogramming Supervisor (UMMPS)

The UMMPS supervisor is a time-sharing supervisor for IBM 360/67 supporting up to four processors. It consists of a set of subroutines for processing interrupts, and is only entered by interrupt, either hardware, or internal such as calls to the supervisor (the 360 SVC instruction), and the various program

interrupts. All interrupts are processed as close to completion as possible, and a queue is maintained for all those things that must be postponed. When there is nothing more to be done at the moment, the supervisor gives control to a ready task (if any). UMMPS runs with interrupts disabled, which means that it cannot itself be interrupted, and also with the relocation mechanism turned off. It also allows some tasks to run with relocation turned off which permits them to reference any real core location. This ability is restricted to special tasks such as the Paging Drum Processor (PDP) and tasks controlling unit record equipment, which need to be able to reference all of main memory. User tasks cannot be given this power, since the relocation hardware is used to provide inter-task protection.

1.2.3 Processor Scheduling

UMMPS maintains only one CPU queue which all the processors scan looking for available work. Tasks in the system can be in one of four states as far as the supervisor is concerned.

1) Running
2) Ready
3) Waiting
4) Page wait

All tasks which are running or ready and some waiting tasks are on the processor queue, and when the queue is being scanned, the first ready task found is given a processor. A task which uses up its time slice (100ms) is given a new one and placed on

the bottom of the queue (but ahead of any waiting tasks which are on the queue). This is the only time that a task is given a new time slice. A task which initiates an I/O request or requests a page not in main storage is not replaced on the processor queue, but when it becomes ready it is added to the top of the queue. Whenever a task is added to the processor queue for any reason, it is added to the top of the queue. This gives very quick service to interrupts, and allows tasks which are acquiring real storage to do so quickly.

It is possible for a task to wait for some bits in a byte in virtual storage to become all zero. This is used to allow different tasks to communicate, and also to allow tasks to wait for asynchronous interrupts. In the latter case, the interrupt handler sets the bits. When a task waits for a byte that is not in its private virtual memory, the task remains on the CPU queue and the byte is tested each time the task is the next to be given a processor. If the bits are zero, the task gets a processor, if not the next task is examined. The CPU queue is re-ordered so that all such waiting tasks follow any ready task in the queue. If the byte is in the tasks own virtual storage, the task is removed from the processor queue, since some interrupt must occur to allow the bits to be changed.

The fact that a task that quits waiting is placed at the top of the processor queue gives very fast service to interrupts and allows ordinary tasks to control I/O equipment. For example, the PDP is not treated specially as far as scheduling is concerned, but it can easily keep up with the drum

interrupts, and such tasks can be given very large time slices which will prevent them from being forced to the end of the processor queue.

In an effort to prevent too many tasks from competing for main storage, UMMPS has a mechanism for making certain tasks privileged or non-privileged. There are therefore, three classes of tasks in UMMPS:

1) Neutral
2) Privileged
3) Non-privileged

The term privileged applies only to the amount of processor time that a task gets, and the amount of paging that it is allowed to do.

When a task is first added to the processor queue, it is classed neutral, but if it attempts to accumulate more than a certain threshold of pages, a decision is made. If there are fewer than the maximum allowed number of privileged tasks, it is made privileged, and is given a time slice equal 4*K*U, where K is the number of additional privileged tasks allowed at that point in time, and U is the basic time slice. The total number of privileged tasks allowed is 7 for 1 core box, 12 for 2 core boxes, 15 for 3 core boxes, 22 for 4 core boxes, and 30 for more than 4. The decision threshold is initially set to about 30 pages. (This mechanism is currently in the process of revision. Further information may be obtained from the University of Michigan Computing Center). The decision threshold for the next task is then lowered, and this one is allowed to obtain as many

real pages as it wants. If there are already the maximum number of privileged tasks when the decision point is reached, the task is made non-privileged, which removes that task temporarily from contention for the CPU's and main storage. A privileged task remains privileged until it uses up its extended time slice or enters the wait state (except page-wait). At this point it is made neutral and the decision threshold is raised, allowing a non-privileged task, if any, to be made privileged. Non-privileged tasks are never made neutral without first becoming privileged.

Two queues are also maintained for each task, a task CPU queue and a task wait queue. The task CPU queue is used to keep track of multiple levels of execution. Each entry under the top one represents a subtask which has been interrupted but may later be resumed. Some I/O interrupts cause a new entry to be added to the top of the CPU queue, and after their processing the next lower level resumes processing. A wait entry is maintained in the wait queue for each level of the CPU queue at which a wait is outstanding. The end of a wait at any level can be recorded by removing the wait queue entry at that level.

Supervisor routines exist to remove all entries from the task CPU queue but the top one or to remove the top entry. An example of the above is illustrated by the following situation. A user is running a program at a terminal and enters an attention interrupt. This forces the saving of status at the initial CPU queue level and a new entry to be added to the top. He may now enter commands at the terminal, then he can enter a

command to restart the program (remove the top entry), or he can elect to run another program, which causes the previous entry or entries to be removed.

1.2.4 Storage Scheduling

In MTS, virtual memory is implemented using drum storage devices (IBM 2301's). These drums have 200 tracks and are capable of storing 4.5 pages per track. In order to obtain the best utilization of the drums, they are arranged with 100 logical tracks having 9 pages each. Two revolutions of the real drum are needed for one revolution of the logical drum. The 9 slots around the circumference of a logical track are called sectors.

The task of virtual memory management in MTS is divided between UMMPS and a special job called the Paging Drum Processor (PDP). The PDP takes care of the actual reading and writing of pages to and from main memory, while it is the responsibility of the supervisor to decide which pages need moving, UMMPS and the PDP communicate via Page Control Blocks (PCB's) and there exists one PCB for each virtual page. Each PCB contains all the information concerning the status of its page, and may be linked at any time on one of four queues:

1) Page In Queue (PIQ) contains the PCB's for all page requests that PDP has not yet started reading

in.

2) Page In Complete Queue (PICQ) contains the PCB's of all pages that the PDP has read or a new page for which reading was unnecessary but of which the supervisor has not been notified yet.

3) Page Out Queue (POQ) contains PCB's for all pages in main storage that can be removed.

4) Release Page Queue (RPQ) contains PCB's for all pages which have been released by the tasks that own them, but which the PDP has not released yet (the PDP must be notified so the drum space can be released).

All pages that are read in are immediately placed on the POQ at the top. When the PDP requests some pages to be written, the supervisor checks the amount of free space available and if it is sufficient (about 20% of main memory), no pages are given to the PDP. Otherwise the supervisor starts scanning the POQ for a page to be written out. If the page under consideration in the scan has been referenced since the last scan, the reference bit is reset and the PCB will be placed on the bottom of the POQ. If the page was not referenced it is given to the PDP for writing out. This process continues until enough pages are found for the PDP.

When a page is requested by a task, the supervisor places the PCB on the bottom of PIQ and if the PDP is not already running, it is started. The PDP places the PCB on one of nine sector queues, corresponding to the sector that the page occupies on the logical drum, and then requests pages to be written from the supervisor to fill any of the nine sector queues which are empty. If any of these pages to be written have not been changed since last writing, the supervisor will be notified that the main storage for that page is free and another page is requested for writing. When all sector queues are filled as much as possible, a channel program to do the reading and writing is constructed by the PDP and executed. The PDP will then place the PCB's for the pages read on the PICQ and notify the supervisor that main storage for the pages written is free. The supervisor takes PCB's from the PICQ and places them on the POQ at the top and restarts the tasks involved. The PDP handles the job of constructing new channel programs in parallel with processing old ones so that the process of reading and writing is continuous.

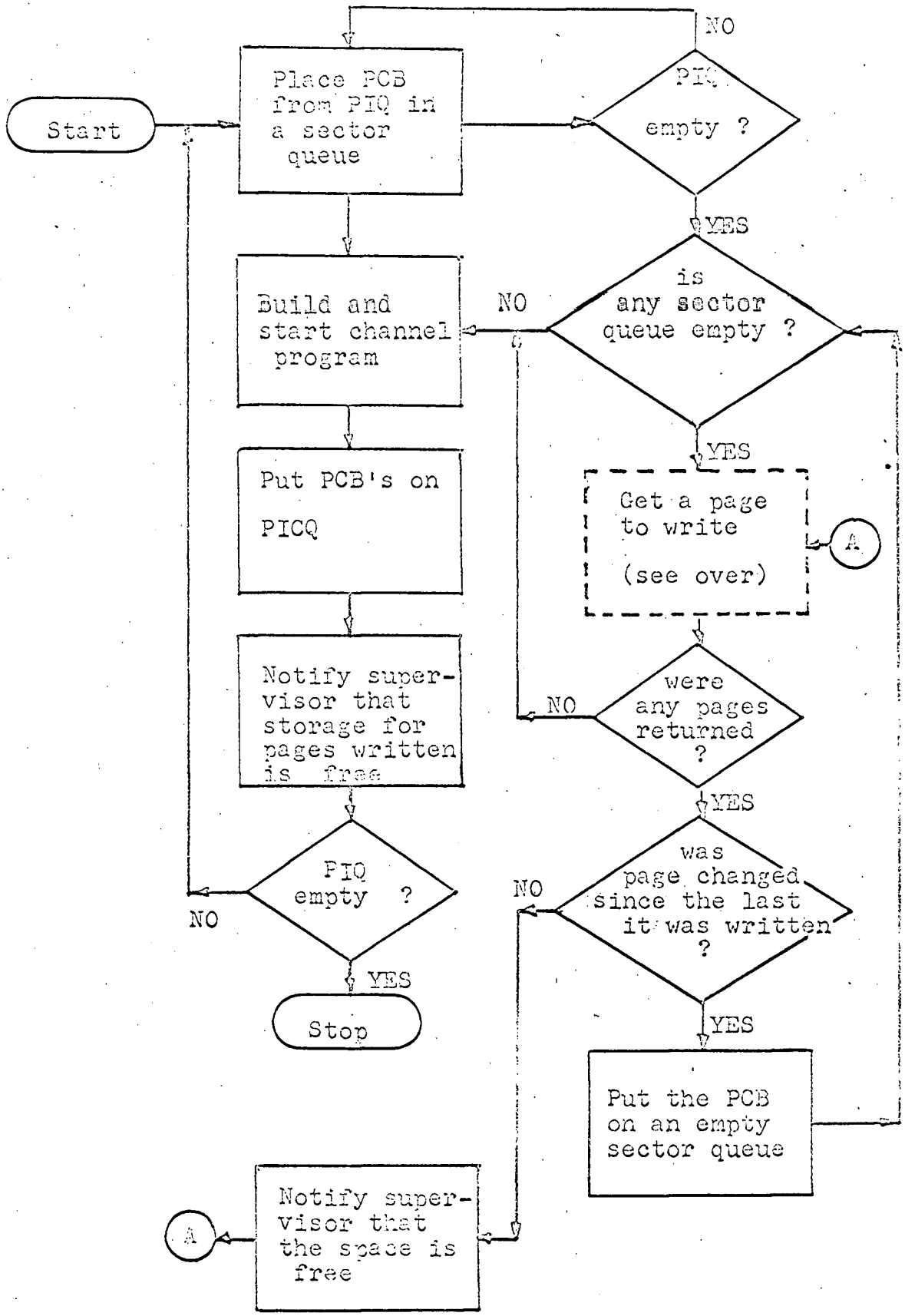


Figure 1.1 Flowchart of storage scheduling

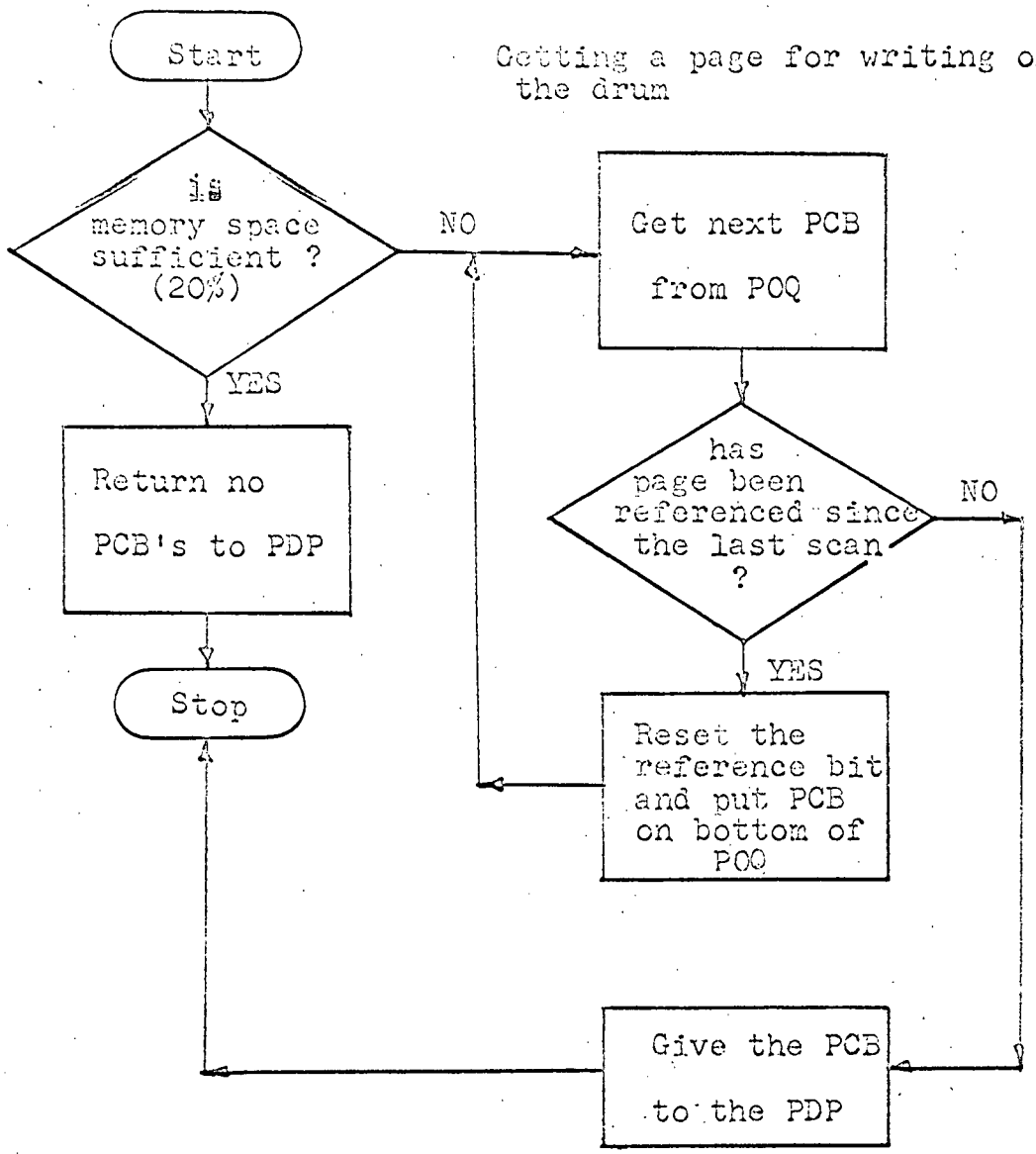


Figure 1.1 (b) Flowchart of storage scheduling

1.2.5 General

Flow of tasks in MTS at the University of British Columbia is depicted in fig. 1.2. Both processors scan the single queue looking for work, and the first ready task encountered is given a CPU. A task which is running on a CPU can subsequently stop for any of the following reasons:

1) It requests some I/O
2) It stops to wait for some byte to be set to zero
3) It references a page not in main storage
4) It exhausts its time-slice
5) It is pre-empted

Tasks which are waiting for an interrupt to occur, to signal the end of an I/O operation, or the completion of a page transfer are not on the processor queue, but immediately seize one of the CPU's when the interrupt occurs, pre-empting a task that is currently running. The pre-empted task is placed on the top of the queue, and is the next to receive service provided that another task is not pre-empted in the interim.

There are no special priorities assigned to any tasks in MTS, so that all tasks compete on an equal basis for system resources.

Most of the occurrences of stoppages in processing for the second reason above have to do with terminal I/O for interactive tasks. In this study, the time that the device support routine (DSR) for the terminal is entered to get an input line or to write an output line, is taken as the start of the terminal I/O

operation. The time that an exit is made from the DSR is taken as the finish of that operation, and all processor activity for that task between is ignored. In effect, the entry of the DSR is taken as the beginning of a wait I/O operation for the terminal, and the exit from the DSR is taken as the completion of that wait.

When a task releases a virtual memory page, the PDP is started by the supervisor (if it is not already running) in order to notify it that the drum space may now be used for other page write operations. If the PDP was not running, it will pre-empt the task that released the page for the time taken to modify the appropriate tables.

1.3 Problem Area

This study uses the Data Collection Facility called the DCF (continuous monitoring type) available in the MTS operating system to study the characteristics of a typical university job stream, the purpose being to allow the production of an accurate model of the job stream for simulation. It has been possible to measure the times taken for many of the events occuring in the system, in particular:

1) The amount of processor time requested by a task when it obtains a CPU
2) The amount of processor time that it actually gets

3) The waiting times for I/O, and pages

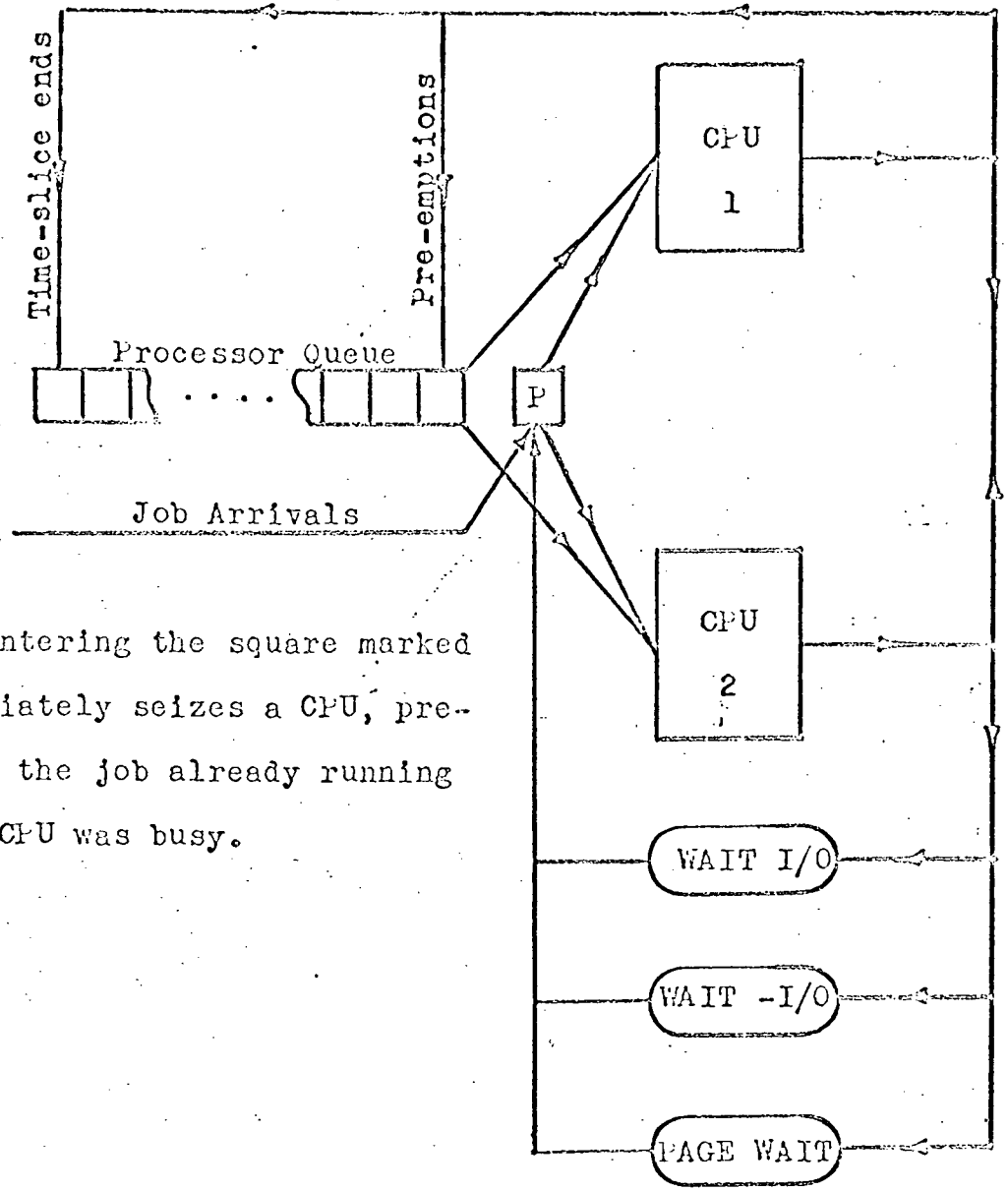
4) The interarrival times for jobs into the system

5) The total processor time used by jobs

6) The storage requirements of jobs

Some system dependent characteristics (CPU time required to process interrupts) have been removed as far as possible from the measure of CPU service required by jobs, so that this measure should be applicable on other systems.

Data was collected for the study on two magnetic tapes, called tape 1 and tape 2. The data on tape 1 was collected on a Saturday afternoon, and represents a time of light load on the system, while the data for tape 2 was collected on a Friday afternoon, which is a time of moderately heavy load. Together the tapes contain data for about 300 jobs.



A job entering the square marked P immediately seizes a CPU, pre-empting the job already running if the CPU was busy.

Figure 1.1 Flow of tasks in MTS

CHAPTER 2- SOME BASIC CHARACTERISTICS

2.1 Requests For CPU Service

In the past, it has been difficult to obtain measurements of the time that computer jobs run between successive I/O operations, because of the lack of appropriate monitoring devices and programs (see [31,33]), but with the development of tools like the DCF in MTS [3, Appendix A], such measurements may now be readily obtained. The amount of processor time that a job requests from the termination of one I/O operation until the start of the next, will be called a *requested CPU interval.*

Many simulation studies have assumed that the distribution of these intervals is exponential [33,34,46], or hyper-exponential [21], since the use of these distributions allows easy generation of random samples, and analytic methods exist for studying systems governed by them.

Measurements of requested CPU intervals in this study show considerably more variability than they would if they were exponentially distributed, and even a hyper-exponential distribution does not provide a close fit to the observed distributions. Requested CPU intervals for batch jobs had a mean of 23.6 ms and a standard deviation of 73.6, while interactive tasks had a mean of 17.0 ms with a standard deviation of 76.4.

A Weibull distribution provided the closest fit for the observed intervals for batch jobs, and a hyper-exponential distribution for those of the interactive jobs. (See figs. 2.1, and 2.2). The Kolmogorov-Smirnov goodness of fit test is

used throughout, and gave maximum deviations for the observed values from the fitting curve of 0.1673 for batch jobs, and 0.0802 for interactive jobs. The probability of finding a deviation as large as 0.0200 (batch) or 0.0119 (interactive) for samples this large is less than 1% if the fit is to be considered good. This value will be referred to in future as KS(.01).

The Weibull distribution is useful for fitting sample distributions of this type (with the appropriate choice of parameters), since one can obtain a distribution with more very large values, and more very small values than with a hyper-exponential distribution having the same mean and standard deviation. The problem with using the Weibull distribution, however, comes in choosing the parameters. (See Appendix B for details).

The CPU time requested by interactive jobs between terminal I/O operations is larger than the requested CPU intervals on the average since several instances of disk or tape I/O may occur between successive terminal operations. The mean observed time was 58.6 ms with a standard deviation of 669, and a Weibull distribution gave the closest fit with a maximum deviation of 0.0942. (KS(.01)=0.0211) (see fig. 2.3) The Weibull distribution fitted by the method described in Appendix B had a mean of 27 ms, and a standard deviation of 48 which are much less than the observed values for the mean and standard deviation, however, the observed distribution had a few very large sample values (as great as 30 seconds of CPU time), which

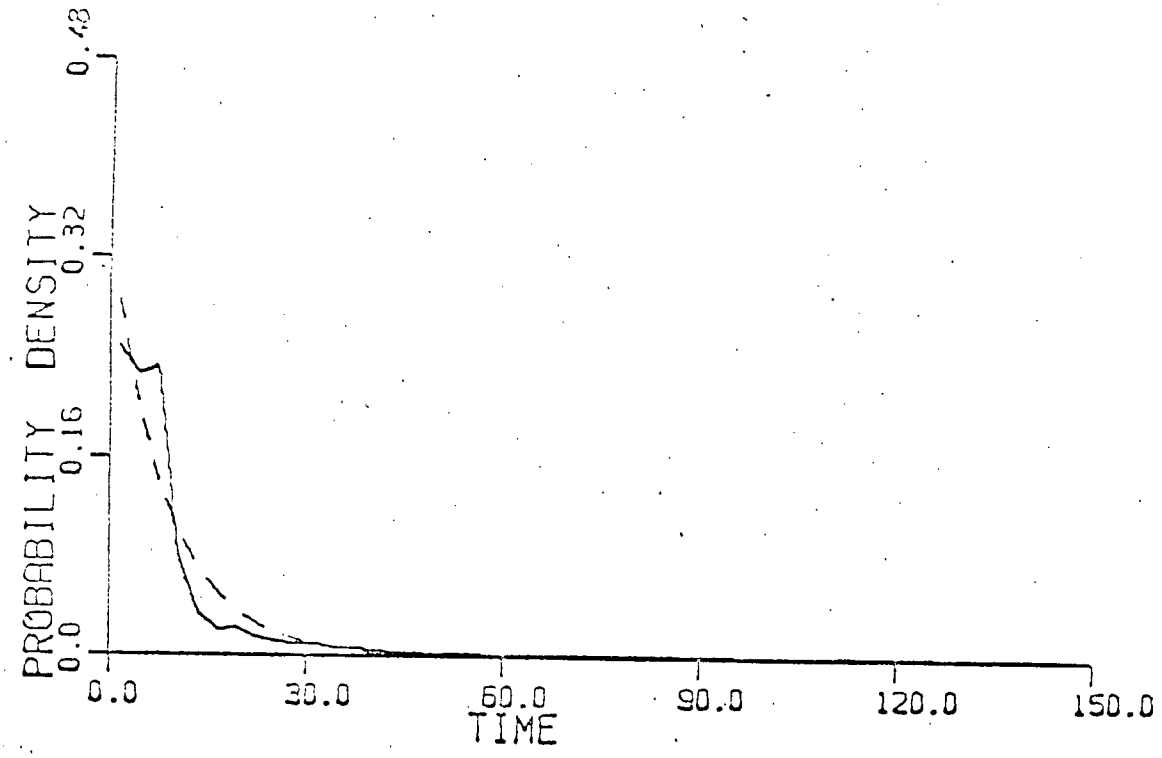


Figure 2.1 Requested CPU intervals (interactive)

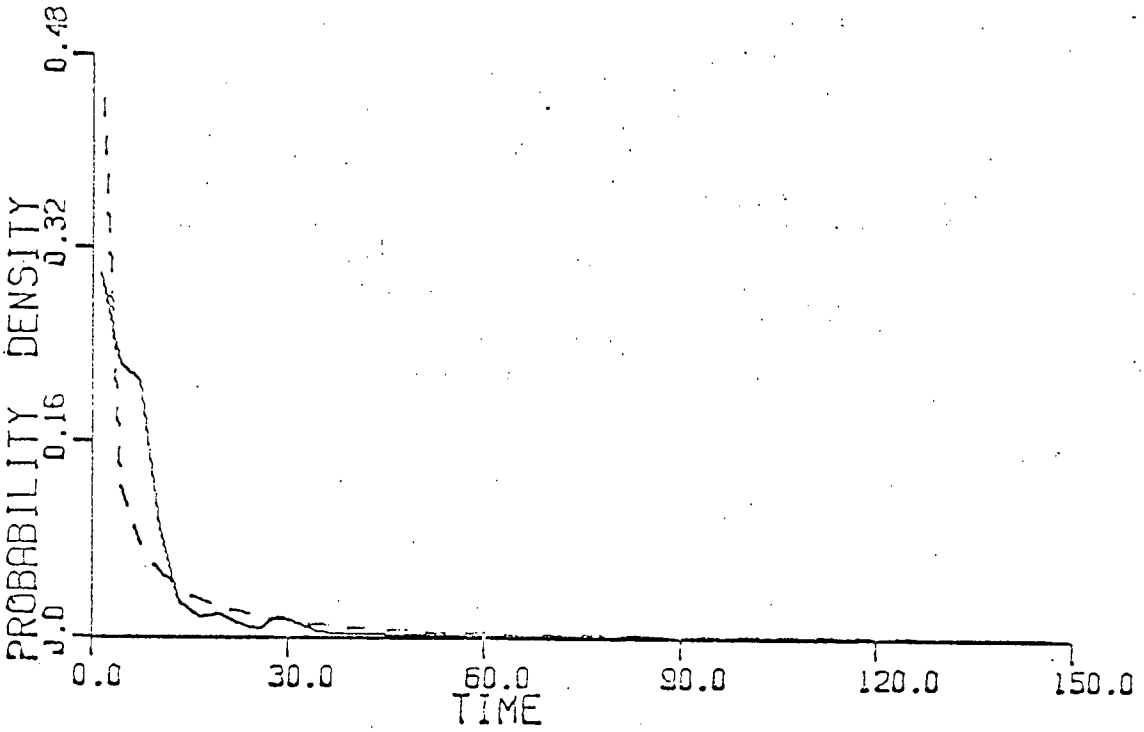


Figure 2.2 Requested CPU intervals (batch)

made the mean and standard deviation large. Such large samples are very rare.

Scherr [43] found a similar shaped distribution for these times, with a mean of 0.88 seconds for jobs run on an IBM 7094 running CTSS.

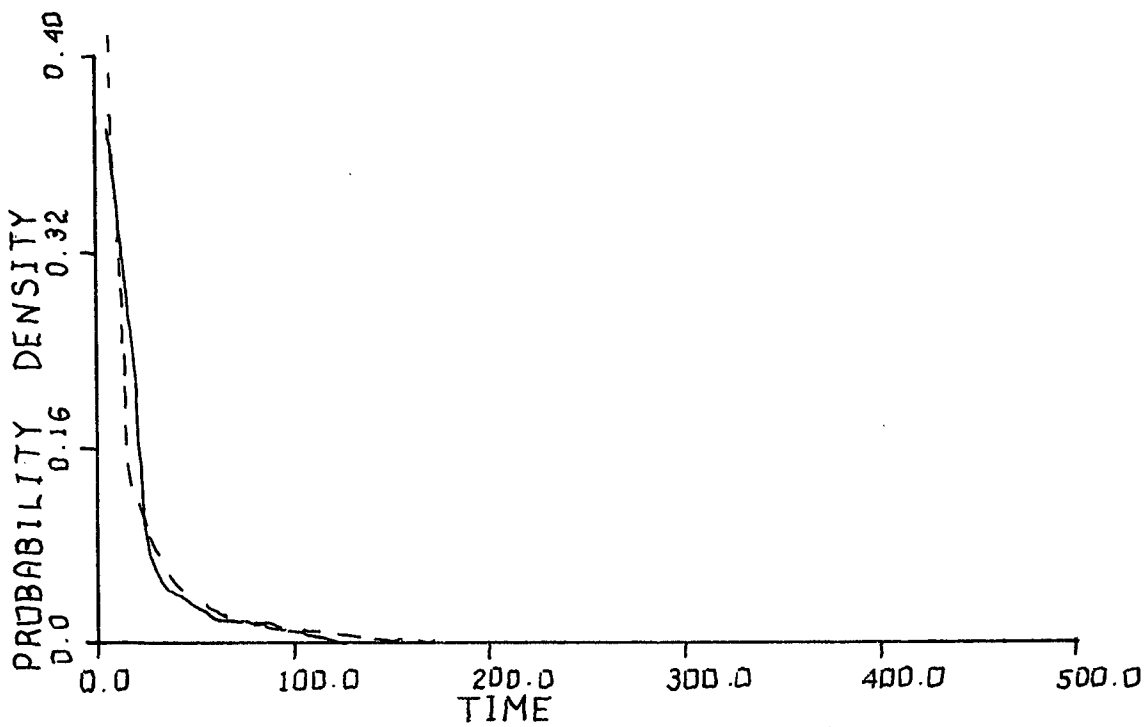


Figure 2.3 Processor time between interactions

2.2 CPU Service Obtained

The amount of time that a job actually gets on a processor in one shot, called an *actual CPU interval*, is considerably less than it requests, since the maximum time that a job may hold a CPU is limited by its time slice, and a job has a high probability of being pre-empted if there are more than a few other jobs in the system. Means for the CPU service obtained (actual CPU intervals were 14.3 ms for jobs on tape 1, and 8.9 ms for jobs on tape 2 with standard deviations of 42.5 and 14.9 respectively. The average actual CPU intervals on tape 1 represent about 80% of the requested intervals, while those on tape 2 represent only about 45%. The reason for this difference is that tape 2 records activity at a busier time than tape 1, and jobs are pre-empted more frequently when the system is busy.

A hyper-exponential distribution provides the closest fit for these values, and the observed data show a maximum deviation of 0.0900 (KS(.01)=0.0111) from the fitting curve for tape 1, and a maximum deviation of 0.027 for tape 2. (See figs. 2.4 and 2.5).

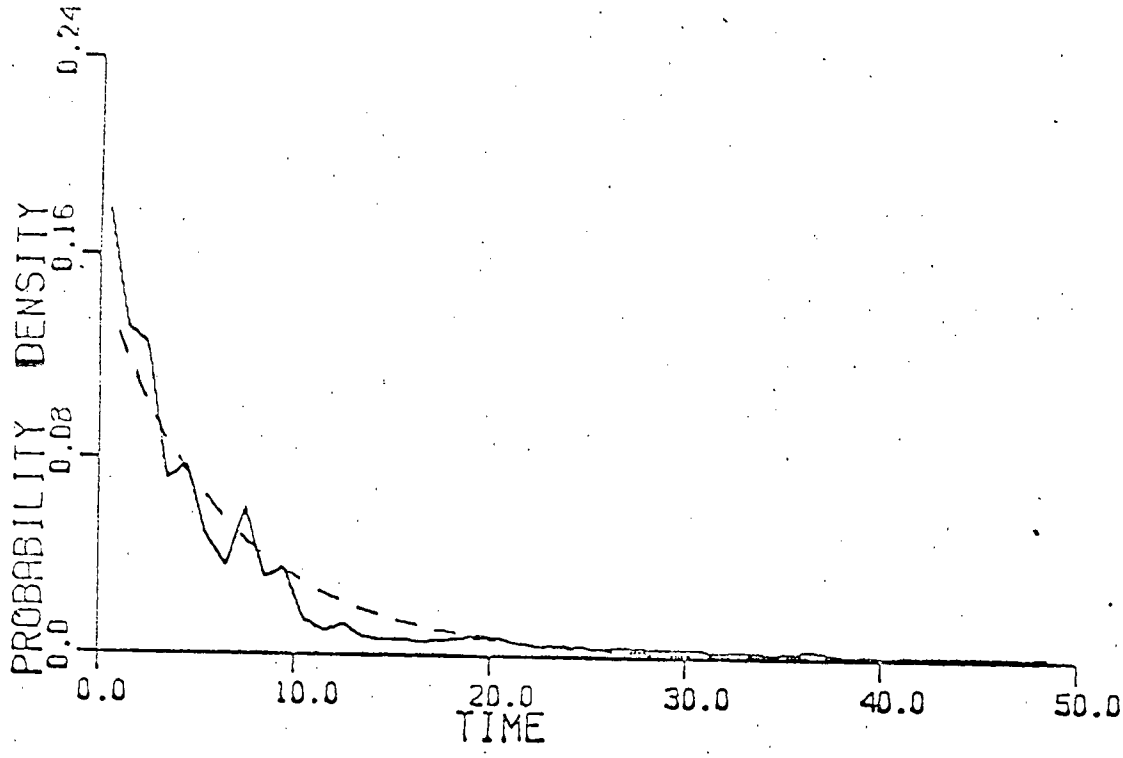


Figure 2.4 Actual CPU intervals (tape 1)

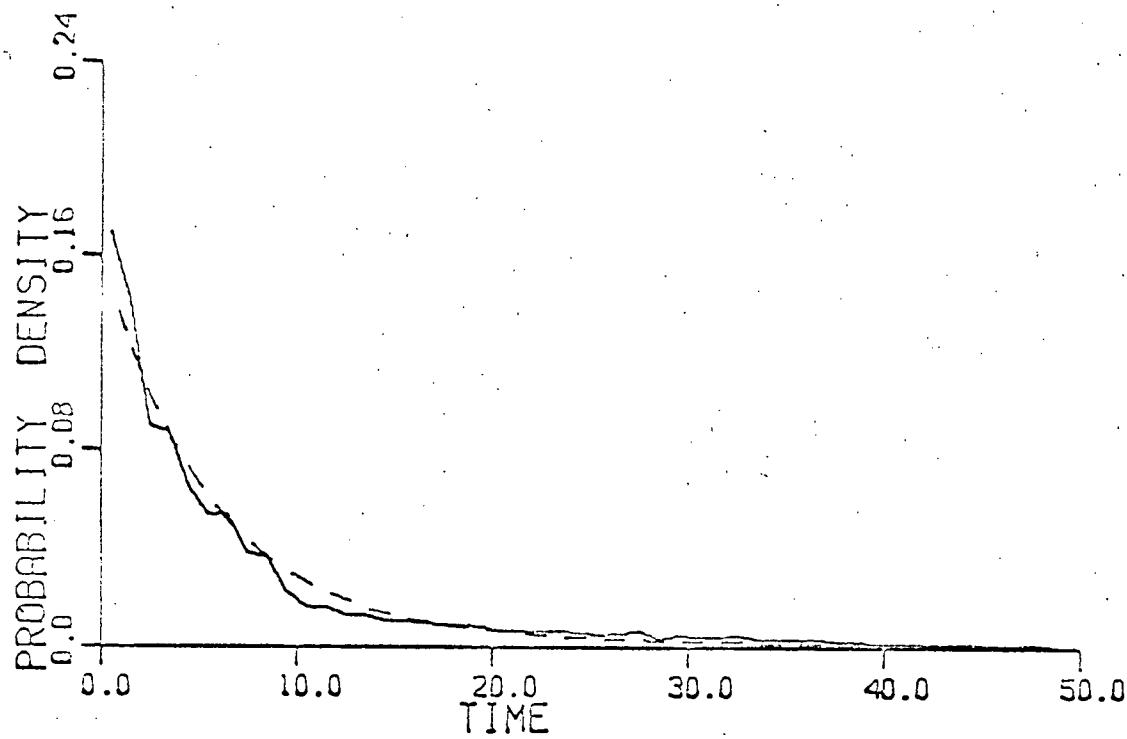


Figure 2.5 Actual CPU intervals (tape 2)

2.3 I/O Delays

In MTS, the processing of a job is not overlapped with its own I/O, and as a result, the job must remain idle during these times.

Most of the I/O activity of MTS jobs is for disk operations, with some for tape I/O. For interactive jobs, some of the I/O is also for the terminal on which the job is run. The distributions of I/O delays for both interactive and batch tasks (excluding the delays for the terminal itself) are quite similar, (figs. 2.6, and 2.7) and also ressemble closely the distribution for disk I/O delays measured by Baskett et al [5] for the University of Texas' CDC 6600 system. The peaks are close to the same point (35 ms), and the shapes of the distributions are similar. The reason for the difference in means for the batch and interactive tasks is probably due to the fact that much more disk arm movement is required for the interactive tasks, since fairly long periods can be spent waiting for terminal responses during which time the arm can be repositioned by another task.

The peak observed in fig. 2.3 at 80 ms is due to several jobs in the sample doing tape I/O. 80ms is the time taken to transfer a tape record one page in length, and this is a very common size used in the MTS system. Most disk operations take between 25 and 75 ms (from Pinkerton [39]), with those requiring arm movements taking longer.

Although these distributions look Erlangian, the standard

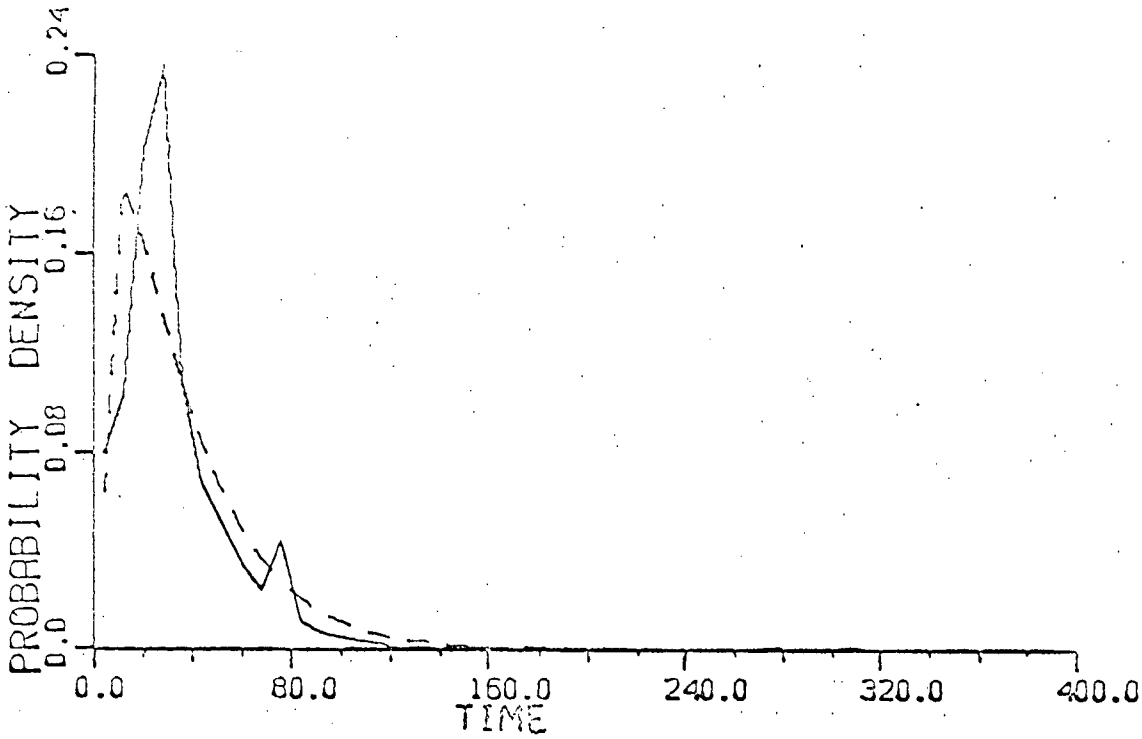


Figure 2.6 I/O delays (batch)

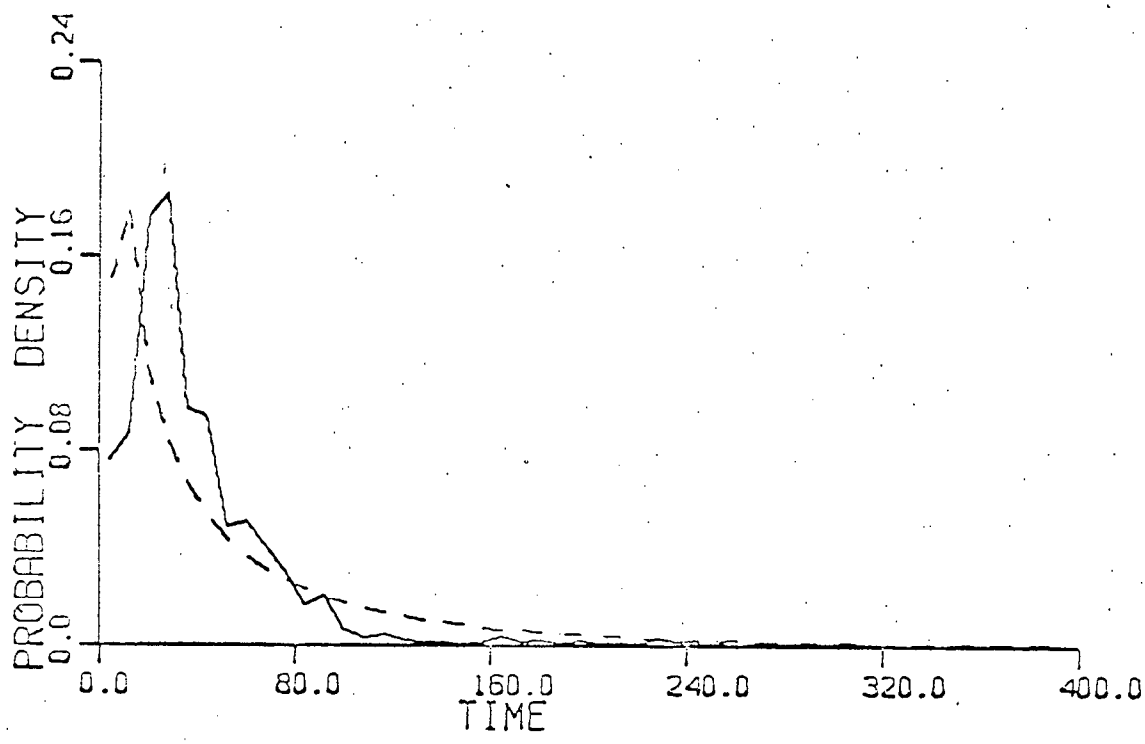


Figure 2.7 I/O delays (interactive)

deviations are greater than the means. For batch tasks, the observed mean was 38 ms with a standard deviation of 67, and for interactive tasks, the mean was 123 ms with a standard deviation of 986. The dotted lines in figs. 2.6 and 2.7 show the Weibull density function giving the closest fit according to the Kolmogorov-Smirnov goodness of fit test. Greater values of (causing the peak to be shifted right) yielded larger deviations from the cummulative distribution than the curves plotted.

The term "User Response Time" refers to the time that it takes a user at an interactive terminal to frame his response and to type it in. It is measured from the time that the system starts waiting for some action from the user, until the user enters a carriage return or other end of line signal indicating that he has finished his response. Scherr [43] calls these times "Think Times". The distribution of user response times for interactive tasks will vary from system to system, depending on the type of response required, the devices used, and the sophistication of the users.

Scherr found the mean user response time to be 35 seconds for users of the CTSS sytem, but for MTS users the mean is closer to 11 seconds with a standard deviation of 16. The distribution of these times was similar in shape to that observed by Scherr, if one removes the program generated responses from his observations. It is also similar to the distribution observed by Schwetman and DeLine [44] for the RESPOND system.

A Weibull distribution again showed the closest fit (fig.

2.8) and the observed data displayed a maximum deviation of 0.0655 from it with KS(.01)=0.05. The distribution in the figure may be seen to have two large peaks. The first corresponds to quick responses requiring little thought on the part of the user, and also little typing. Such responses are quite common in MTS with the user entering a null line, or one with only a few characters in reply to some query on the part of the system. The second peak represents those responses requiring some thought, or considerable typing, or both. The number of responses falling into the first category (less than 5 seconds) is roughly 35% of the total.

The system responds by typing a reply which takes an average of 0.65 seconds (fig. 2.9) and writes about five times as many lines as the user inputs. The peaks in the distribution correspond to different devices.

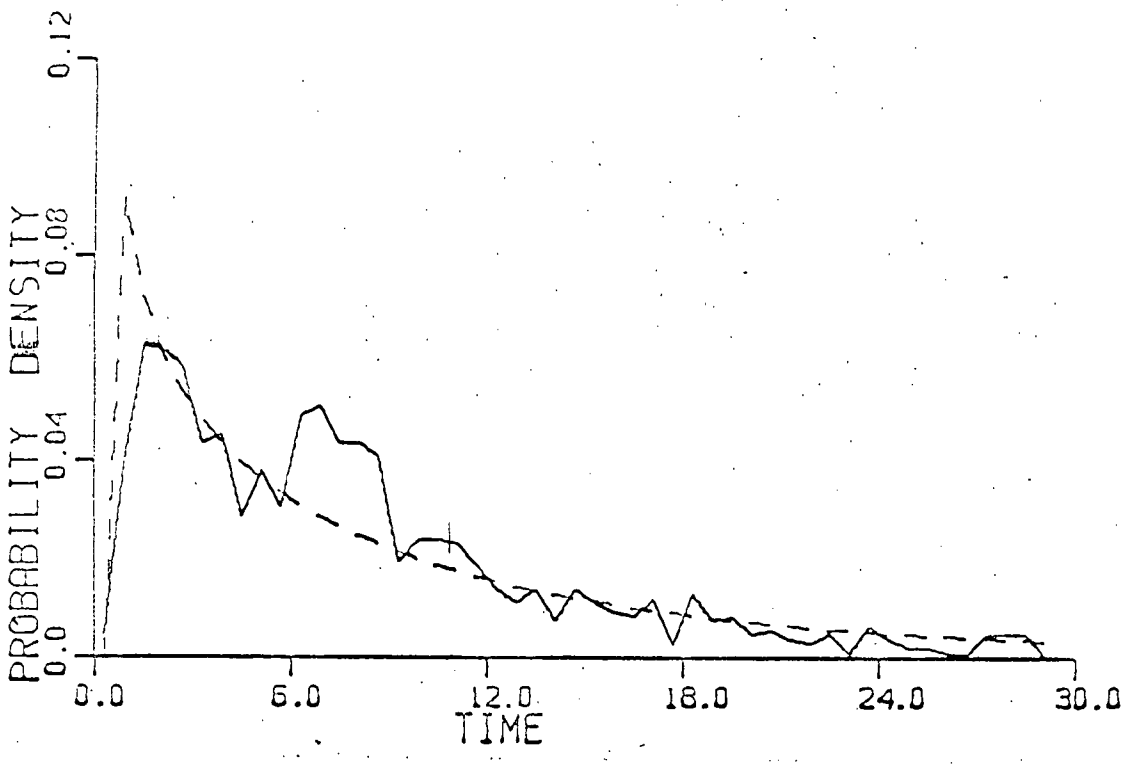


Figure 2.8 User response times

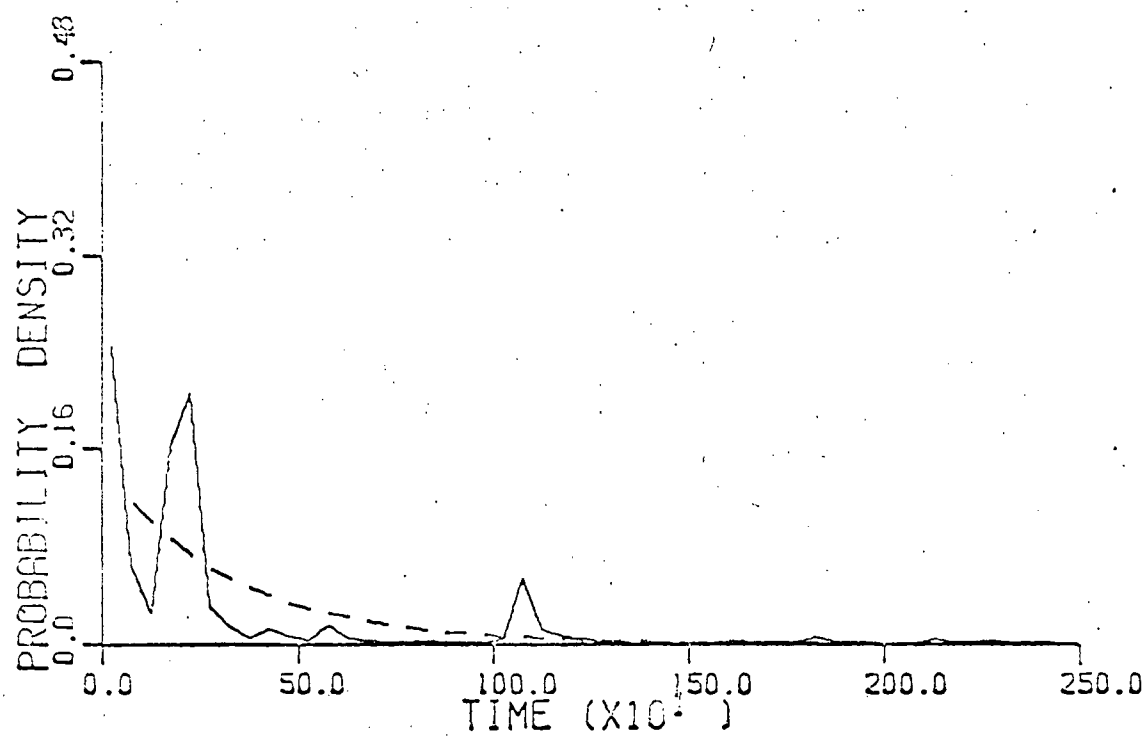


Figure 2.9 System write times (interactive)

2.4 Page Waiting Times

The distribution of page waiting times is divided into several types of events in MTS, depending on the reason for the page fault that generated the request, and the status of the page involved. The possibilities are outlined below:

1) Page fault was for the first reference to a new page, which can be allocated nearly immediately.

2) Page fault was for the first reference to a new page, but core space can be allocated only after first writing out another page to provide space.

3) Page fault was for a page existing on the drum, which must be brought into core.

4) Page fault was for a page existing on the drum, but it can only be brought into core after first writing out another page.

5) Page fault was for a page that still exists in core, although a page out operation was in progress for that page. The page out operation can be cancelled, and the page will become available immediately.

Since MTS tries to keep about 20% of the available core space empty to allow quick service to page reads, situations of

type 2, or 4 above occur only rarely. In addition to the five types of events above, the supervisor may decide to make a task non-privileged (see sec. 1.2.3) if it attempts to obtain more than a certain threshold of real pages. The job must then wait for some other privileged job to leave that status before it can be made privileged, and have its page request serviced. This has the effect of making some page waits very long. Events of this type are easily separated from the rest by the large difference in magnitude.

Because of the various types of events making up the distribution of page waiting times, no one function can be made to fit the observed data very closely. If one breaks the distribution into part corresponding to the events of different types, however, a reasonable fit can be obtained. (See figs. 2.10, 2.11, and fig. 2.12). In this case, the distribution was broken into two parts, the first corresponding to events of types 1, and 5 above, and the second to events of type 3.

An exponential distribution with a mean of 1ms was used to fit the distribution of events of types 1, and 5, and an Erlang distribution with a mean of 22 ms and 6 channels was used to fit the events of type 3. This gives a fair fit to the observed data, which shows a maximum deviation of 0.1453 with KS(.01)=0.0591. The probability density function for the resulting distribution is seen in fig. 2.12.

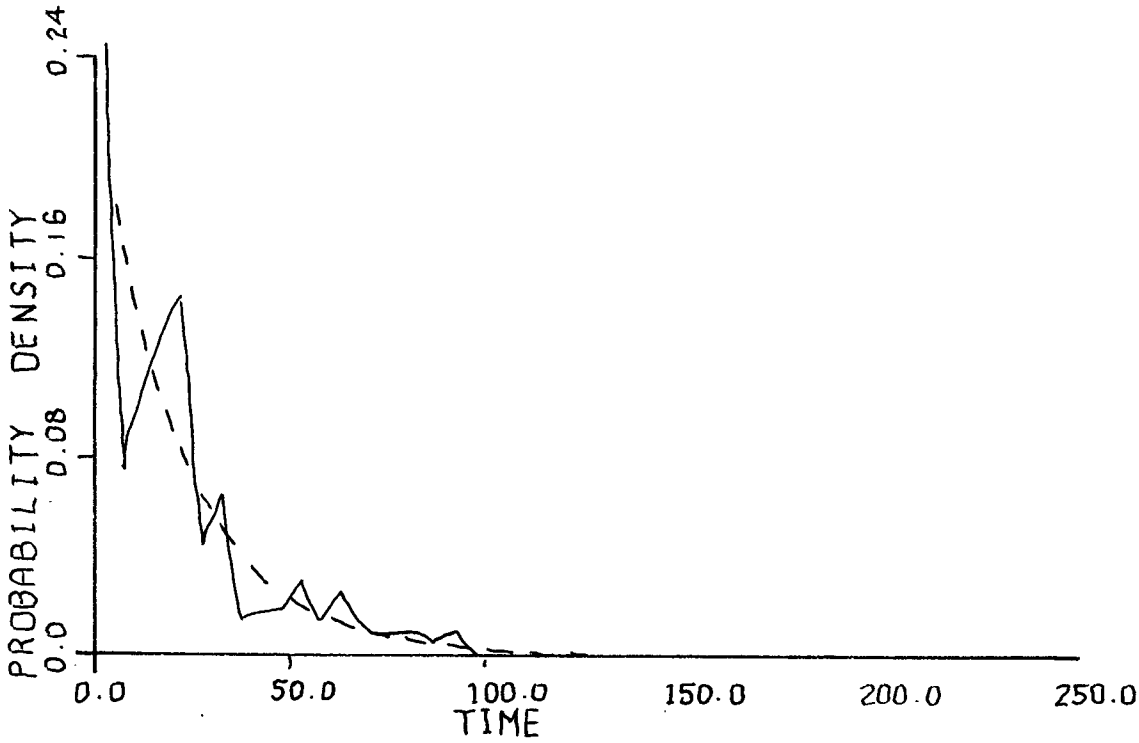


Figure 2.10 Page waiting times (tape 1)

2.5 Other Parameters Of Interest

2.5.1 System Response Time

System response time is defined as the time that it takes the system to start to type a response, after a user has entered his carriage return signal at an interactive terminal. This quantity is of interest, as it is often used as a criterion for judging the performance of time-sharing systems.

The distribution observed had a mean of 0.42 seconds, with a standard deviation of 2. (see fig. 2.13) The median was only

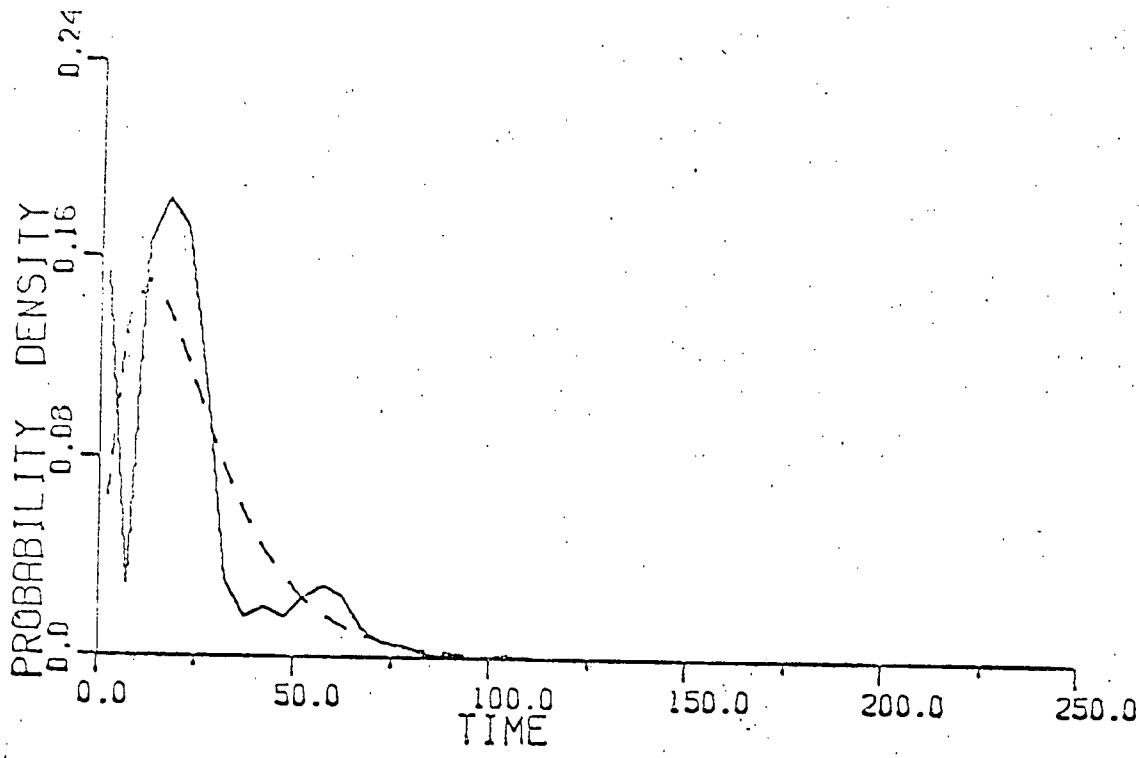


Figure 2.11 Page waiting times (tape 2)

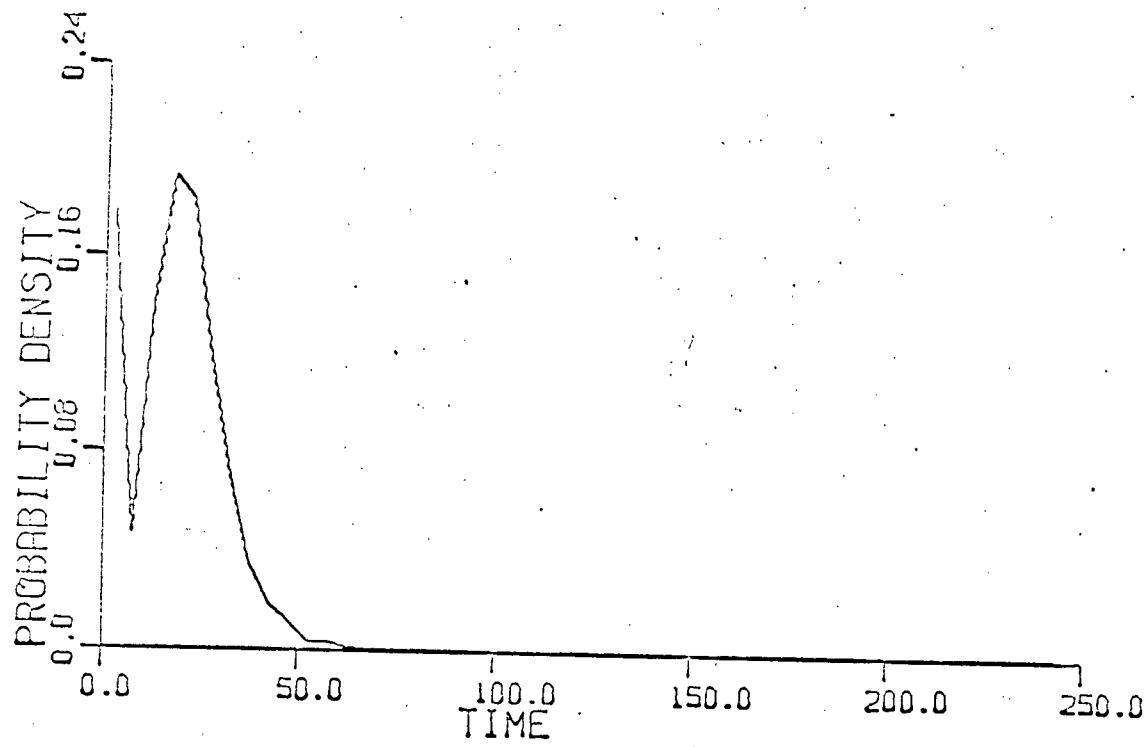


Figure 2.12 Page waiting times generated by model

30 ms. A Weibull distribution gave the closest fit, and the observed values showed a maximum deviation of 0.070 from it, with KS(.01)=0.05. The distribution is similar in shape to the one obtained by Scherr [43], although the mean is an order of magnitude less. It is quite different in shape to the one observed by Schwetman and DeLine [44] for the RESPOND system, although the mean time is similar in this instance. The RESPOND system seems to make almost no very quick responses.

Looking at the distribution of processor time required between interactions, one would expect very quick responses since so little time is used in most cases (less than 55 ms), and the mean time for disk operations if they are required is small (less than 100 ms).

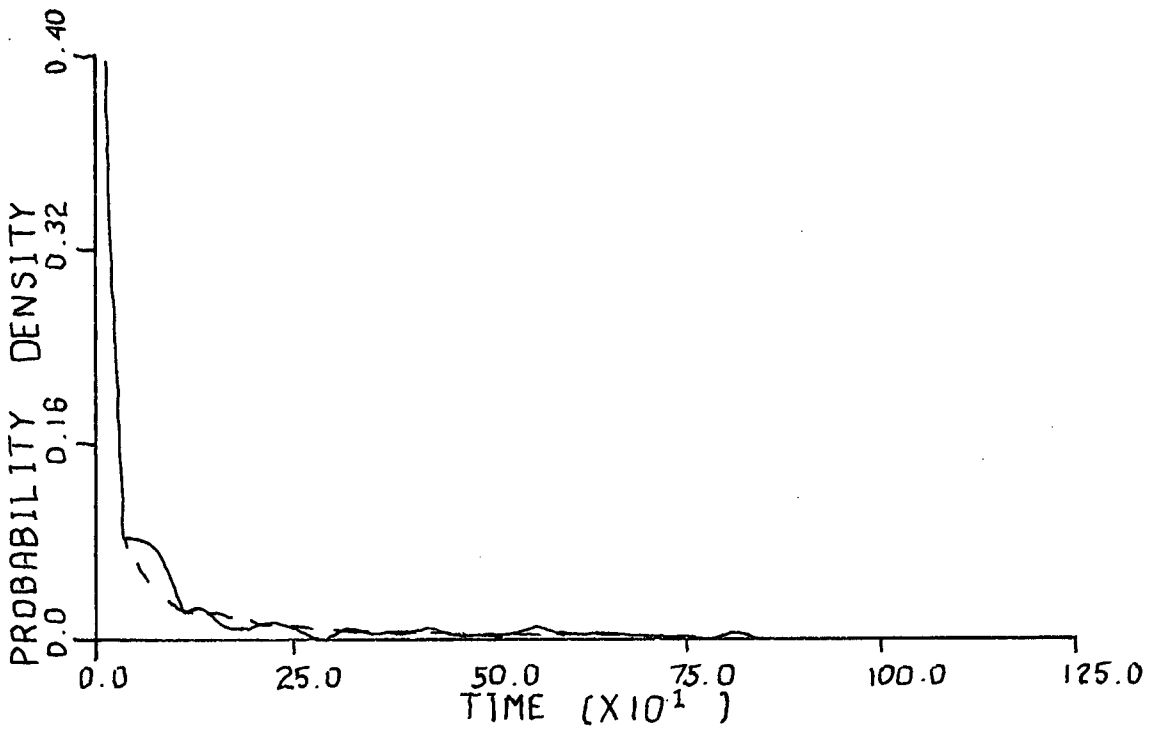


Figure 2.13 System reponse time

2.5.2 Ready Intervals

Ready intervals are defined as the intervals of time that tasks which could be running on a CPU spend waiting on the processor queue. These include tasks which have used up their time-slices and are placed on the bottom of the queue, and tasks which have been pre-empted and are placed on the top. The mean value for ready intervals is very small since a large portion of the intervals represent pre-empted tasks waiting for some interrupt to be processed, and so will obtain CPU service again in a relatively short time, since most interrupts are processed in 200 - 300 us. It is also unlikely (at UBC) that many tasks will be stacked in such a manner that each is waiting for the previous task to finish processing some interrupt. Processing of interrupts has been removed from the distribution of actual CPU intervals, so the mean of the actual intervals is larger than it would be if these were included. It is for this reason that the mean of the ready intervals is so small compared to the mean of the actual CPU intervals. (See figs. 2.14 and 2.15 for distributions of ready intervals).

2.5.3 Inter-Arrival Times

The inter-arrival times of jobs to the system is the time between successive "SIGNON's", not the time between submissions of jobs to the HASP queue. For terminal jobs, the act of signing on puts the job immediately into the system, but for batch jobs the case is different. Batch jobs are enqueued by HASP, and ordered according to an assigned priority which is

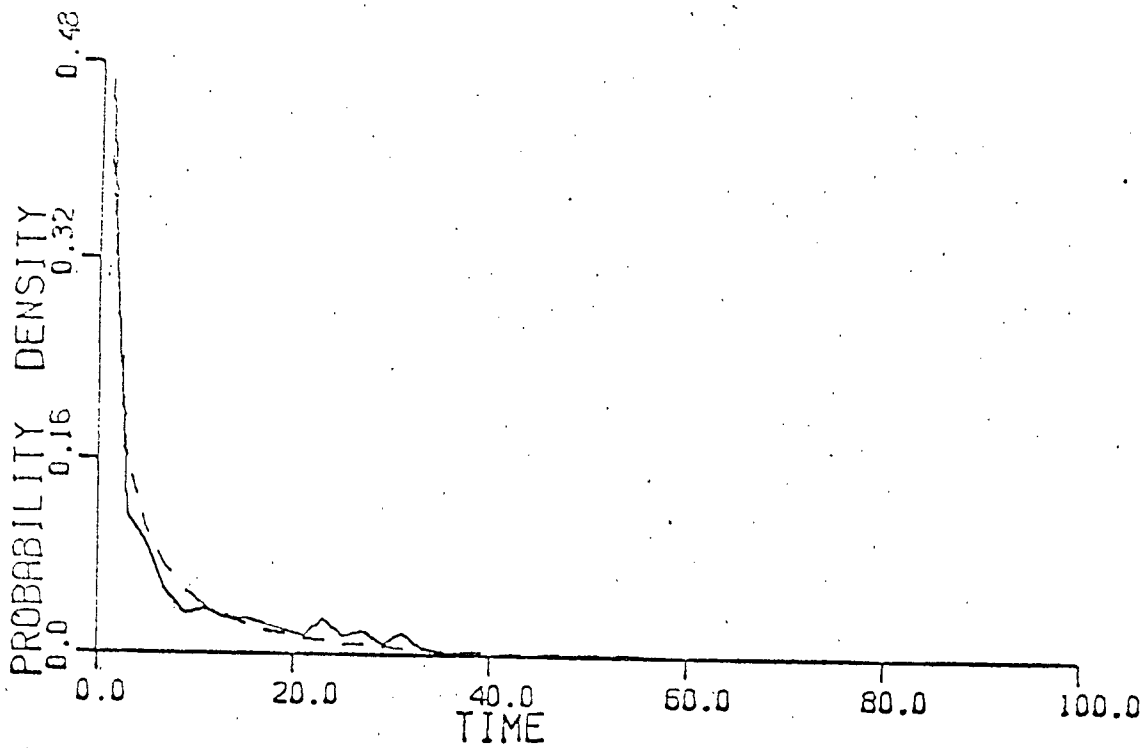


Figure 2.14 Ready intervals (tape 1)

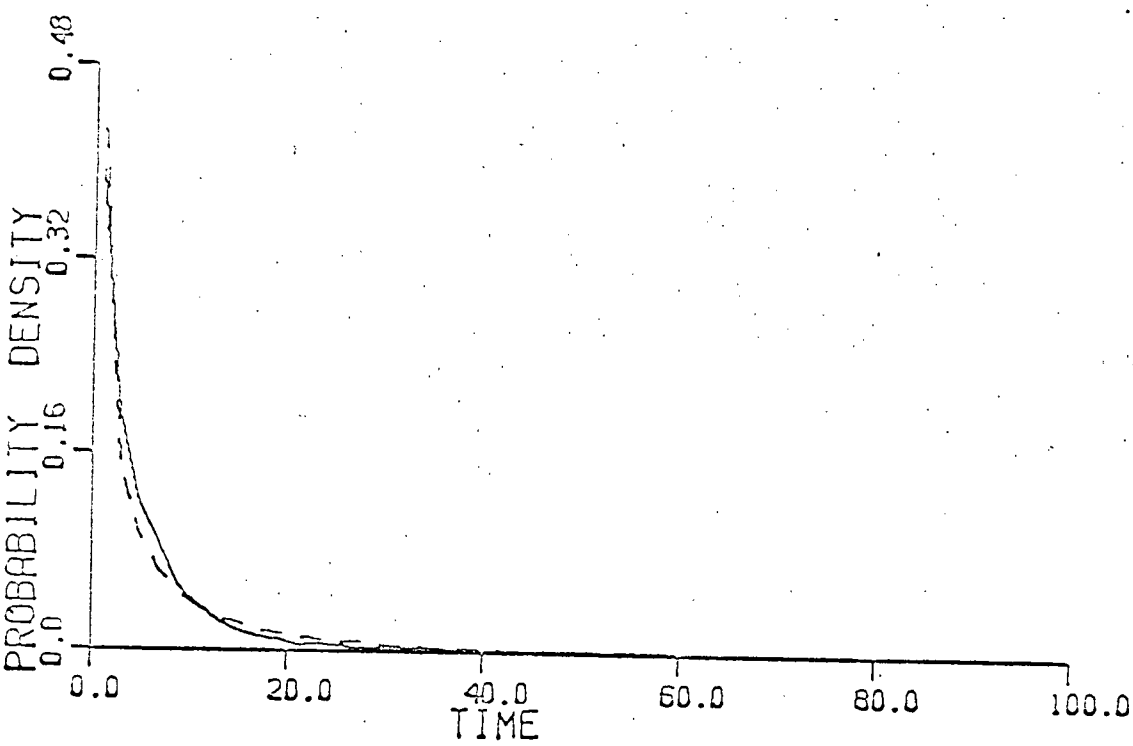


Figure 2.15 Ready intervals (tape 2)

calculated from the user's estimates of CPU time, and pages of output required for the job. Jobs are released from the queue for execution at a rate that is dependent on the system load. Only two batch jobs are permitted if there are many terminal users signed on. The time that a batch job actually leaves the queue and starts executing is deemed its arrival time.

The inter-arrival times for terminal jobs were found to be exponentially distributed as expected, and the same was found to be true for batch jobs. The mean inter-arrival time for batch jobs was 40 seconds, and was 94 seconds for the terminal jobs. More accurate values for inter-arrival times could be obtained by examining the system accounting records.

2.5.4 Total Processor Time

Processing time required by jobs is the total amount of CPU time required to complete the job. This quantity was found to be hyper-exponentially distributed with a mean of 16 seconds, and a standard deviation of 28, based on a sample of 136 jobs drawn from the data tapes. Again more accurate values could be obtained by examining the accounting statistics since a much larger sample could be used.

	Tape 1		Tape 2	
	Terminal	Batch	Terminal	Batch
Requested CPU Intervals	18.1 ms (87.6)	24.1 ms (319)	14.9 ms (48.4)	23.2 ms (213)
Actual CPU Intervals	12.9 ms (37.6)	19.4 ms (60.1)	8.0 ms (14.1)	10.1 ms (16.1)
I/O Delays	150.2 ms (1203)	30.1 ms (31.9)	76.1 ms (381)	42.2 ms (82.3)
User Response	10.4 sec (14.9)		12.1 sec (17.7)	
System Write	0.64 sec (1.90)		0.57 sec (1.67)	
System Response	0.39 sec (2.01)		0.47 sec (2.14)	
CPU Time Between Interactions	59.9 ms (719)		55.8 ms (535)	
Page Waits	22.5 ms (22.0)	22.2 ms (22.4)	25.7 ms (20.7)	24.1 ms (15.4)
Ready Intervals	3.7 ms (11.4)	3.9 ms (14.8)	6.4 ms (15.4)	7.0 ms (17.5)
No. Concurrent Tasks	~7	1~2	~19	2~3
Inter-arrival Times			94 sec (91)	40 sec (41)

Bracketted numbers are standard deviations.

Table 2.1 Job stream characteristics

CHAPTER 3 STORAGE REQUIREMENTS

3.1 Preamble

In a time-sharing environment, there is usually insufficient core storage available to allow all users to have their full allotment of storage in core simultaneously. Schemes of virtual memory management have been developed [1,3,15,17,18] which try to give each user only as much core as he needs at the moment, while the remainder of his allotted memory space resides on some secondary storage device such as a drum. The memory is normally grouped into blocks called pages, which are the units of storage transferred between core storage and the drum. A page is brought into core memory when a reference is made to it, and pages that are in core for a long time, or have not been referenced in a long time, are placed on the drum to make more core storage available.

Even with the large amounts of virtual storage that exist in such systems, each user cannot simply be allocated as much storage space as he might possibly require, and be allowed to keep all this space for the duration of his time on the system. In this instance, most of the storage would be unused at any moment and poor utilization of the hardware would result. In addition, a restriction would have to be placed on the number of simultaneous users which would depend on the total amount of storage available.

To overcome this problem, methods of dynamic storage allocation have been devised that allow jobs to acquire storage as they need it, and to release that storage as soon as the need disappears. This process is transparent to the user in most

cases, so that when he runs a program, the correct amount of storage is automatically acquired, and then released when the program terminates. Most systems provide the user with the capability of writing programs that acquire storage during execution (see [25]), either to load additional program modules, or to increase the size of data areas, and this storage may be retained or released as the program writer sees fit until the program terminates.

Since the storage requirement of jobs is a dynamic quantity, any simulation which studies the utilization of virtual memory, must incorporate a model for the storage demands of jobs. Most studies, however, have concerned themselves with paging behavior (such as times between inter-page references, and sequences of references, etc. See [1,13,22]), and real core requirements for efficient running [8,16,29], rather than the variability of storage requirements with time, since the concern was mostly the feasibility of paging algorithms, and paging machines in general. Lehman and Rosenfeld [31] broke their model of a job into job steps, giving each step a different storage allotment. This comes closest to modelling the variable nature of storage requirements. Denning [16] developed a model for the real core demands of computer jobs, which he calls the working set model.

3.2 Observed Program Storage Requirements

A storage profile is a plot of the storage requirements of

a job over its running time (See fig. 3.1, and Appendix D for examples). A number of these profiles were collected, and descriptive parameters for each were measured. The time scale was normalized to make all the profiles the same length for comparison purposes. The parameters measured were the maximum number of pages, the mean number of pages, the number of distinct humps (a hump is an area of the profile where the number of pages possessed exceeds an arbitrary small limit, 15 pages in this case), the width of the largest hump, the number of changes in storage requirement, the average size of these changes as a percent of the maximum number of pages, and the size of the maximum increase and decrease in storage requirement as a percent of the maximum number of pages.

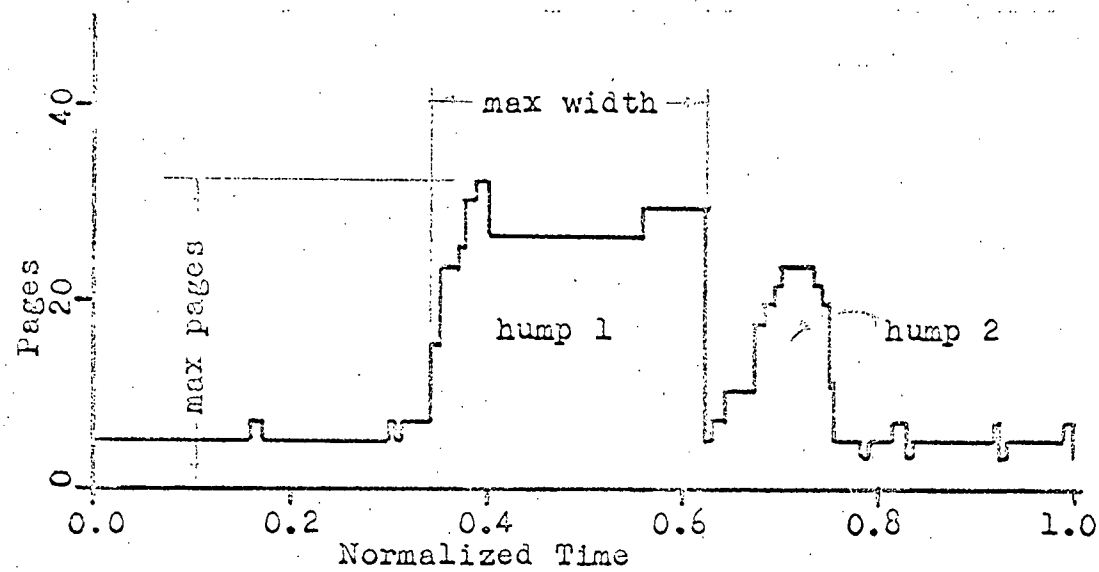


Figure 3.1 - Typical storage profile with measured parameters.

Using the parameters measured for each profile as a vector defining the position of the profile in an eight dimensional Euclidian space (eight parameters), and doing some cluster

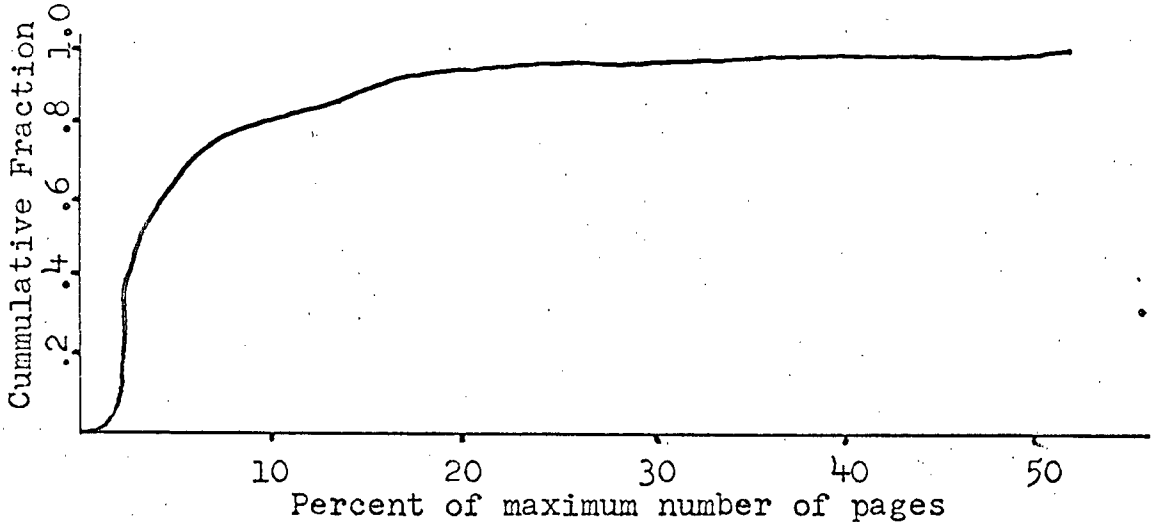
analysis [45] the profiles were found to fall into two clusters.

Profiles from the first group showed large maximum storage requirements, and usually had one large hump covering most of the running time. The average maximum number of pages for this group was 70 pages, and the average size of change in storage requirement was 6.5 pages. (See figs. D.1 through D.8.) Profiles from the second group had an average maximum number of pages of 23, and the average size of change was 4.1 pages. Most of the profiles in this group show a great amount of fluctuation in the storage possessed (see figs. D.9 through D.20).

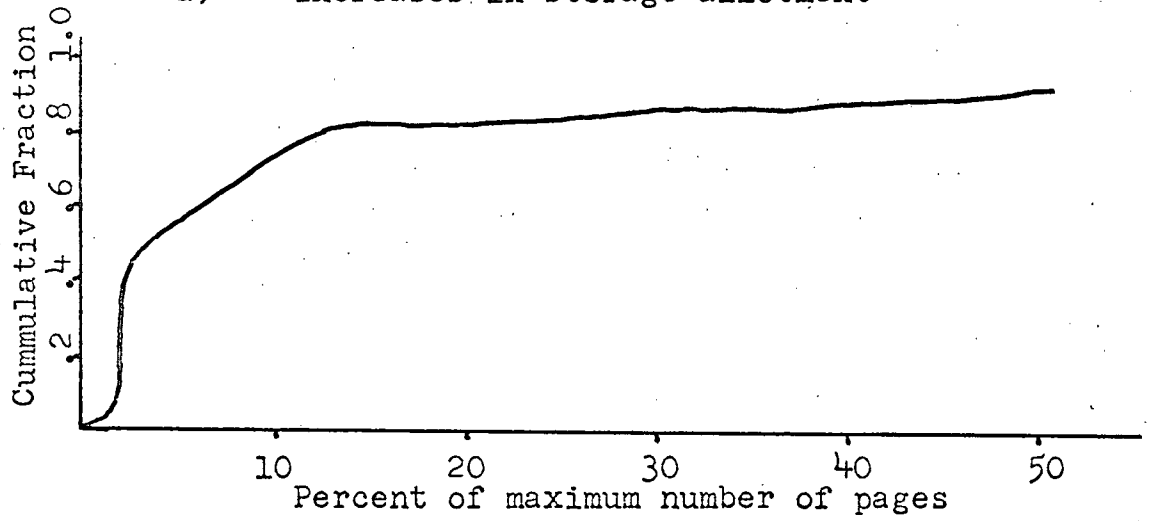
About 50% of the batch jobs examined had storage profiles falling into the first group, while only 25% of the interactive jobs had similar requirements. At the University of British Columbia, the MTS user is charged for storage used during inactive periods when he is signed on at an interactive terminal, but not if his job is run from batch. This makes programs with large storage requirements costly to run interactively, and probably accounts for the lower proportion of interactive jobs having profiles in the first group.

3.3 A Model For Program Storage Requirements

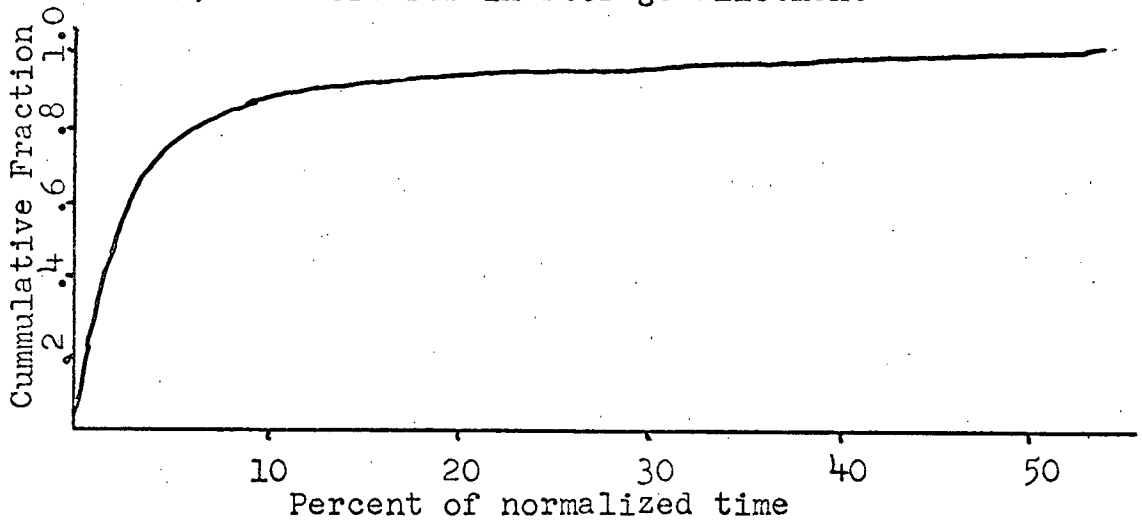
Since it would be desirable to generate storage requirement profiles for computer jobs for simulation purposes, a model was developed to do this. The model used is very simple in principle, but seems to generate profiles quite similar to the ones observed (see Appendix D, and Table 3.1).



a) Increases in storage allotment

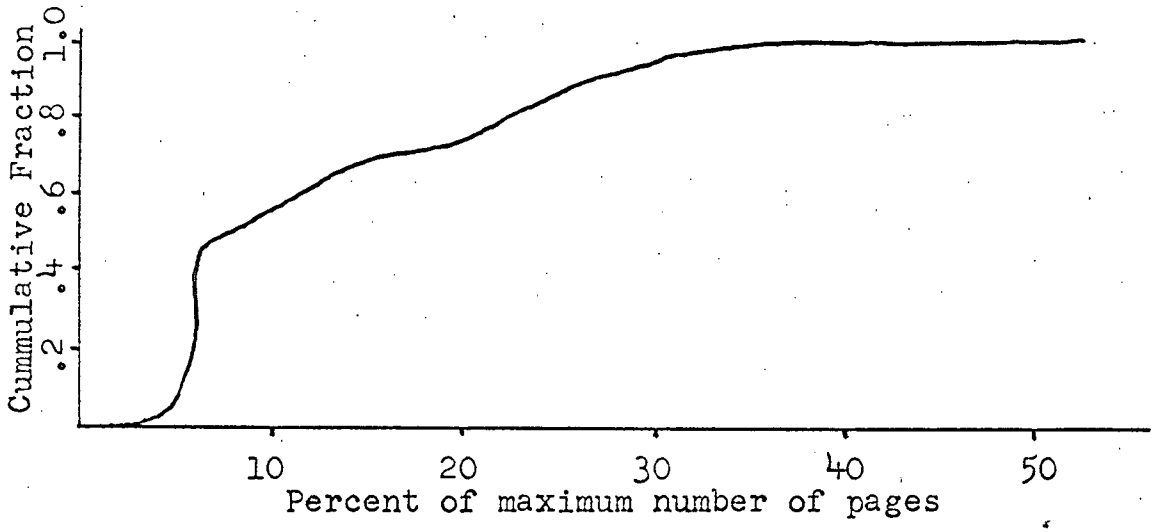


b) Decreases in storage allotment

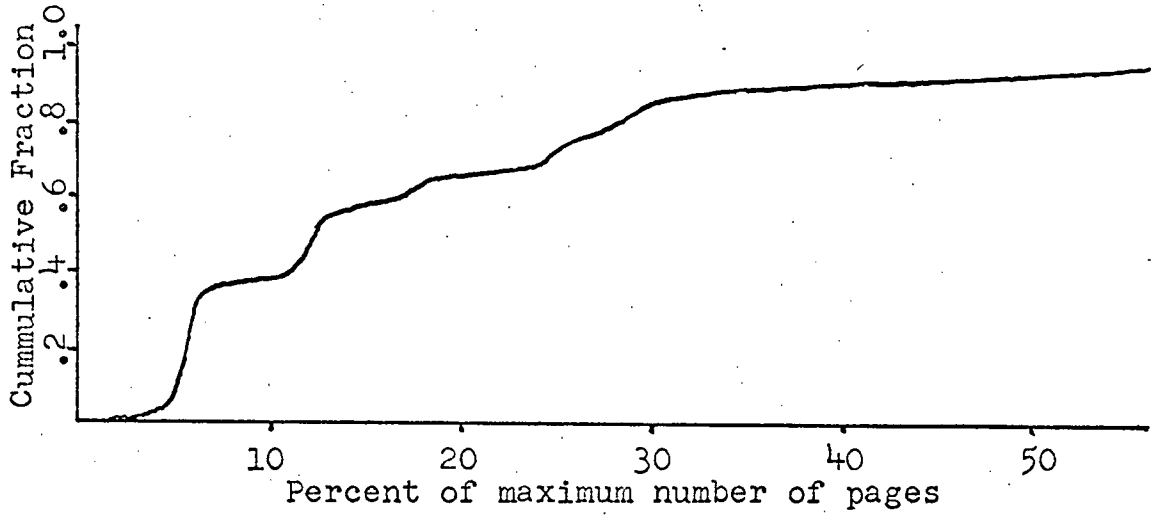


c) Time storage allotment remained static

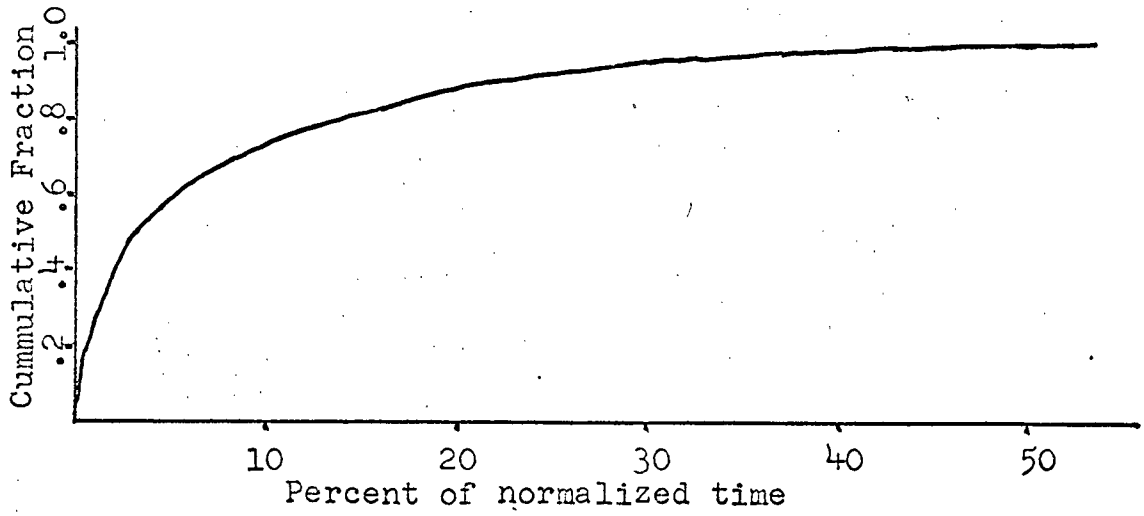
Figure 3.2 Distributions for storage profile model (grp 1)



a) Increases in storage allotment



b) Decreases in storage allotment



c) Time storage allotment remained static

Figure 3.3 Distributions for storage profile model (grp 2)

A set of distributions was determined for the sizes of increases and decreases in the amount of storage possessed for each category (see figs. 3.2 and 3.3), and also a distribution for the lengths of time that the number of pages remained static. Conditional probabilities for the direction of the next change in the number of pages (up or down), given the direction of the last change, were measured from the observed storage profiles. By drawing random numbers from the above distributions, and forcing an increase to the maximum storage size at some point, it is possible to generate very reasonable storage profiles. A comparison is found in Table 3.1

It is possible that a much more elaborate model could be devised, but because of the extreme variability of jobs run on this system, it is unlikely that much better profiles could be obtained.

A flowchart describing the model appears in figure 3.4.

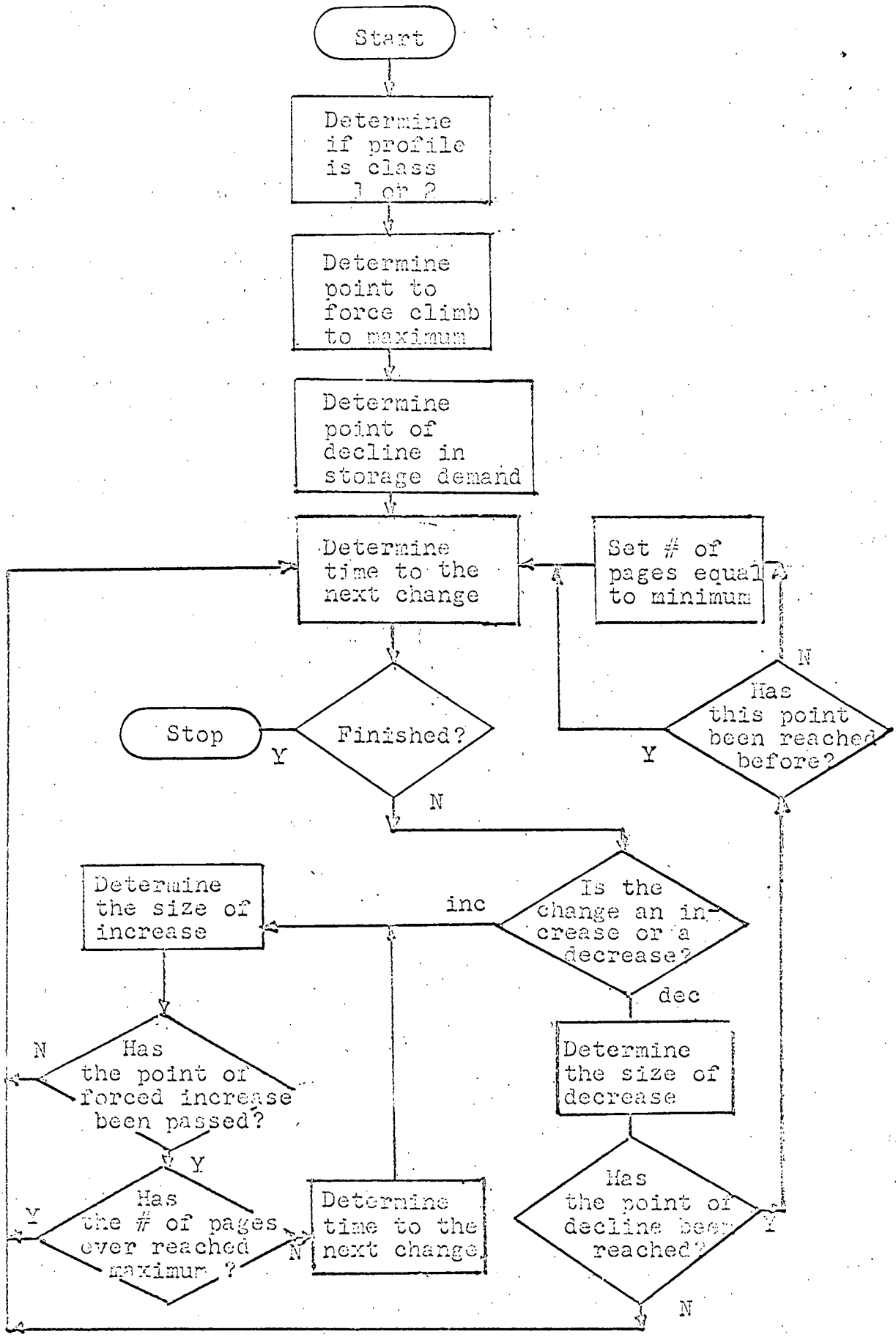


Figure 3.4 Flowchart of storage profile model

	Maximum # pages	Mean # pages	# humps x 10	Max % increase	Max % decrease	# of changes	Maximum width	Mean % change
Group 1 Observed	23.17	10.59	14.17	28.95	49.92	21.00	17.42	17.61
Group 2 Observed	70.43	42.84	11.43	35.55	55.73	25.14	74.00	9.27
Group 1 Generated	23.17	11.72	15.00	32.31	61.66	17.09	21.50	18.66
Group 2 Generated	70.43	34.42	10.91	36.22	57.25	26.88	69.31	10.73

Table 3.1 Comparison of observed profiles, and those generated by the model.

CHAPTER 4 SUMMARY AND DISCUSSION

4.1 Summary And Discussion

It was found in section 2.1 that the requested CPU intervals for batch jobs were about 30% longer on the average than those for interactive jobs, and that these intervals have standard deviations about four times as large as their means. Either a hyper-exponential or a Weibull distribution can be used as a model for the intervals, with the hyper-exponential distribution being the easiest to use, as it is much easier to generate parameters that give the desired mean and variance. The relationship between the distribution parameters, and the mean and variance for the Weibull distribution is very complex, (see Appendix B) and no straight forward solution exists to determine the parameters from them.

The processor time required by interactive tasks, is broken up into requested CPU intervals, with disk or tape I/O operations terminating all but the last of these intervals, which is terminated by a terminal I/O operation. The Weibull distribution gives the closest fit according to the Kolmogorov-Smirnov goodness of fit test, has a mean and standard deviation much less than those observed due to a few very large values in the observed distribution.

A shifted Weibull distribution (one with the γ parameter greater than zero) provides a fairly good fit to the observed distribution of user response times, but for the distribution of times that it takes the system to write lines no distribution function was found to provide a reasonable fit. It would probably be a good idea to break this distribution up according

to the various types of devices involved, and to determine a distribution of write times for each one.

A Weibull distribution with a shift can be used to model disk I/O delays, but the fit obtained is not a good one.

The distribution of page waiting times for the MTS system, is best broken up into two categories if one wants to generate page transfer delays with a distribution similar to the ones observed. An exponential distribution models the delays that do not require a transfer of a page from the drum, and the traditional Erlang distribution can be used to model those delays that do.

Distributions of system response time, and actual CPU intervals can be used to check the accuracy of simulation models tested.

In the area of program storage requirements, more work might be profitable. For simulation models, the relationship between virtual storage allotment and the amount of real core needed would be useful, as well as relationships between amount of real core, the number of simultaneous users, and the amount of paging done. The model developed in section 3.3 for virtual storage requirements of jobs is only a beginning in this direction.

4.2 A Simple Model Of MTS For Simulation

The following simple simulation model of MTS could be used to employ some of the measures taken in this study. The model

keeps three lists outlined below:

1) Job List

Each entry on the job list contains all the information concerning a particular job running on the simulated system, and includes these fields:

TC	maximum CPU time for the job
TE	current total CPU time for the job
TR	time remaining in the current request for CPU service
TRTS	time remaining in the time-slice
NB	the reason the job will stop at the end of the current requested CPU interval

Various other fields can be used to keep track of statistics of interest such as the current number of real pages, the cumulative wait time, etc.

2) CPU Queue

This list is linked with the first entry pointing to the next, etc. Each entry contains only a pointer to the job list, and the link to the next entry, with the position of the entry indicating the position of the corresponding job in the simulated CPU queue.

3) Event List

This list is also linked, and ordered according to the times of the events represented. Events in this list will cause the pre-emption of a job that is running on a CPU if any job is running at the time of the event. Each entry contains a pointer to the job list indicating the job associated with the event, an indication of the event type, and a link to the next entry in the list.

In addition the model keeps the current time T.

A flowchart of the model appears in fig. 4.1. The value i indicates the job currently in possession of a CPU. (note: the model is shown for only one CPU since others are identical) Tr is the running time before stopping for the job currently on the CPU, and Tn is the time of the next event from the event list. Tev is the time calculated for an event to occur.

i	pointer to job list (current job)
T	current time
Tr	running time before stopping for job i
Tn	time of next event
Tev	time calculated for an event to occur

JOB LIST

TC	maximum CPU time
TE	current total CPU time
TR	time remaining in current request for service
TRTS	time remaining in time-slice
NB	reason job will stop at the end of the current request

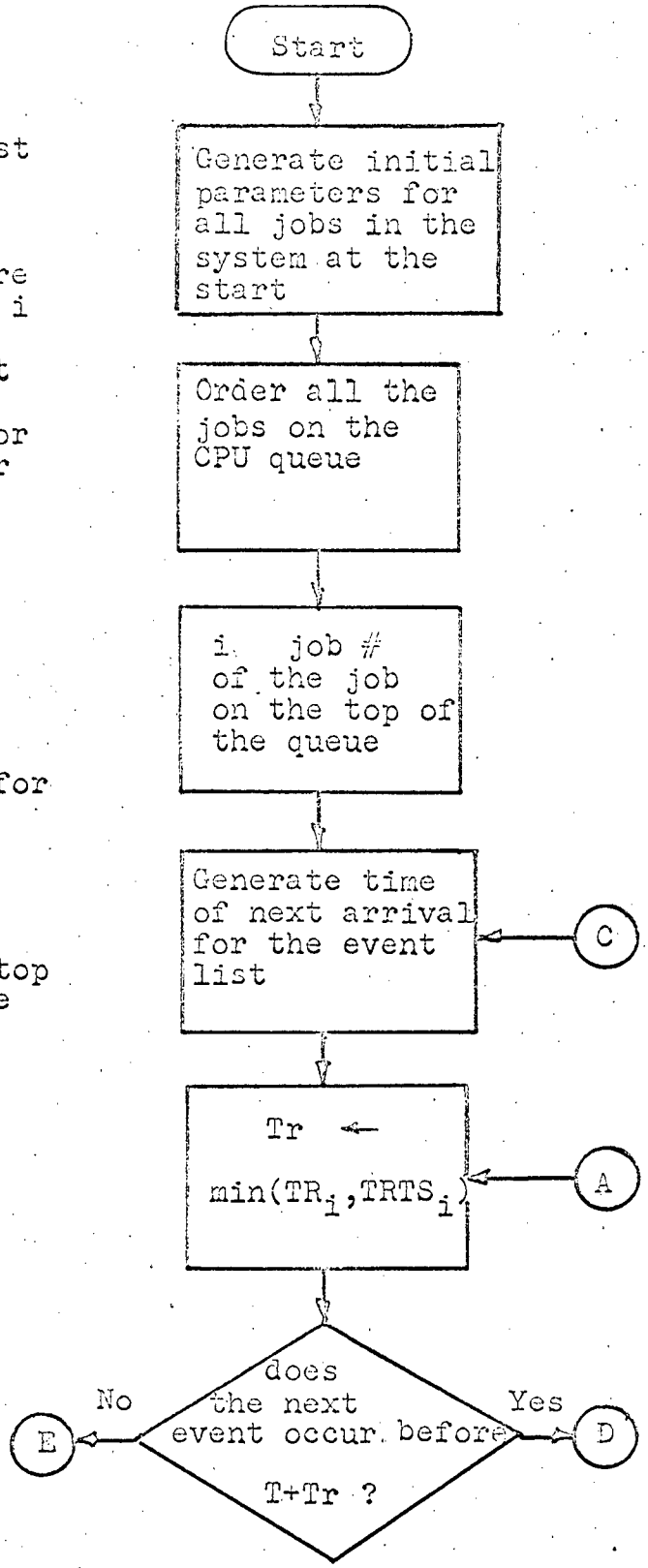


Figure 4.1 Flowchart for a model of MTS (see over)

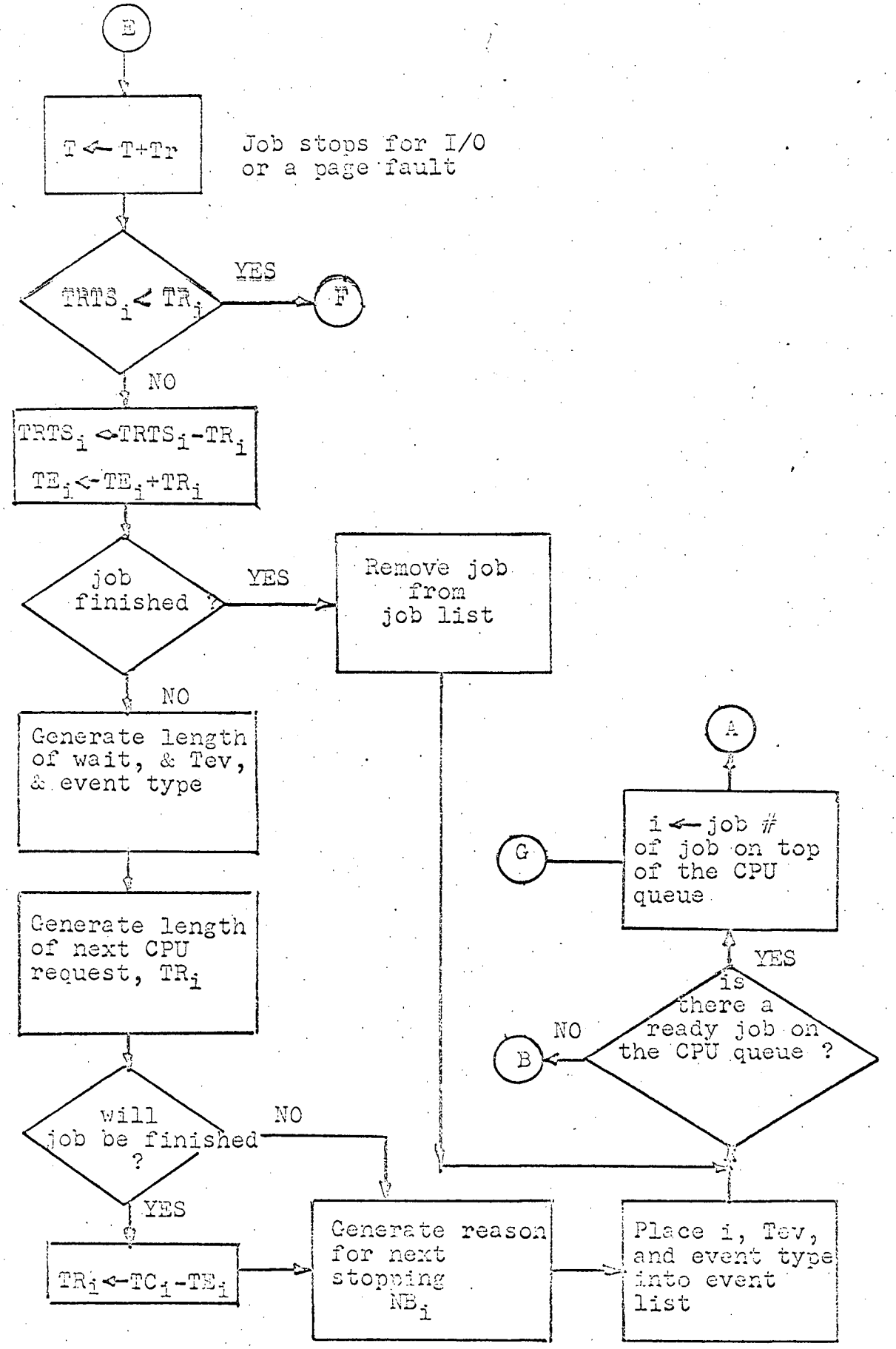


Figure 4.1 (b) Flowchart for a model of MTS (see over)

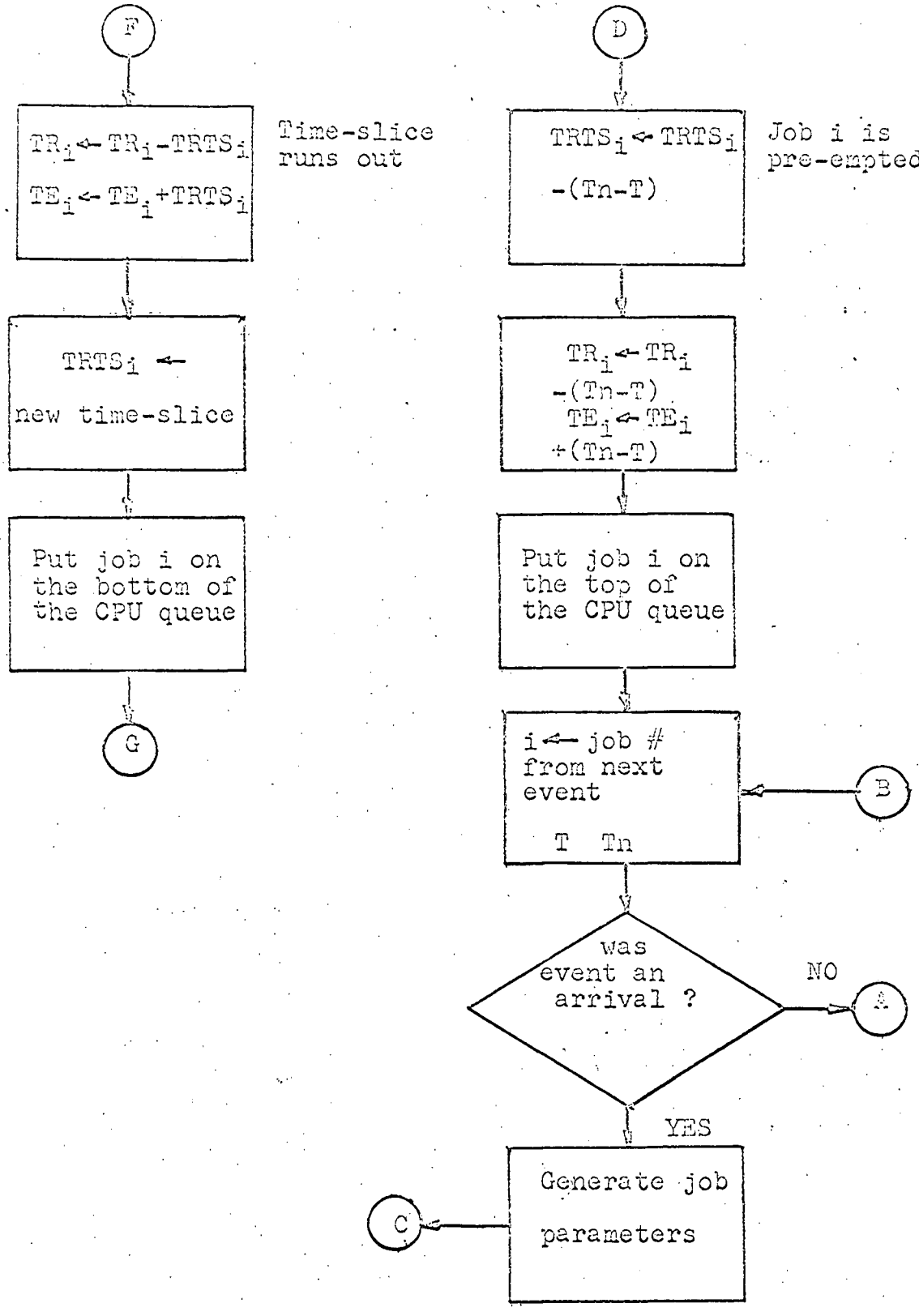


Figure 4.1 (c) Flowchart for a model of MTS

4.3 Concluding Remarks

Information on the workloads of computing systems becomes increasingly difficult to obtain as the systems become more complex. New tools must constantly be developed to overcome this problem, as a good understanding of the behavior of programs is necessary for one to design new operating systems, or modify old ones, with the idea of producing a system that makes good use of the hardware available (to minimize waste), and also gives satisfactory performance to users. Basing an operating system design on faulty assumptions about the characteristics of programs, is bound to be a hit and miss affair.

The purpose of this study has been to measure some of the important characteristics of programs at the University of British Columbia. The Data Collection Facility written by Dr. T. B. Pinkerton for the MTS operating system has proven to be a valuable tool to this end.

BIBLIOGRAPHY

1 Aho, A. V., Denning, P. J., Ullman, J. D., "Principles of Optimal Page Replacement", Journal of the ACM, vol. 18, 1 (Jan 1971).

2 Alexander, M. T., "Time-Sharing Supervisor Programs", University of Michigan Computing Cetner Memorandum (May 1969).

3 Alexander, M. T., & K. M., "Data Collection Facility and Data Analysis Program for MTS", University of Michigan Computing Center Memorandum #M200 (Nov 1971).

4 Bard, Y., "Performance Criteria for a Time-Sharing System", IBM Syst. Journal, vol. 10, 3 (1971).

5 Baskett, F., Browne, J., Raike, W., "The Management of a Multi-Level Non-Paged Memory System", Proc. Of AFIPS 1970 SJCC, vol. 36.

6 Berrettoni, N. J., "Practical Applications of the Weibull Distribution", Industrial Quality Control, vol. 21, 2 (Aug 1964).

7 Bonner, A. J., "Using System Monitor Output to Improve Performance", IBM Syst. Journal, vol. 8, 4 (1969).

8 Brawn, B., Gustavson, F., "Program Behavior in a Paging Environment", Proc. Of AFIPS 1968 FJCC, vol. 33.

9 Bryan, G. E., "20,000 Hours at the Console: A Statistical Summary", Proc. Of AFIPS 1967 FJCC, vol. 31.

10 Campbell, D. J., Heffner, W. J., "Measurement Analysis of Large Operating Systems During System Development", Proc. Of AFIPS 1968 FJCC, vol. 33, part 1.

11 Cantrell, H. N., Ellison, A. L., "Multiprogamming System Performance Measurement and Analysis", Proc. Of AFIPS 1968 SJCC, vol. 32.

12 Cheng, P. S., "Trace Driven System Modeling", IBM Syst. Journal, vol. 8, 4 (1969).

13 Coffman, E. G., Varian, L. C., "Further Experimental Data on the Behavior of Programs in a Paging Environment", Communications of the ACM, vol. 11, 7 (July 1968).

14 Coffman, E. G., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System", Journal of the ACM, vol. 16, 1 (Jan 1969).

15 Daley, R., Dennis, J. B., "Virtual Memory, Process, and Sharing in MULTICS", Communications of the ACM, vol. 11, 5 (May 1968).

16 Denning, P. J., "The Working Set Model for Program Behavior", Communications of the ACM, vol. 11, 5 (May 1968).

17 Denning, P. J., "Virtual Memory", Computing Surveys, vol. 2, 3 (Sept 1970).

18 Dennis, J. B., "Segmentation and Design of Multiprogramming Computing Systems", Journal of the ACM, vol.12, 4 (Oct 1965).

19 Drummond, M. E. Jr., "A Perspective on System Performance", IBM Syst. Journal, vol 8, 4 (1969).

20 Estrin, G., Kleinrock, L., "Measures, Models, and Measurements for Time-Shared Computer Systems", Proc. Of the ACM 22 National Conference (1967).

21 Fife, D. W., "An Optimization Model for Time-Sharing", Proc. Of AFIPS 1966 SJCC, vol. 28.

22 Fine, G. H., Jackson, C. W., McIsaac, P. V., "Dynamic Behavior of Programs Under Paging", Proc. Of the ACM 21 National Conference (1966).

23 Freibergs, I. F., "The Dynamic Behavior of Programs", Proc. Of AFIPS 1968 FJCC, vol. 33.

24 Fuchs, E., Jackson, P. E., "Estimates of Distributions of Random Variables for Certain Computer Commumnications Traffic Models", Communications of the ACM, vol. 13, 12 (Dec 1970).

25 Gordon, G., System Simulation, Prentice-Hall, Englewood Cliffs, New Jersey, 1969.

26 Hatfield, D., Gerald, J., "Program Restructuring Techniques in Virtual Memory", IBM Syst. Journal, vol. 10, 3 (1971).

27 Kilburn, T., Edwards, D., Lanigan, M., Sumner, F., "One Level Storage Systems", IEEE Trans. On Elec. Computers, vol. 11, 2 (Apr 1962).

28 Kleinrock, L., Coffman, E. G., "Distribution of Attained Service in Time-Shared Systems", Jour. Comput. Syst. Sci., vol. 1 (1967).

29 Kuehner, C. J., Randell, B., "Demand Paging in Perspective", Proc. Of AFIPS 1968 FJCC, vol. 33.

30 Lauer, H., "Bulk Core in a 360/67 Time-Sharing System", Proc. Of AFIPS 1967 FJCC, vol. 31.

31 Lehman, M., Rosenfeld, J., "Performance of a Simulated Multiprogramming System", Proc. Of AFIPS 1968 FJCC, vol. 33, part 2.

32 MacDougall, M. H., "Simulation of an ECS-Based Operating System", Proc. Of AFIPS 1967 SJCC, vol. 30.

33 MacDougall, M. H., "Computer System Simulation: An Introduction", Computing Surveys, vol. 2, 3 (Sept 1970).

34 McKinney, J. M., "Analytical Time-Sharing Models", Computing Surveys, vol. 1, 2 (June 1969).

35 Nakamura, G., "A Feedback Queueing Model for an Interactive Computer System", Proc. Of AFIPS 1971 FJCC, vol. 39.

36 Nielsen, N. R., "The Simulation of Time-Sharing Systems", Communications of the ACM, vol. 10, 7 (July 1968).

37 O'Neil, R. W., "Experience Using a Time-Shared Multiprogrammed System With Dynamic Address Relocation Hardware", Proc. Of AFIPS 1967 SJCC, vol. 30.

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

39 Pinkerton, T. B., "Program Behavior and Control in Virtual Storage Computer Systems", University of Michigan Tech. Rep. #4, CONCOMP Project.

40 Pinkerton, T. B., "Performance Monitoring in a Time-Sharing System", Communications of the ACM, vol. 12, 11 (Nov 1969).

41 Ruiz-Pala, E., Avila-Beloso, C., Hines, W. W., Waiting-Line Models, Reinhold Publishing Co., New York, 1967.

42 Scherr, A. L., "Time-Sharing Measurement", Datamation, vol. 12, 4 (Apr 1966).

43 Scherr, A. E., Analysis of Time-Shared Computer Systems, MIT Press, Cambridge, Mass., 1967.

44 Schwetman, H. D., DeLine, J. R., "An Operational Analysis of a Remote Console System", Proc. Of AFIPS 1969 SJCC, vol. 34.

45 Sebestyen, G. S., Decision-Making Processes in Pattern Recognition, Macmillan, New York, 1962.

46 Shemer, J. E., Heying, D.W., "Performance Modeling and Empirical Measurement in a System for Batch and Time-Sharing Users", Proc. Of AFIPS 1969 FJCC, vol. 35.

47 Stanley, W. I., "Measurement of System Operational Statistics", IBM Syst. Journal, vol. 8, 4 (1969).

48 von Maydell, U. M., Gatha, A. K., "Analysing the Performance of the CP/67 Time Sharing System", INFOR, vol. 9, 3 (Nov 1971).

49 Zeigler, J. R., Time-Sharing Data Processing Systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1967.

APPENDIX A THE DATA COLLECTION FACILITY

The DCF consists of two UMMPS jobs called STAT and STATSW and of supervisor subroutine which is invoked from various points in the supervisor, and which may also be invoked by a SVC.

The STAT job manages a ring of buffers which it writes onto a tape when full. The STATSW job is responsible for setting and resetting a word of switches in each entry in the job table, which indicate whether or not data is to be collected for that job, and what items are to be collected for it if data is to be collected. The supervisor subroutine is responsible for placing the items in the buffers. Each item consists of a two word prefix containing the item type, item length, ID of job to which the item applies, an indication of which CPU the item occurred on, and the time of the event, followed by from zero to six words of specific information about the event.

The presence of such a facility in a system, must cause some degradation in performance, but the amount in this case is small. Roughly 10 us of CPU time is required every time the supervisor is entered to see if the DCF is currently active. If it is, the buffer entry subroutine can place each item into a buffer at a cost of about 65 us each. In addition, the SVC call from a job program to save an item requires about 330 us of CPU time, but only about 10% of items are collected in this manner. My measurements have shown that on a fairly busy day, with the DCF operating, roughly 0.6% of the CPU time is used by the STAT job, and about 4% of the CPU time by buffer entry subroutine and SVC call to enter data in buffers. The DCF also uses roughly 3%

of the real core available to users.

It is possible for no buffer to be available when an item is to be entered. In this case, a count is kept of all items missed until a buffer becomes available and a special item indicating this number is inserted as the first item of the new buffer. This situation, however, occurs only rarely.

The data collection facility in MTS is currently the most flexible and detailed available.

APPENDIX B FITTING WEIBULL AND HYPER-EXPONENTIAL DISTRIBUTIONS

THE WEIBULL DISTRIBUTION

The Weibull distribution has the general form

$$F(x) = 1-e^{-\alpha(x-\gamma)^b}$$

where γ is a shift parameter which is normally equal to zero when one is interested in time dependent variables. This gives us the following form for the Weibull distribution

$$F(t) = 1-e^{-at^b}$$

The mean of this distribution is given by

$$\alpha^{\beta}\Gamma(\beta+1)$$

and the variance by

$$\alpha^{2\beta}\{\Gamma(2\beta+1)-\Gamma^2(\beta+1)\}$$

where $\alpha = 1/a$, $\beta = 1/b$

In order to obtain estimates of the parameters a and b we may apply the following transformation

$$1-F(t) = e^{-at^b}$$

$$\ln(1-F(t)) = -at^b$$

$$\ln(1/(1-F(t))) = at^b$$

$$\ln(\ln(1/(1-F(t)))) = \ln a + b \ln t$$

which results in a linear equation in the variables $\ln(\ln(1/(1-F(t)))$ and $\ln t$. The slope of this line is b, and the intercept is ln a. We can find two points on this line by setting each of the variables to zero in turn.

(1) $\ln t = 0$ when $t = 1$

Some suitable scaling factor can be applied to the time variable to get a value of t = 1 near to the maximum time for which data was collected and not grouped in an overflow class. Evaluating $\ln(\ln(1/(1-F(t))))$ at this point gives a value for ln a.

(2) $\ln(\ln(1/(1-F(t)))) = 0$

when $1/(1-F(t)) = e$

or $F(t) = 0.6321$

from the sample distribution, find a value T such that $F(T) = 0.6321$. The parameter b may now be calculated from

$$b = -\ln a/\ln T.$$

An inverse function exists for the Weibull distribution, and is derived as follows:

$$P(x \leq t) = 1-e^{-at^b}$$

$$1-p = e^{-at^b}$$

$$\ln p = -at^b$$

$$t^b = \ln[(1/p)^{\alpha}]$$

$$\ln t = \beta \ln(\ln[(1/p)^{\alpha}]$$

$$\ln t = \ln(\ln[(1/p)^{\alpha}])^{\beta}$$

$$t = [\alpha \ln(1/p)]^{\beta}$$

THE HYPER-EXPONENTIAL DISTRIBUTION

A hyper-exponential distribution may be represented by the general form

$$F(t) = se^{-2s\lambda t} + (1-s)e^{-2(1-s)\lambda t}$$

The mean of this distribution is given by $1/\lambda$, and the variance by

$$\sigma^2 = [(1-2s+2s^2)/(2s-2s^2)](1/\lambda)^2$$

Given the mean and variance of a sample distribution, we may calculate a value for the parameter s as follows:

$$\text{let} \quad k = \sigma^2\lambda^2$$

$$k = (1-2s+2s^2)/(2s-2s^2)$$

this yields

$$s = (2k+2-2(k^2-1)^{1/2})/(4k+4)$$

which has a solution as long as $k \geq 1$, which must be true for the hyper-exponential distribution since the variance is greater than the square of the mean.

APPENDIX C MTS DATA

Data was collected for this study on two occasions, first on June 19, 1971, and secondly on November 11, 1971. The first data tape (tape 1) represents a light load on the system, with only about 7 interactive jobs, and 2 batch jobs running simultaneously. The second data tape (tape 2) was collected to observe the system under heavier load. During the time that collection took place, about 19 interactive jobs, and 2 to 3 batch jobs were running simultaneously. The number of batch jobs is restricted at times of heavy usage by interactive jobs to prevent the system from getting bogged down.

As can be seen in Table C.1, even under the heavier load condition, the CPU's are idle 48% of the time. The utilization of the CPU's at The University of British Columbia's computing centre approaches 100% during the midnight hours, when most of the large batch jobs are run.

It is interesting to note that the DCF uses considerably more of the system's resources (Table C.1) than Pinkerton has indicated in either [31], or [32]. The times were calculated using his formula of 65 us for the collection of 90% of the items, and 330 us for the remainder. The times for the buffer managing subroutine (STAT) were measured directly from the data tapes.

A frequency distribution of the items on the tapes may be found in Table C.2, and a comprehensive description of the items in [3].

	Tape 1	Tape 2
Date	19-6-71	12-11-71
Start Time	15:25	14:03
Durration	97 min	95 min
No. of Items	753389	2749238
CPU's Idle	83.6%	48.4%
CPU Time for DCF	4.0%	4.8%
data collection	3.5%	4.2%
STAT job (buffers)	0.5%	0.6%
CPU Time for PDP	2.9%	6.6%
Total CPU Time for Non-user Tasks	12.3%	16.8%

Table C.1 MTS Data

STAT ITEM	# on TAPE 1	# on TAPE 2
OVERFLOW	0	3
DATE	1	1
ADTOTOP	35453	201681
POPQ	32041	130960
WAYT	91326	299314
UNWAYT	88658	297475
QUEUE	349412	1187322
STATSW	349	397
PAGINSTR	5048	36669
PAGINDON	5048	36872
PAGOUTST	5552	42132
PAGOUTDN	5476	41689
PAGRECLM	8276	30210
GETVMPAG	9196	32658
FREVMPAG	9467	32760
SYNCHRON	91	95
VMPAGES	12895	45484
WAITFOR	1790	2046
UNLOAD	212	595
LOAD	211	634
FREESPAC	29569	96088
GETSPACE	28951	95287
DSRIN	17183	69336
DSROUT	17184	69330
TOTALS	753389	2749238

Table C.2 Frequency of STAT items.

APPENDIX D STORAGE PROFILES

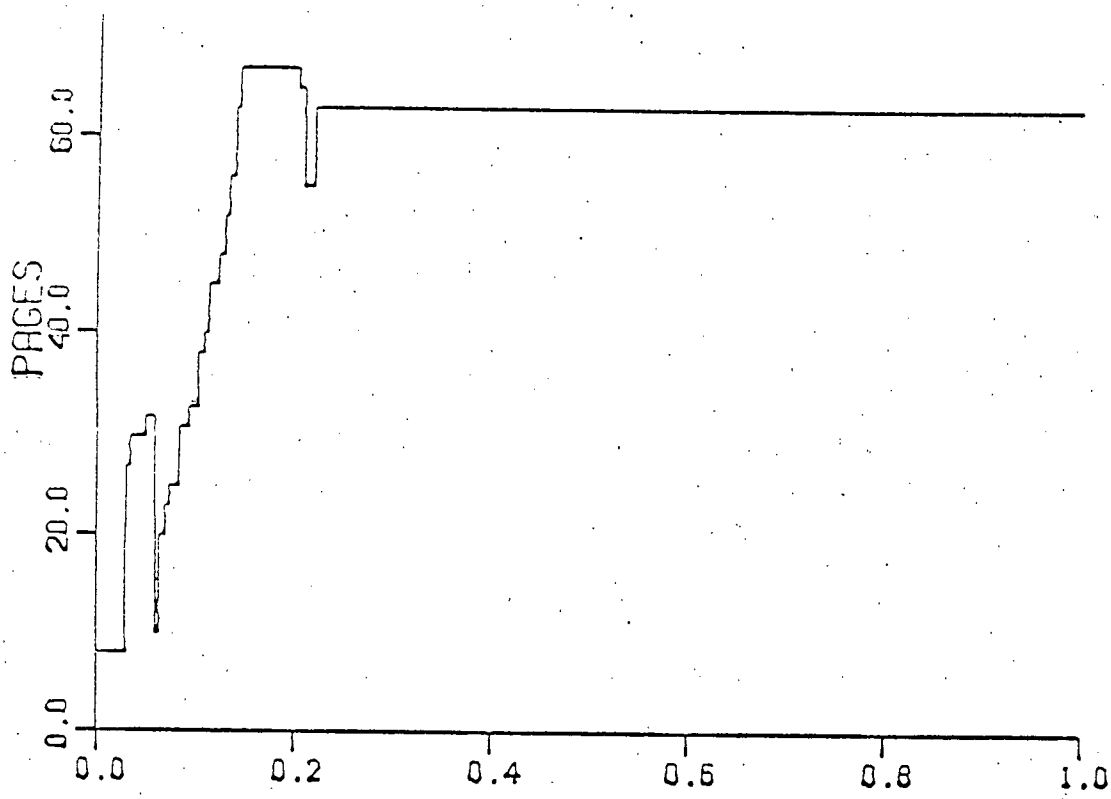


Figure D.1

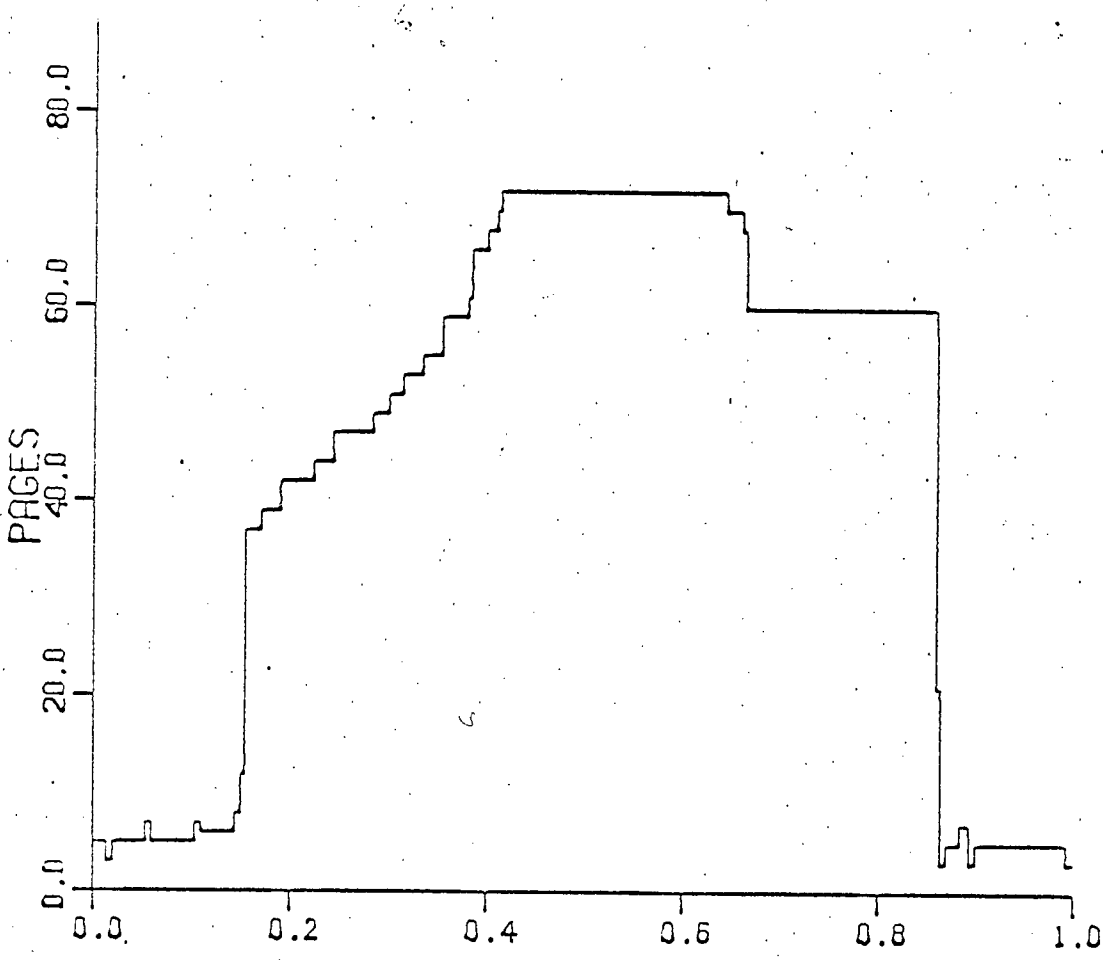


Figure D.2

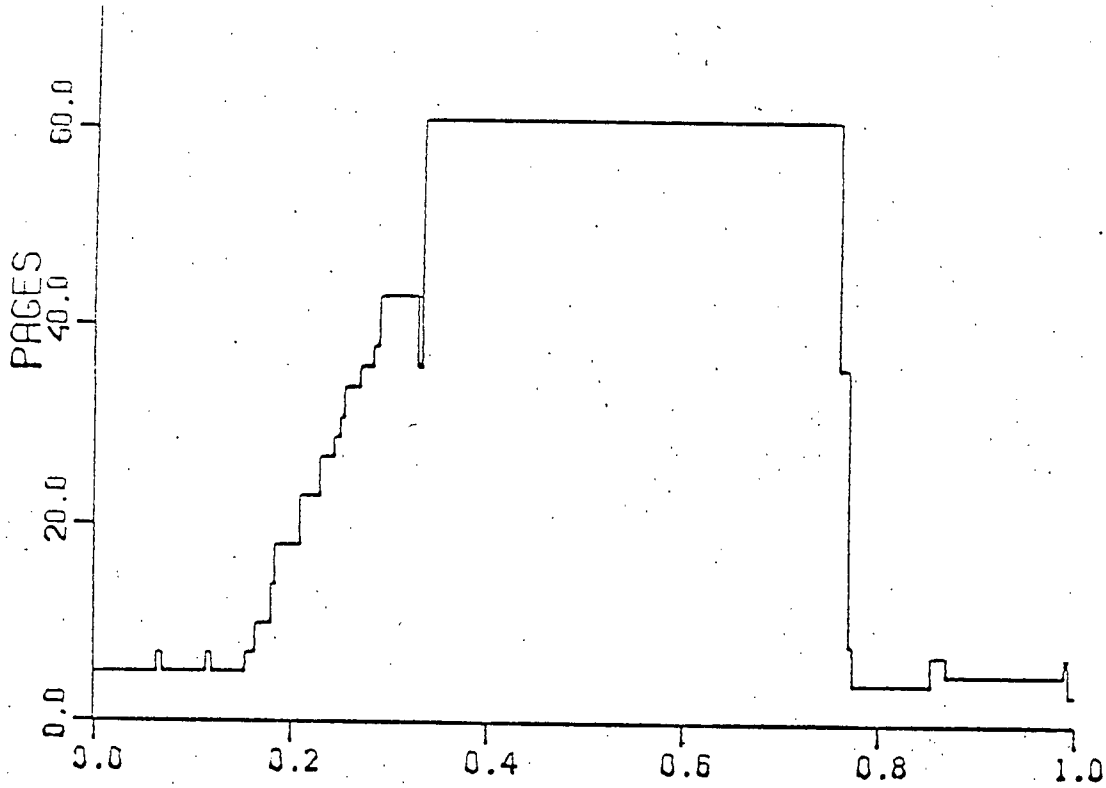


Figure D.3

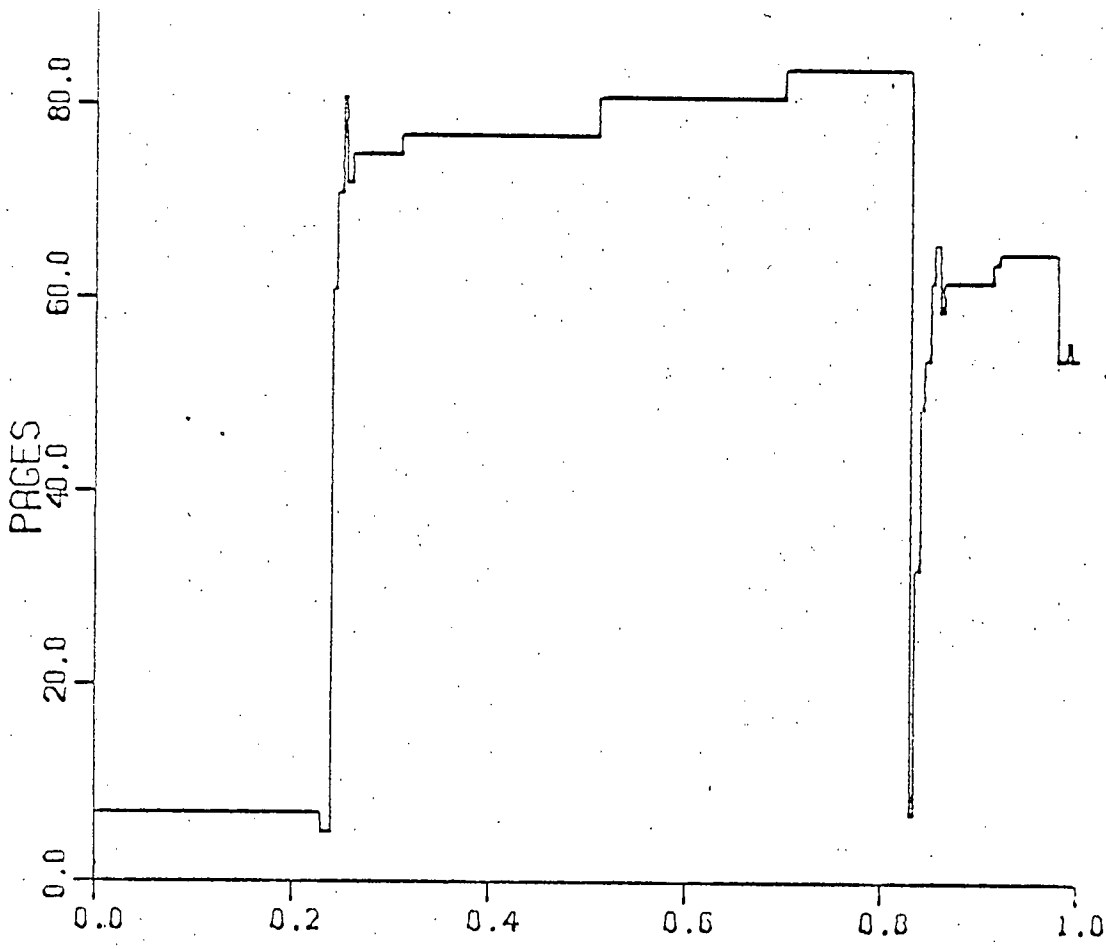


Figure D.4

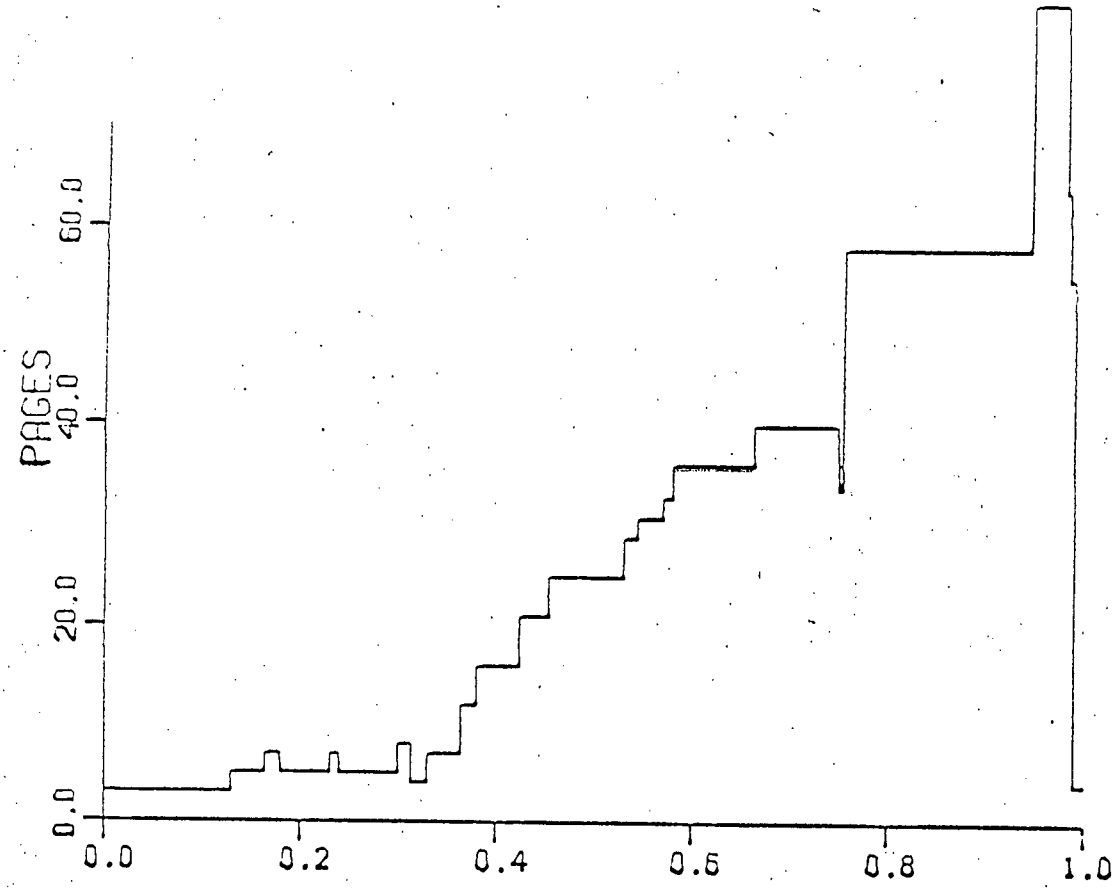


Figure D.5

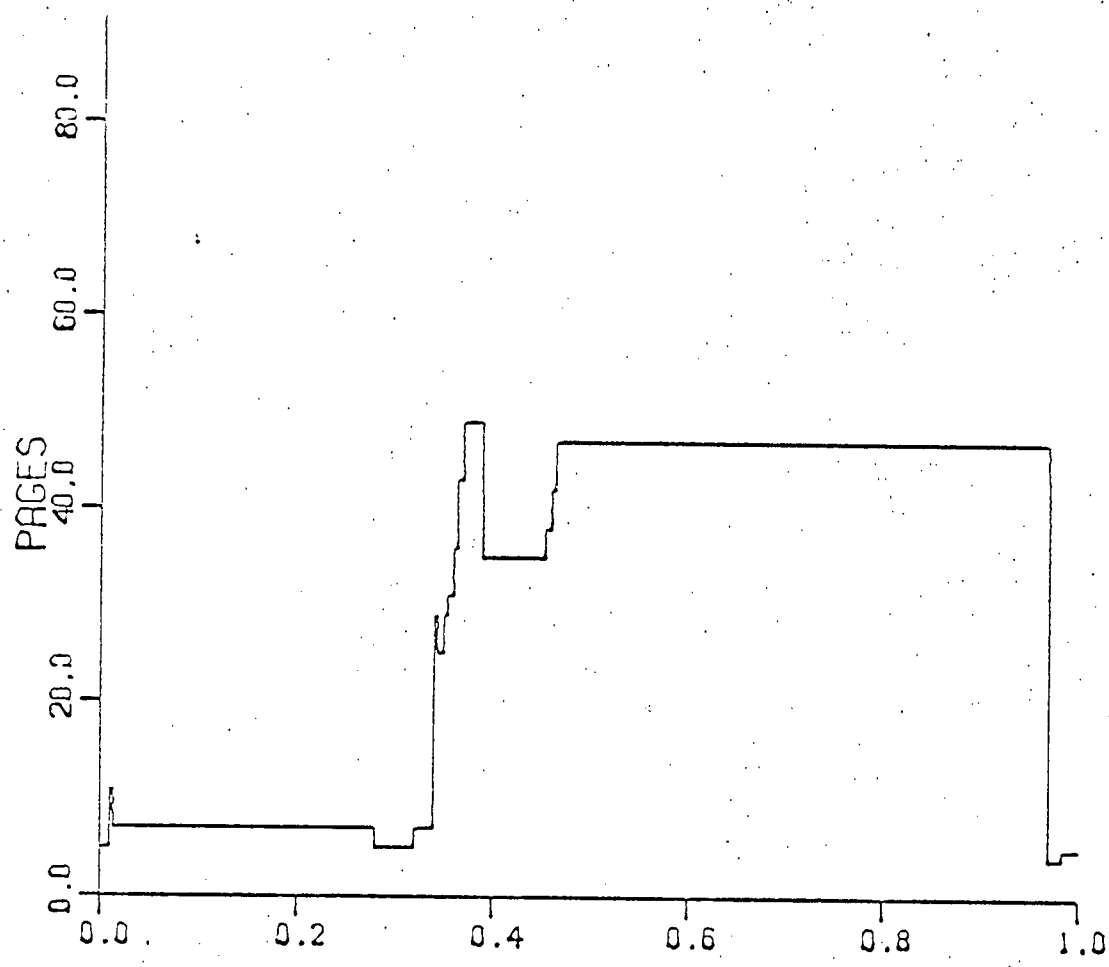


Figure D.6

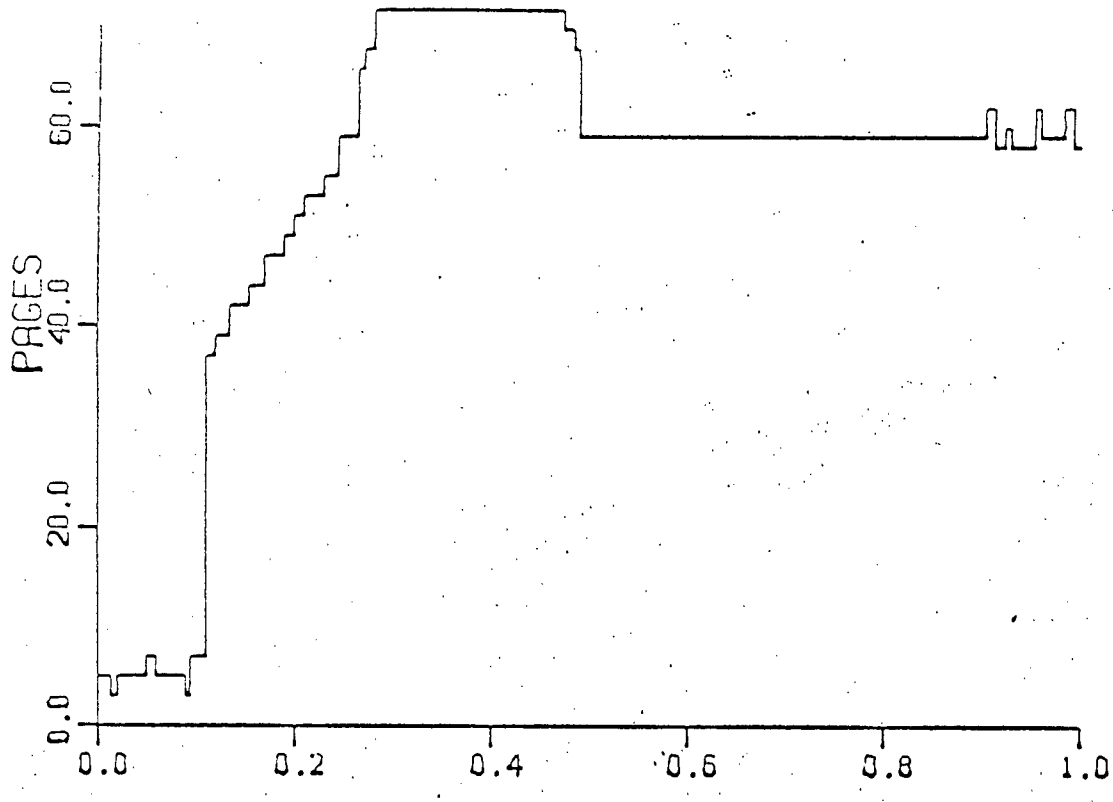


Figure D.7

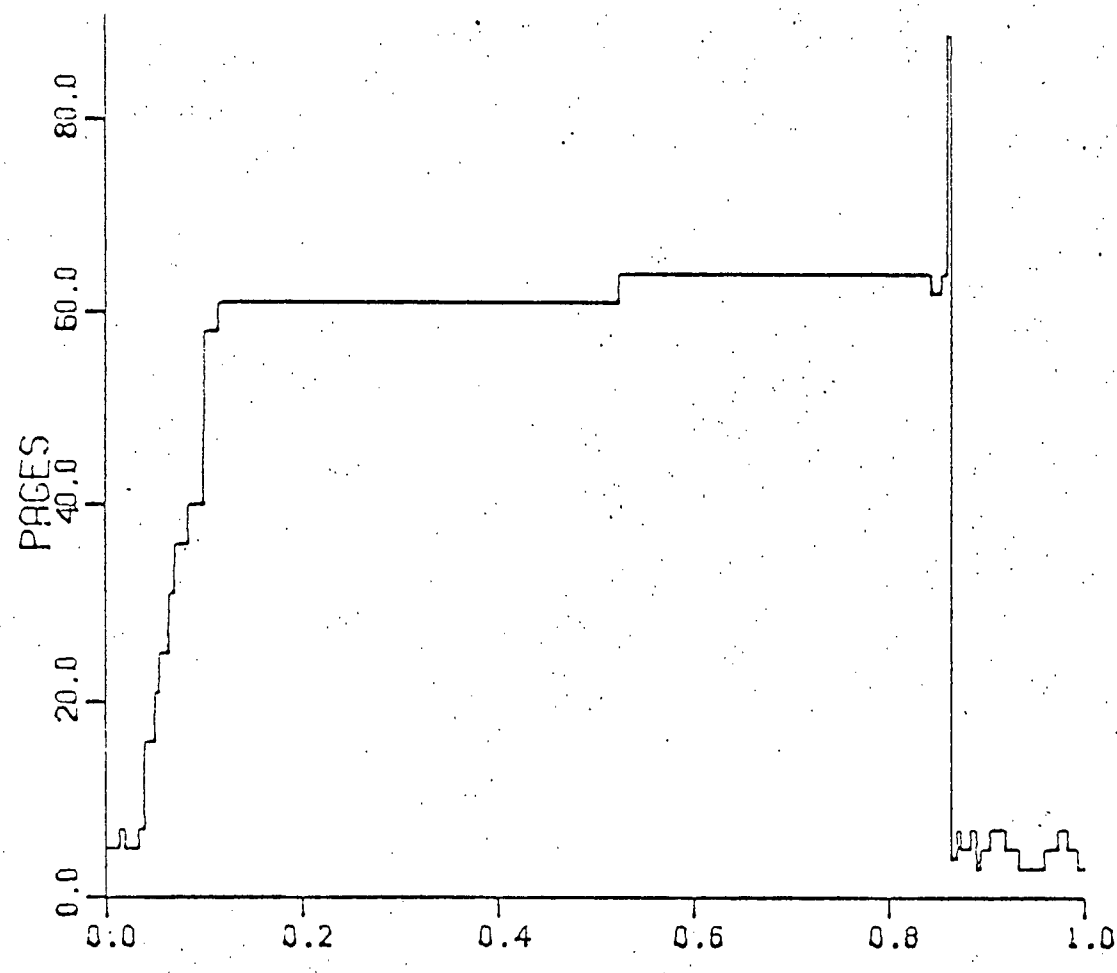


Figure D.8

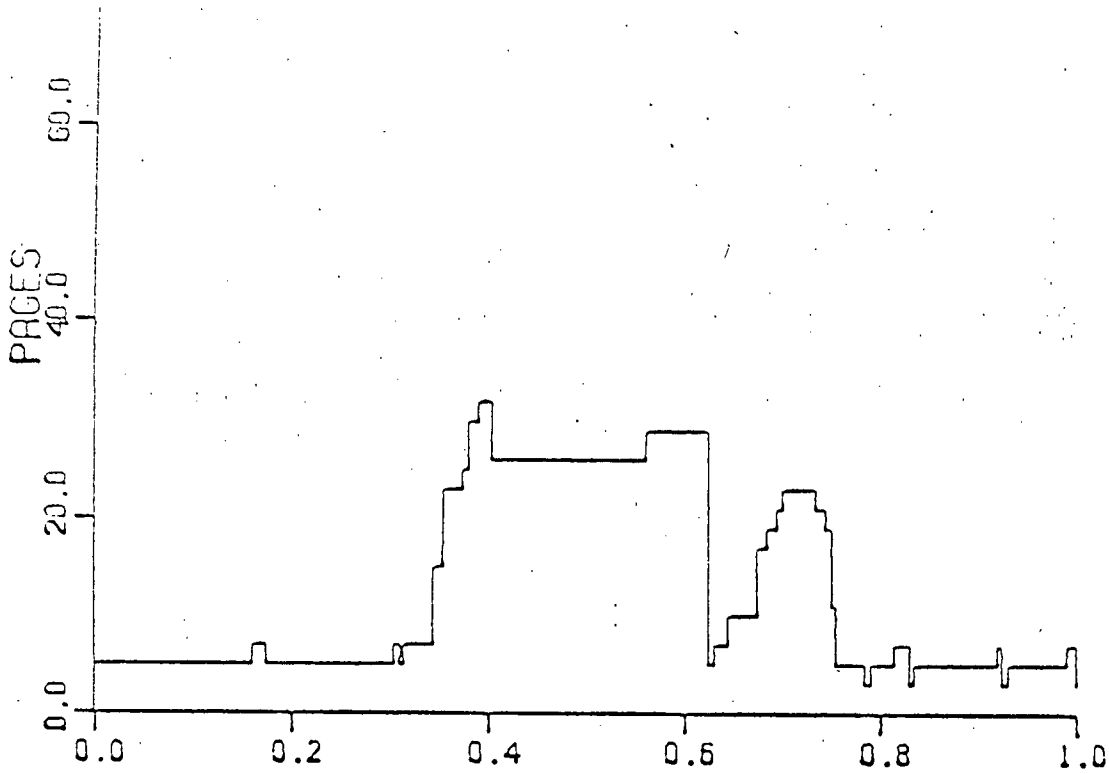


Figure D.9

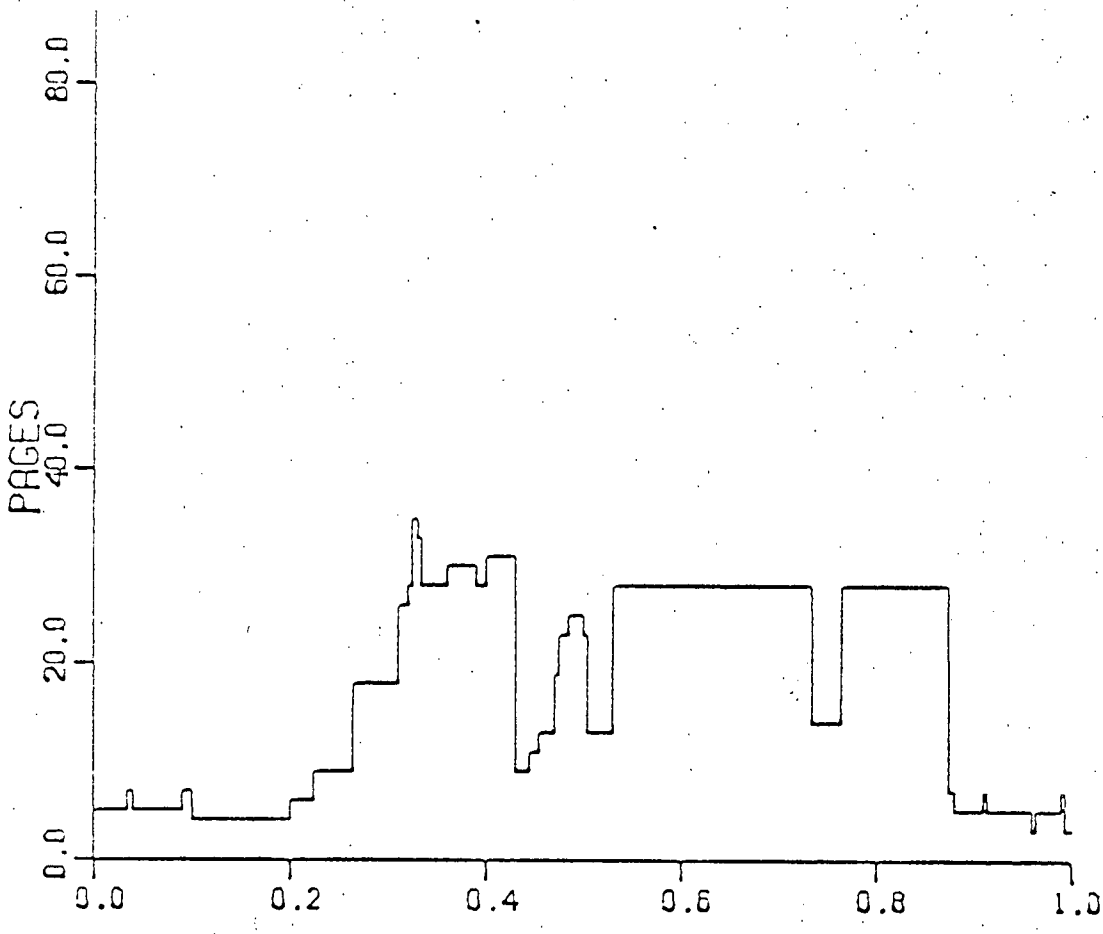


Figure D.10

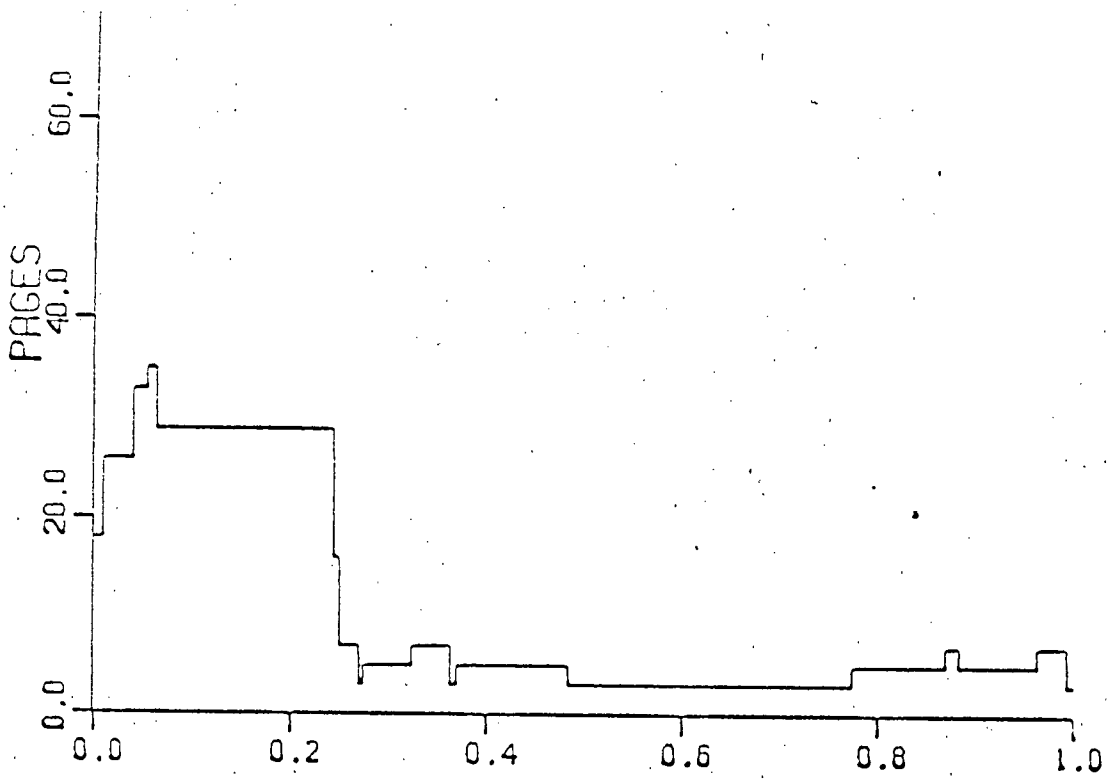


Figure D.11

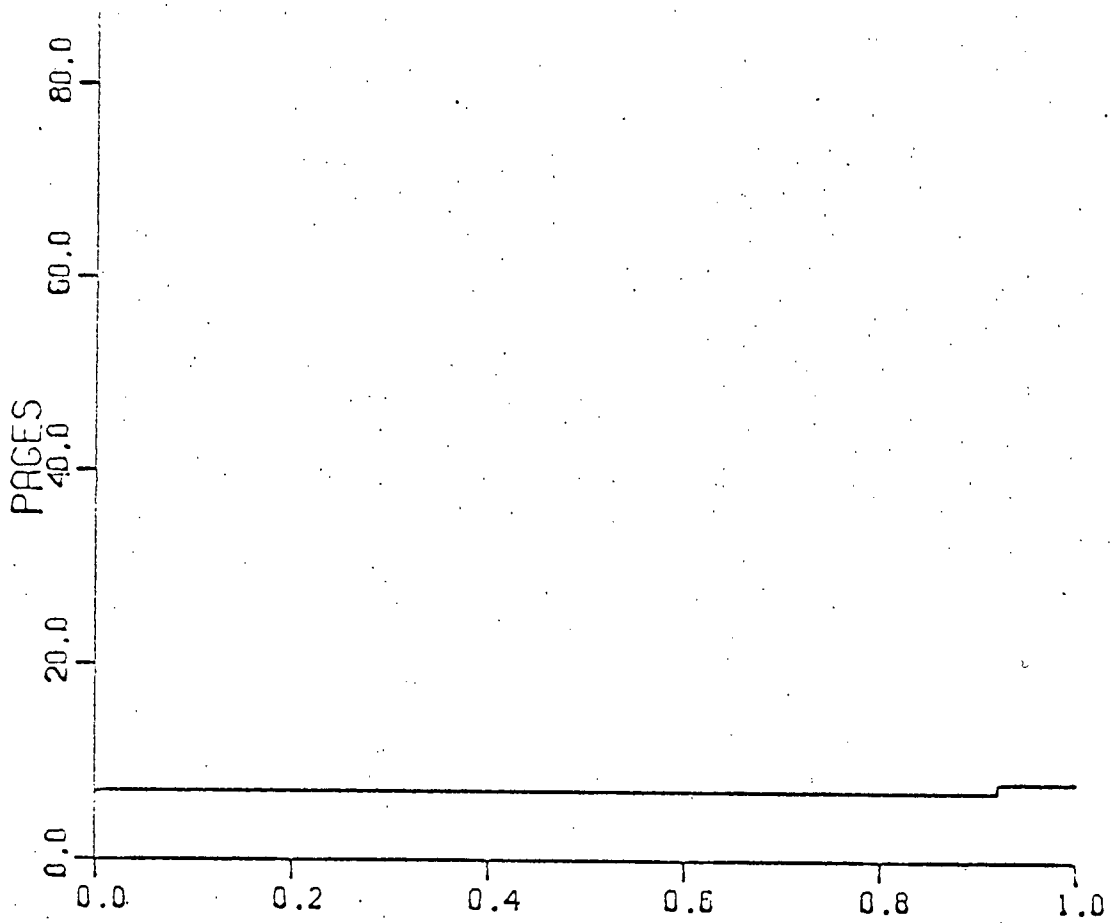


Figure D.12

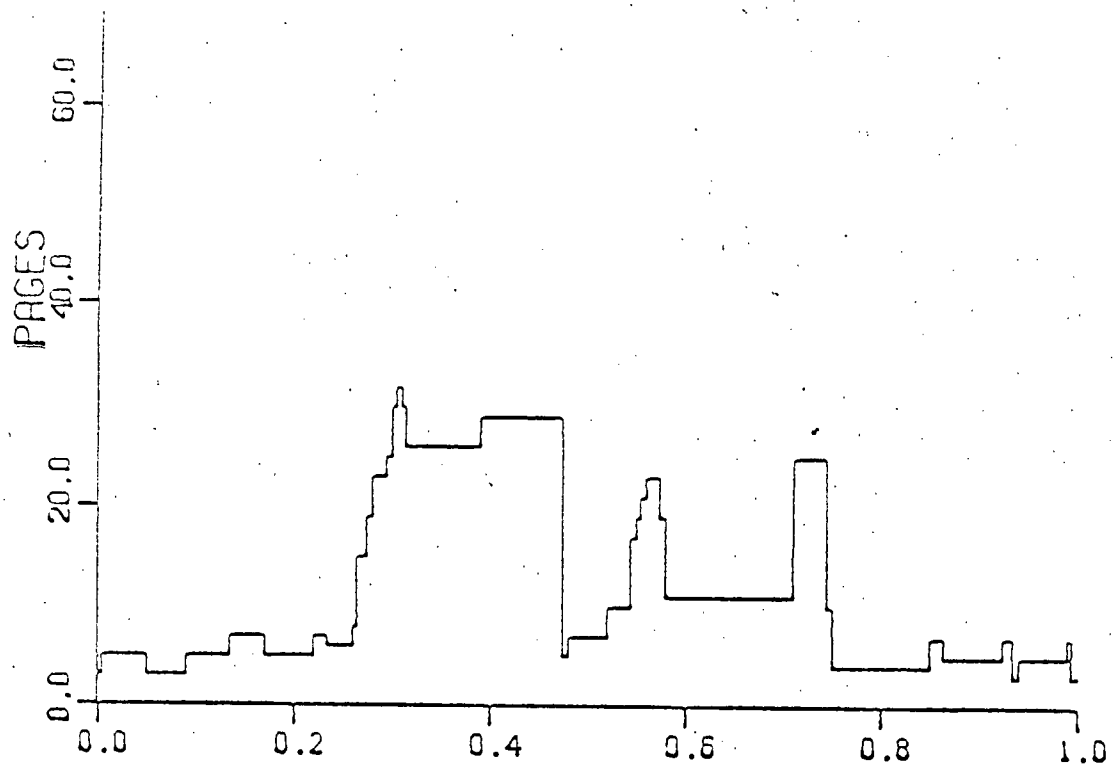


Figure D.13

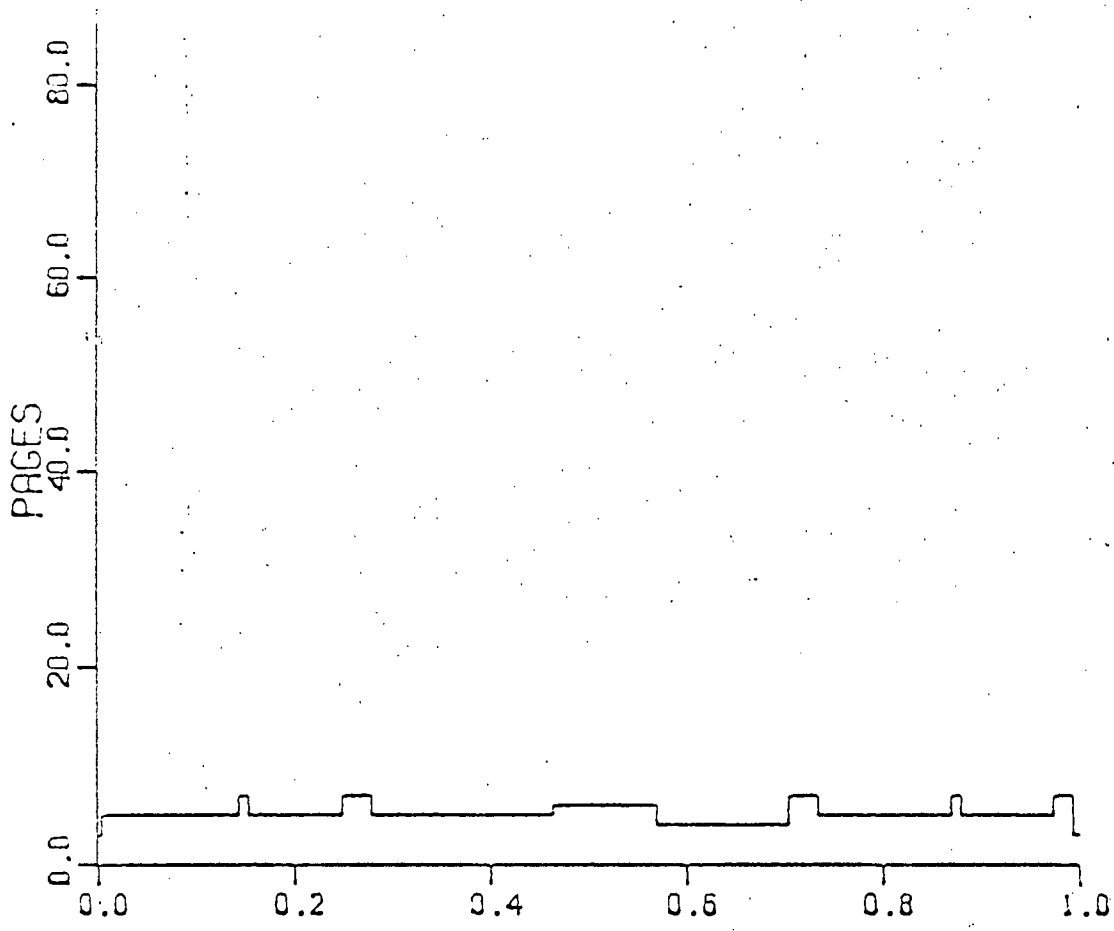


Figure D.14

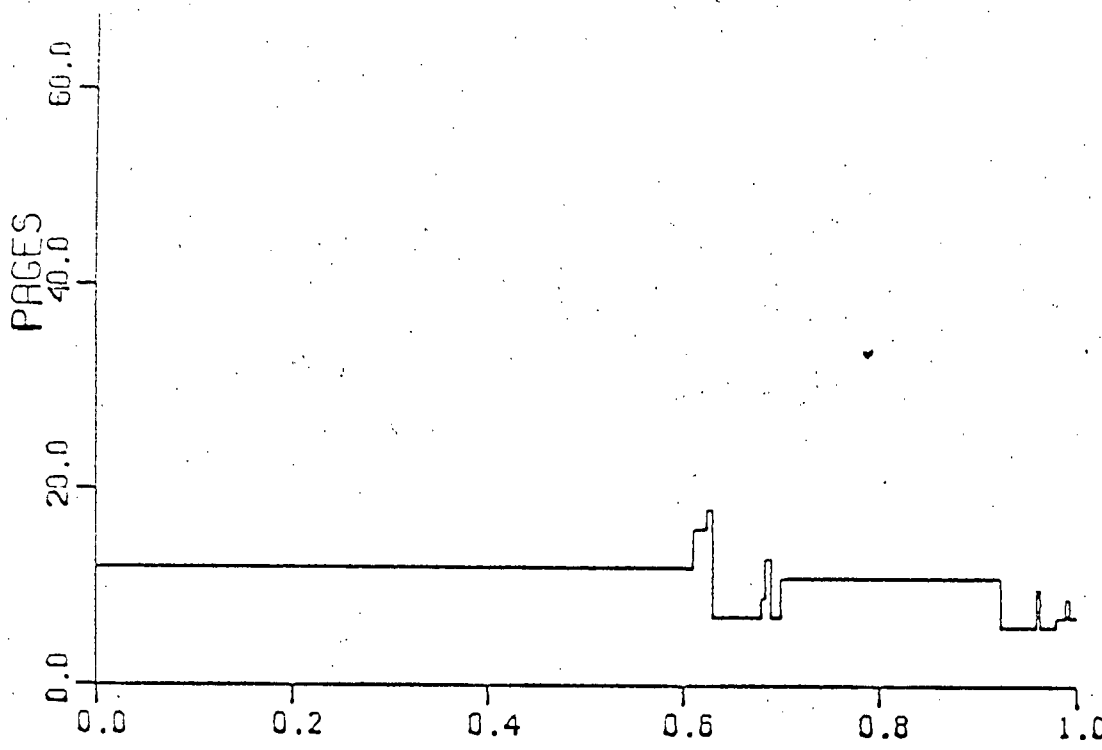


Figure D.15

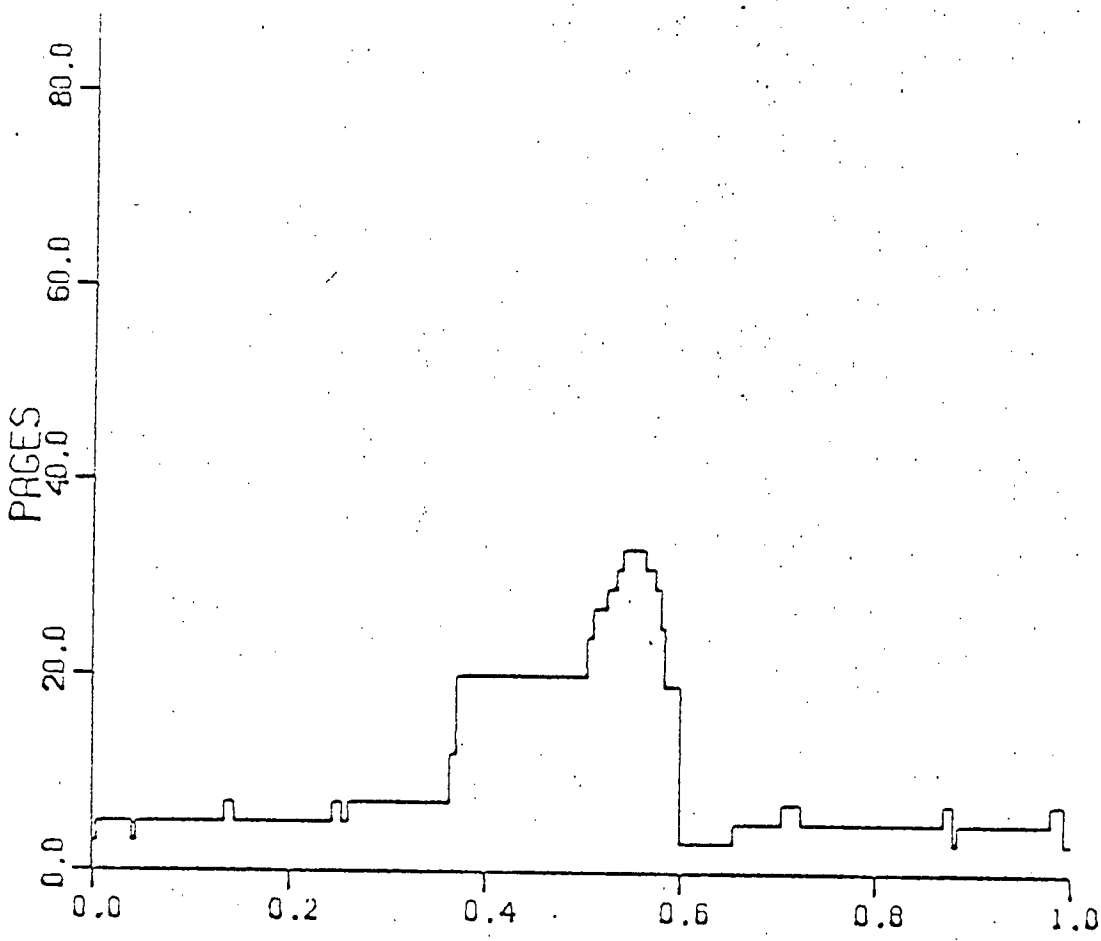


Figure D.16

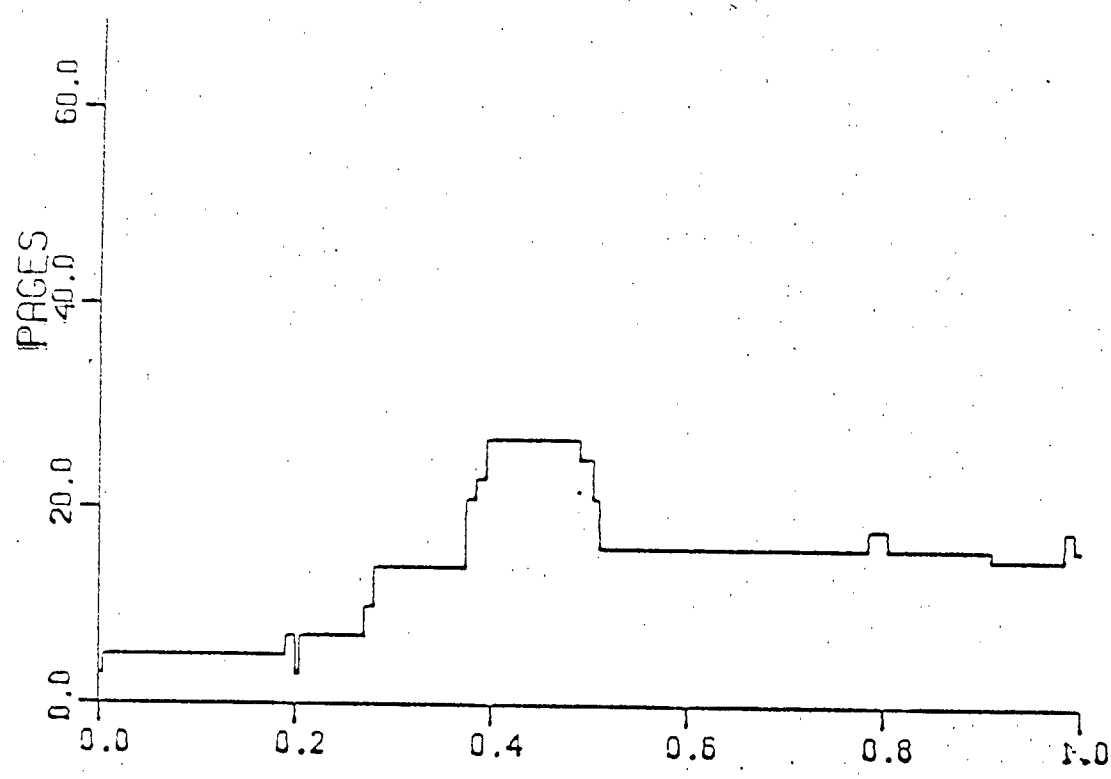


Figure D.17

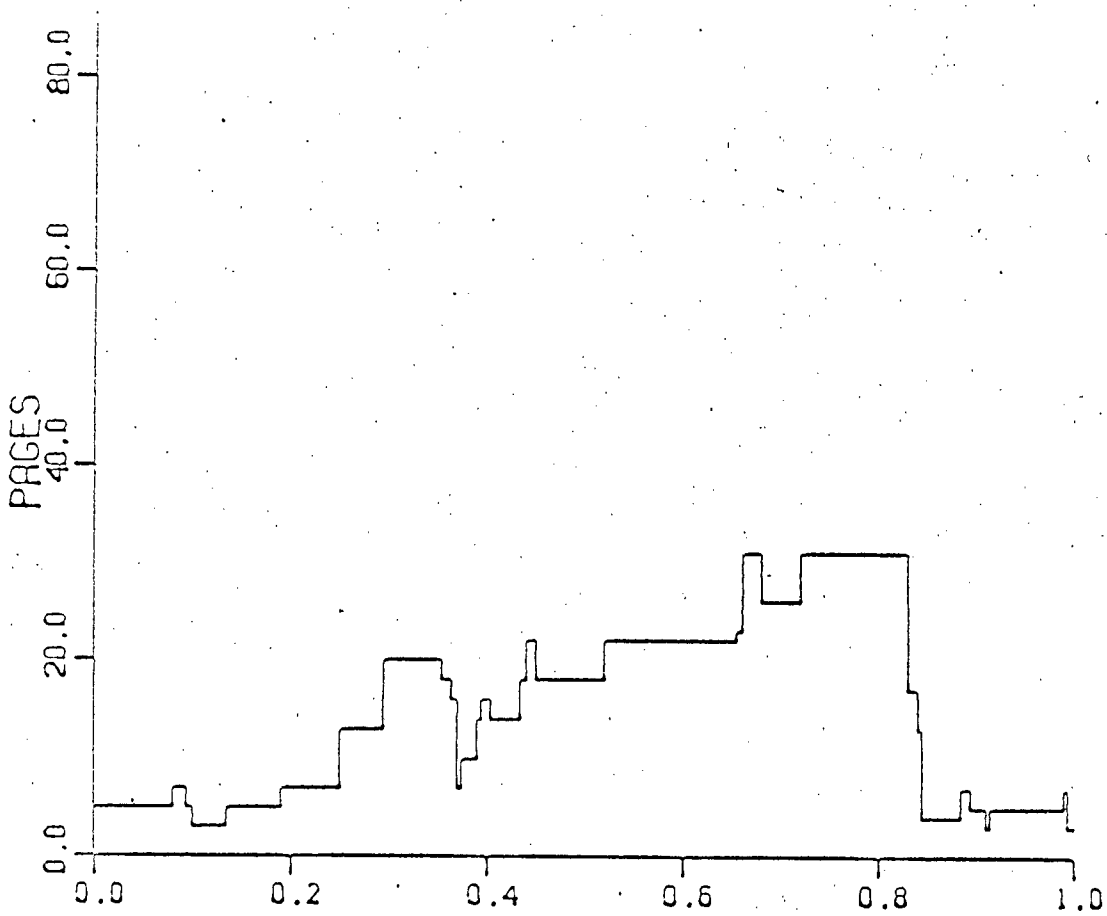


Figure D.18

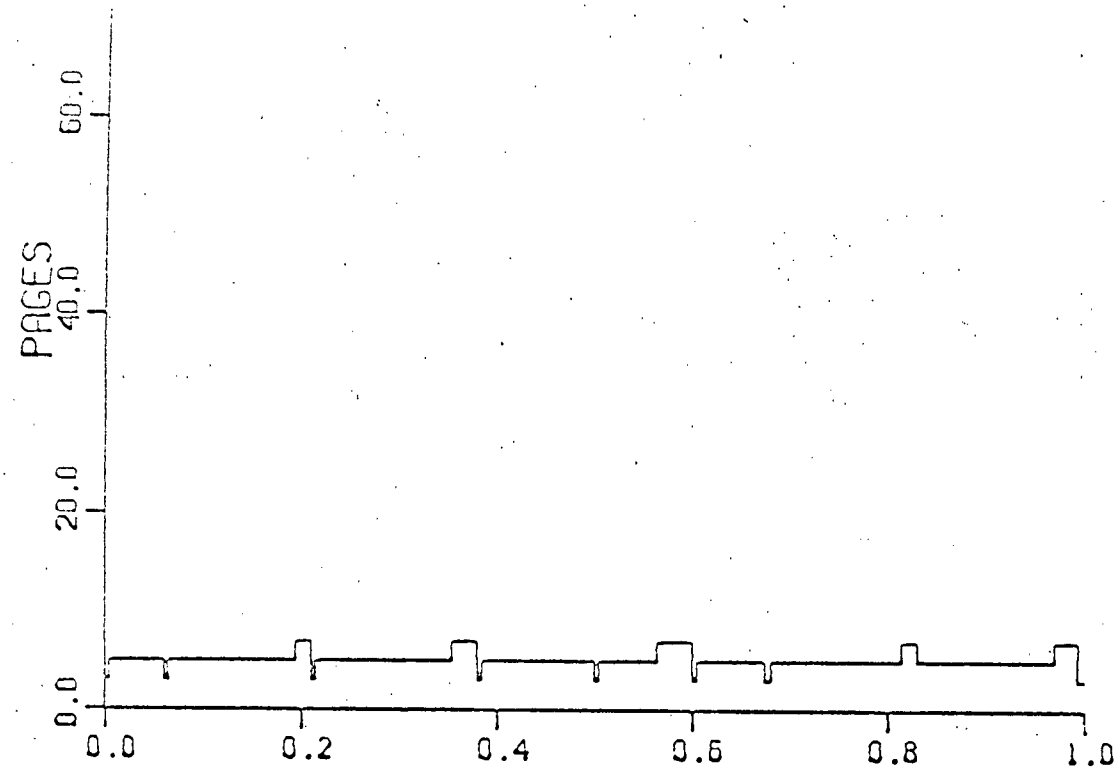


Figure D.19

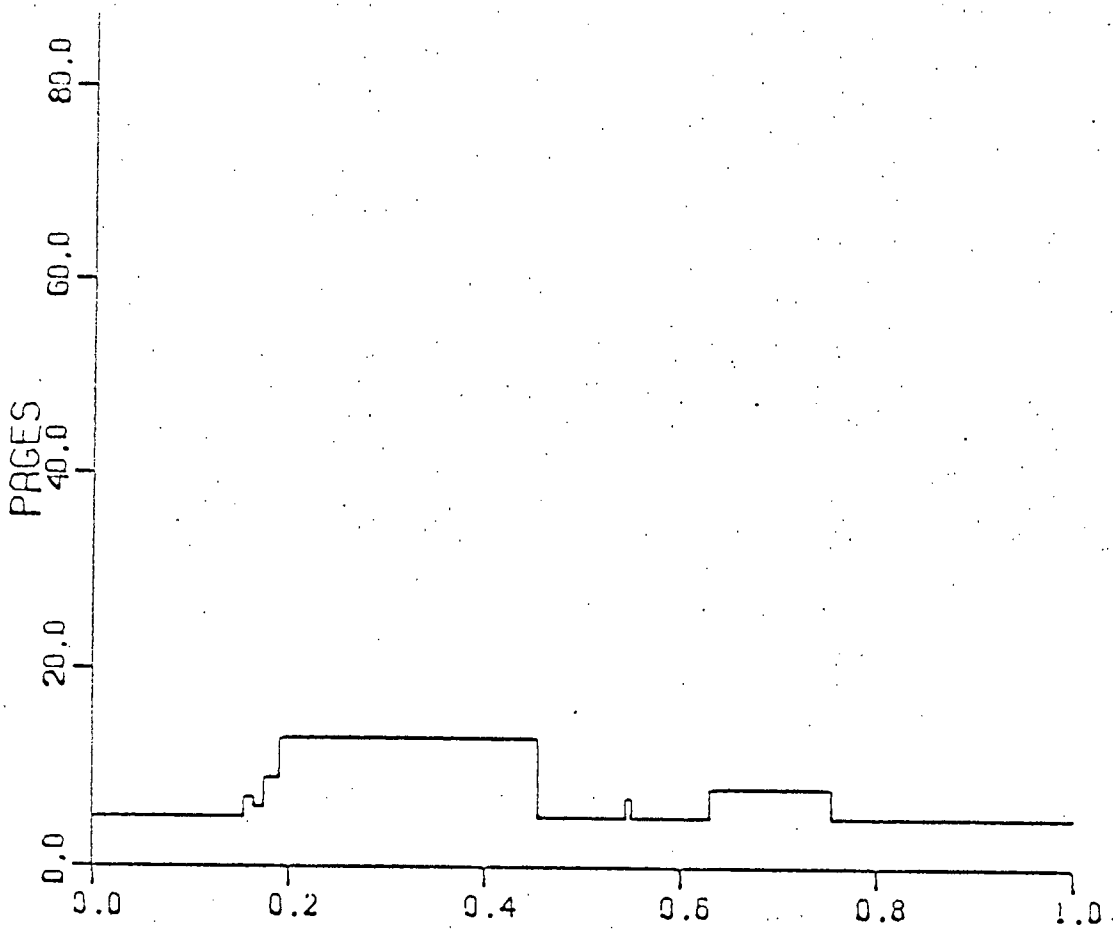


Figure D.20

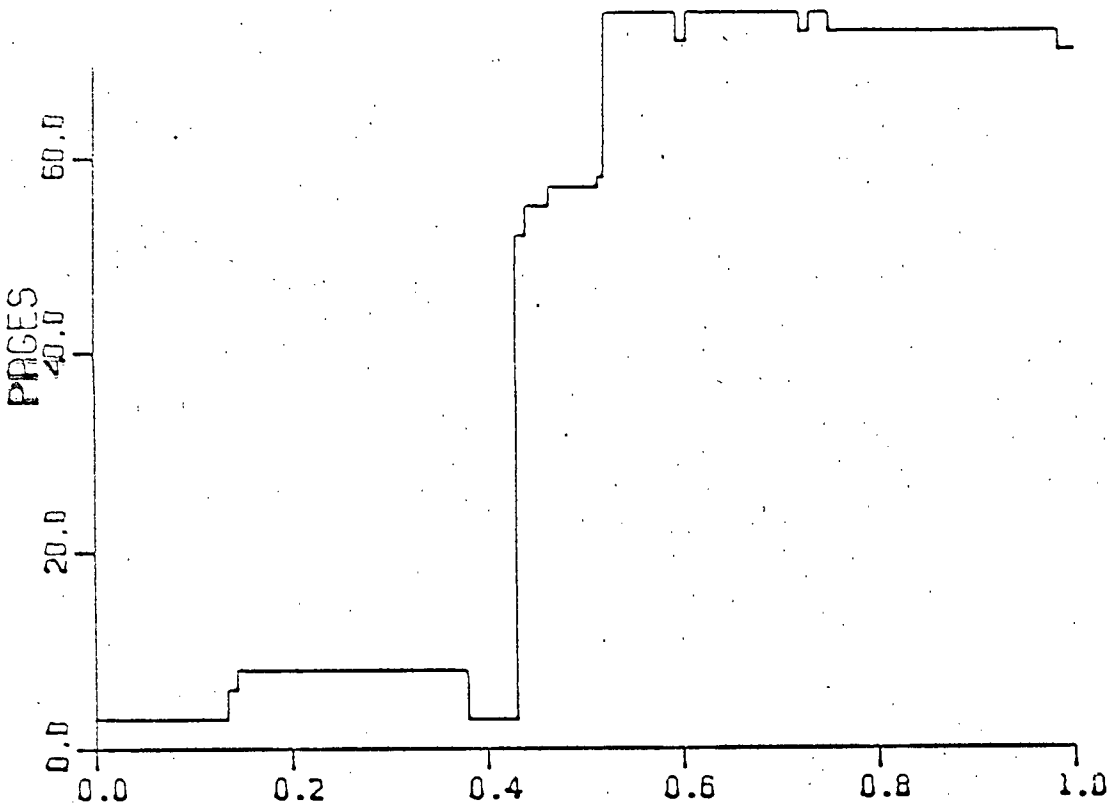


Figure D.21

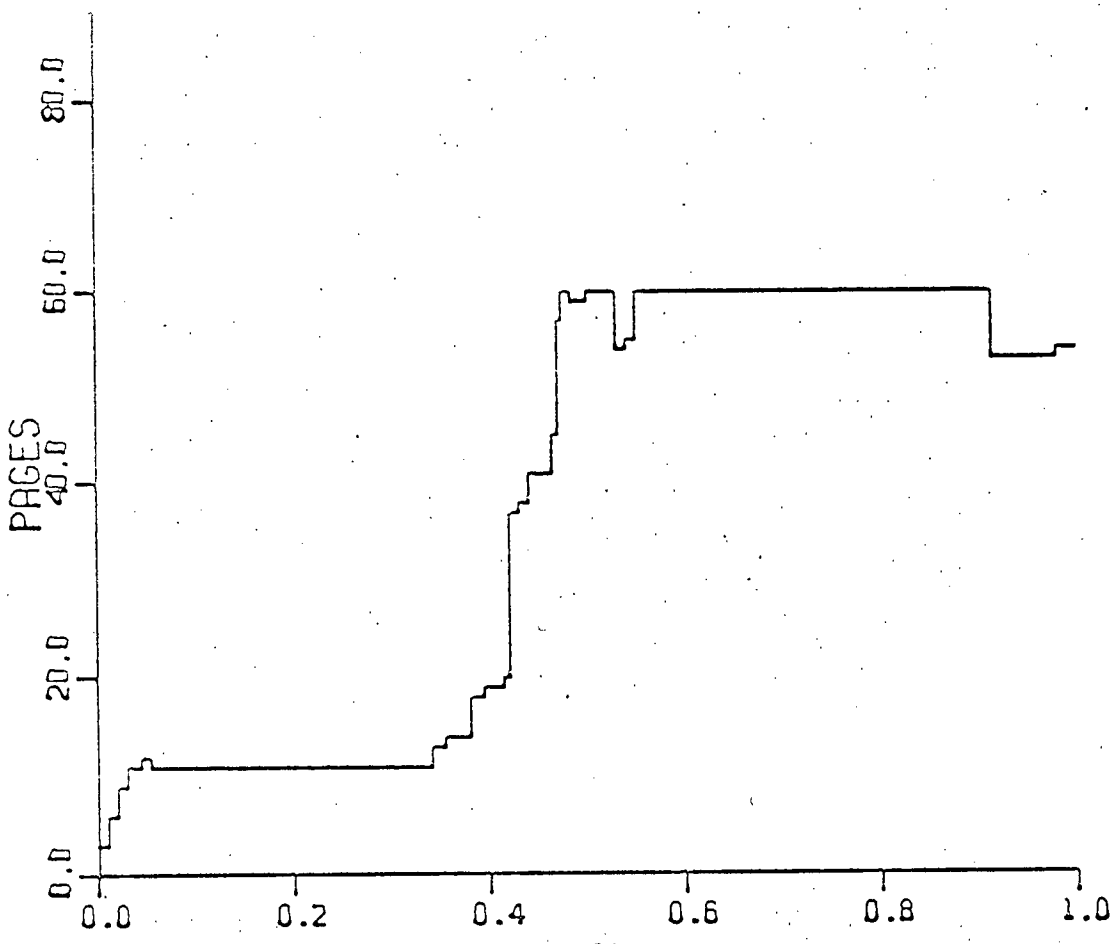


Figure D.22

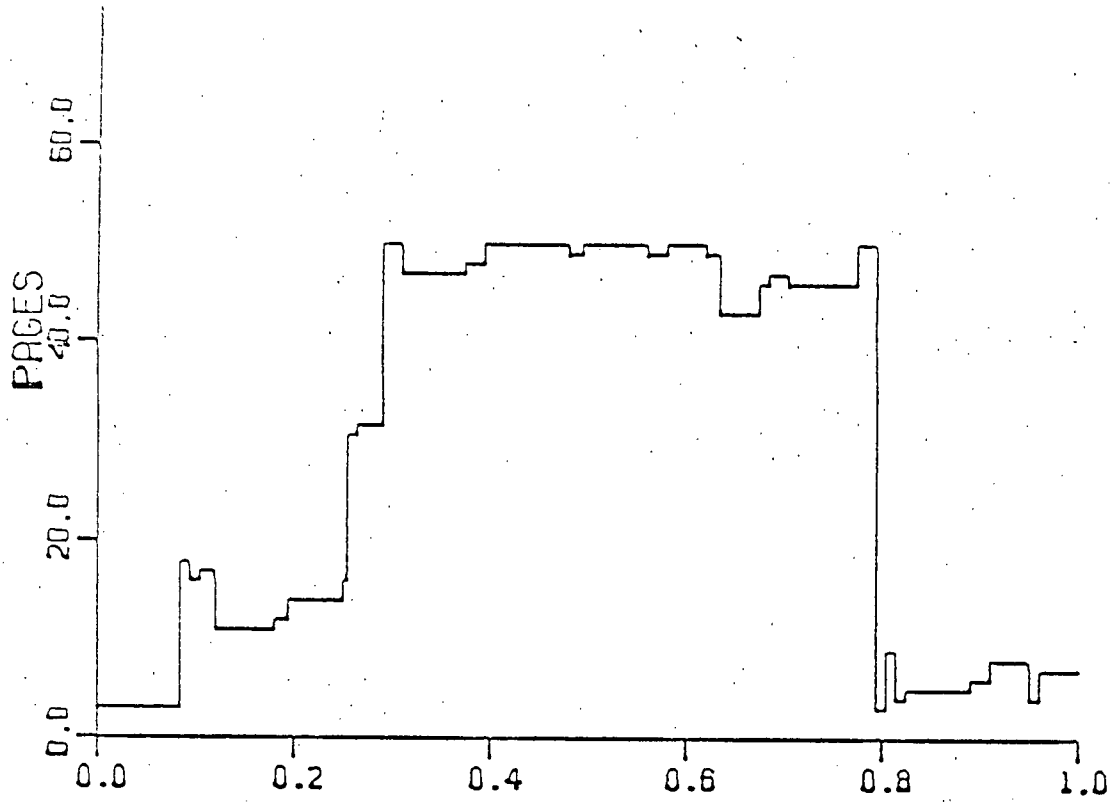


Figure D.23

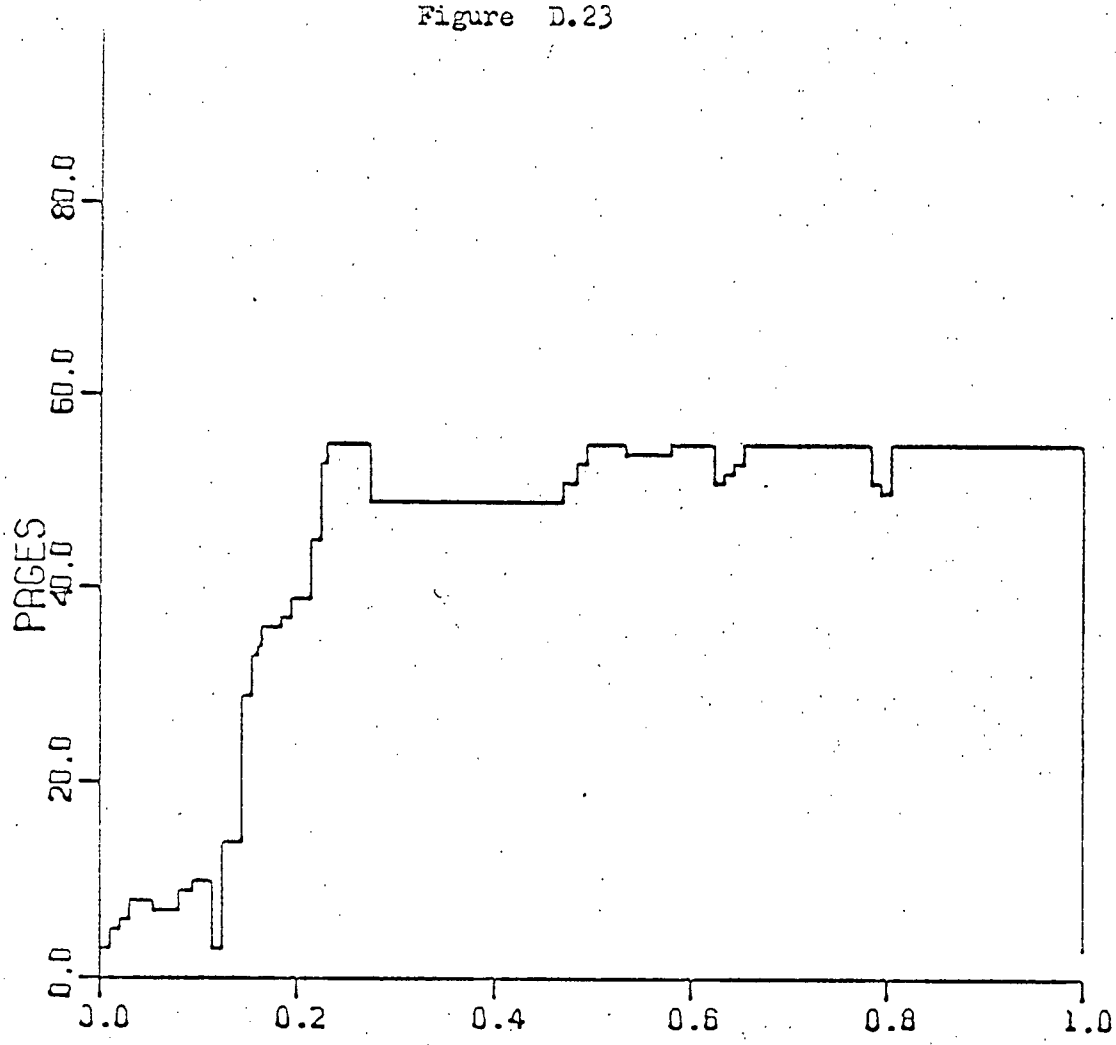


Figure D.24

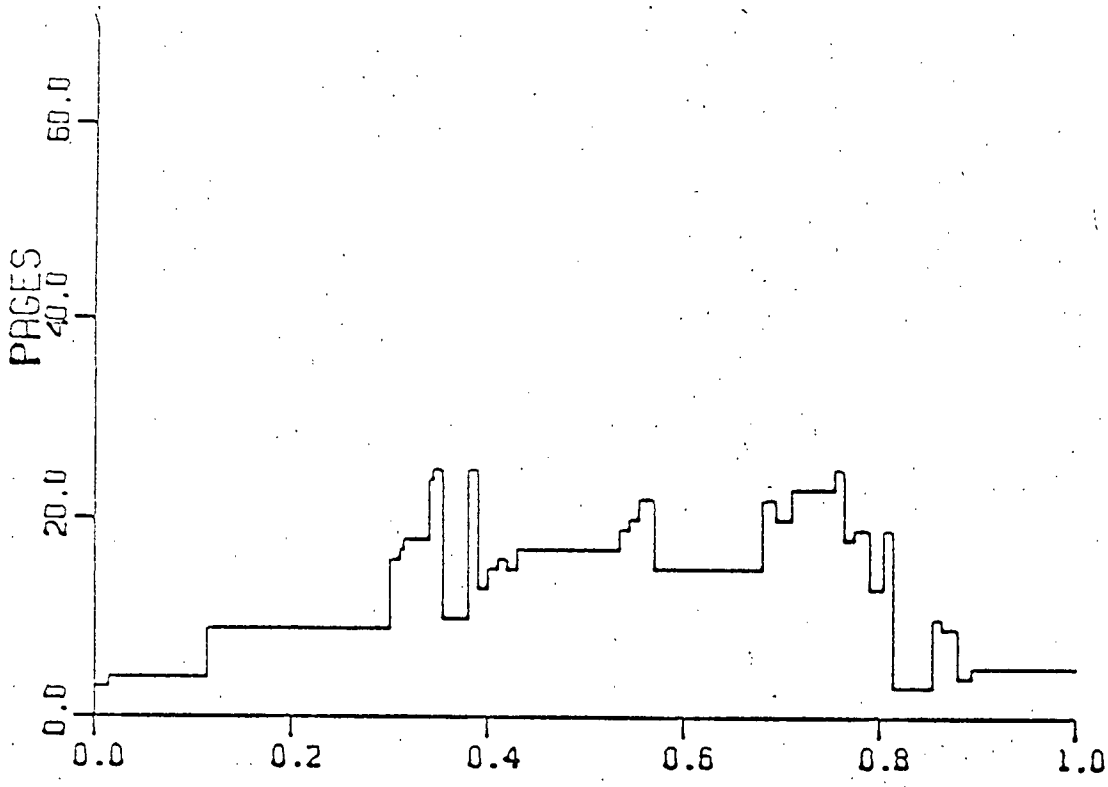


Figure D.25

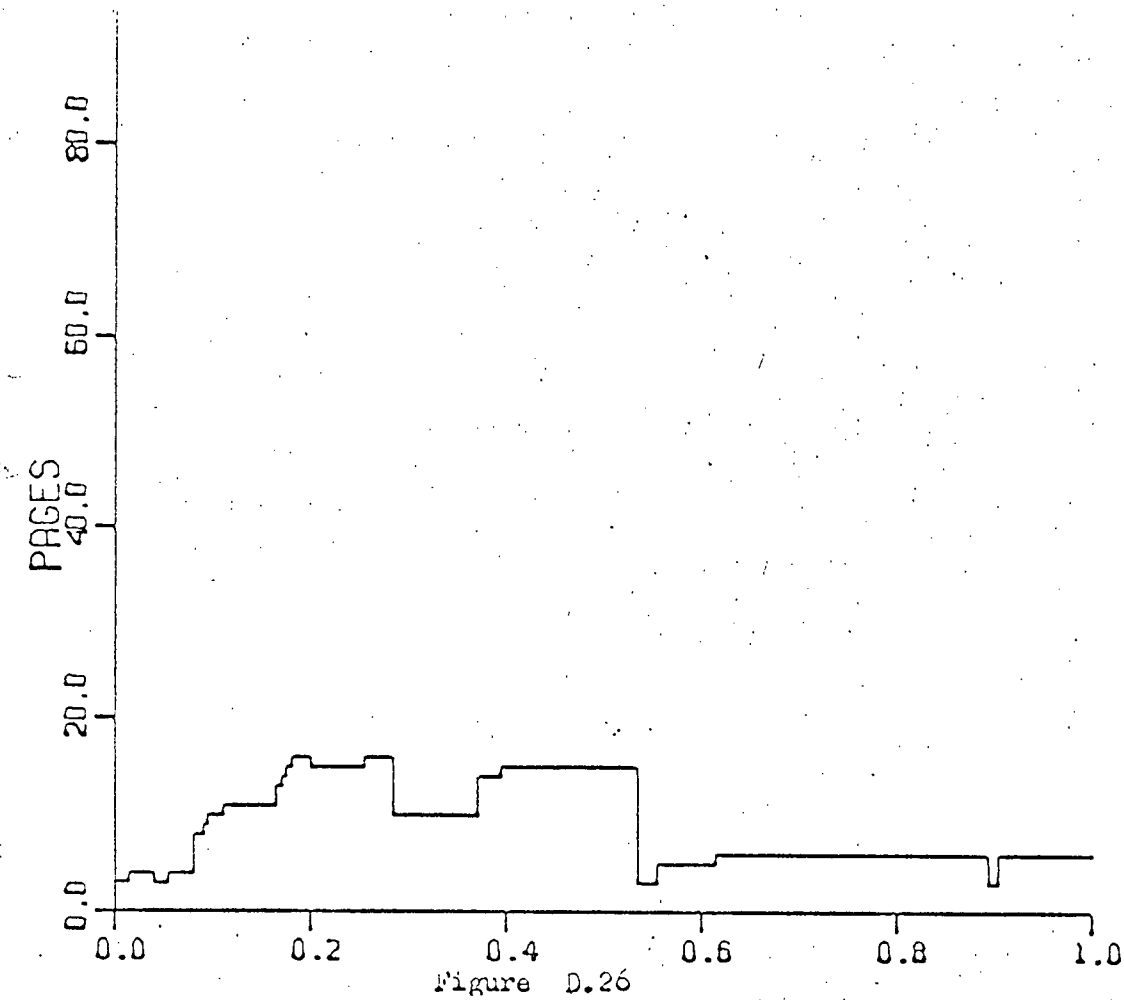


Figure D.26

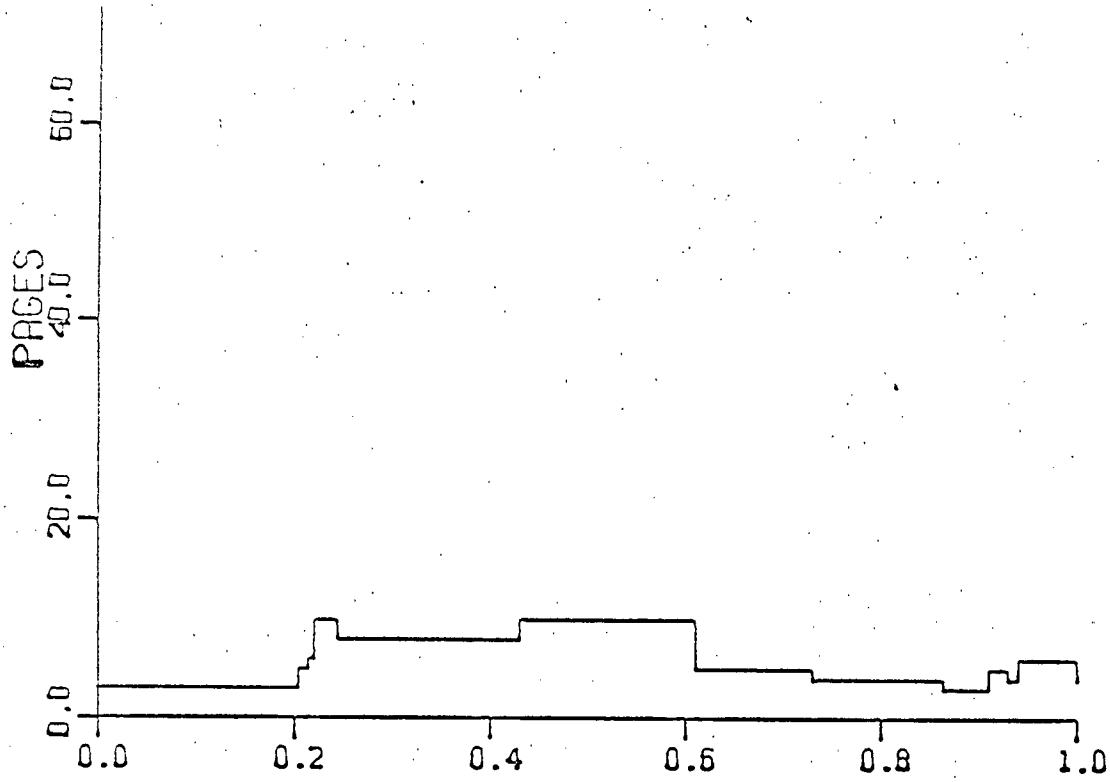


Figure D.27

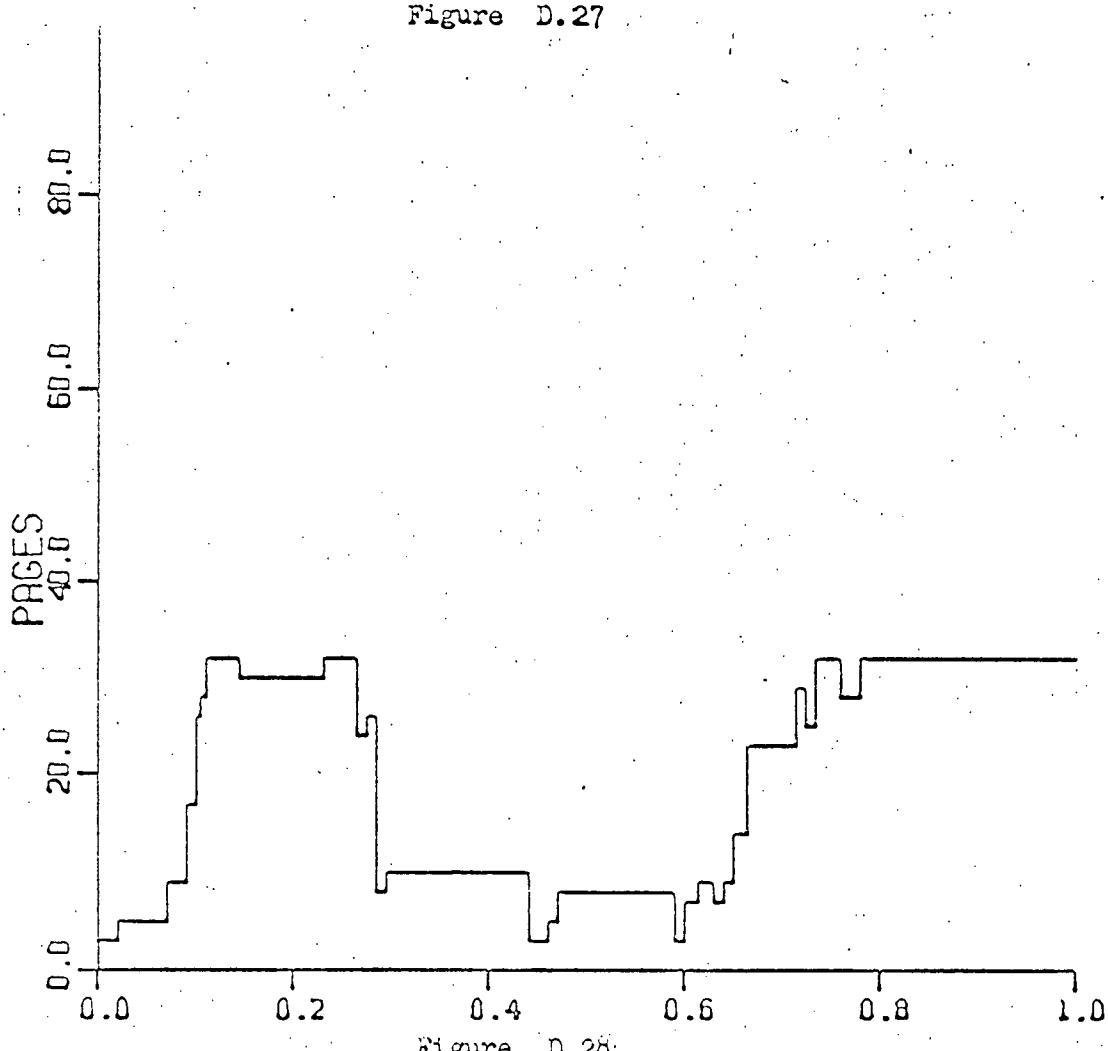


Figure D.28

APPENDIX E THE COMPUTING SYSTEM

The configuration of the 360/67 computing system at the University of British Columbia during the time of this study was as follows:

1) 2 2067 processors
2) 2 2301 drum storage units
3) 4 2365-2 core storage units (1024K bytes)
4) 2 2860-2 selector channels
5) 2 2870 multiplexor channels
6) 2 2314 8-drive disk units
7) 1 2321 data cell
8) assorted card readers, printers, graphics display units, interactive terminals, and the associated control equipment.

The two processors run in full duplex mode.