ABSTRACT

Simulation of a Paged Computer System

A Teaching Tool

This thesis describes the design and implementation of a simulator (written in IBM Fortran IV (Level G)) of a paged, multi-programming, single-processor, computer system.

A general justification of such a simulation is made, followed by details of the particular model chosen and implementation details.

Validation of the simulator is discussed, and followed by details of a number of experiment using various simulated job streams and configurations. Finally the response of a simulated system to two different paging algorithms is discussed and compared to known experimental data.

Finally, the use of the simulator as a teaching tool is described with details of the paging algorithm interface with the rest of the model.

SIMULATION OF A PAGED COMPUTER SYSTEM

A TEACHING TOOL

Th 8806

I hereby declare that the conditions of the Ordinance and Regulations for the degree of Master of Science (M.Sc) at the University of St. Andrews have been fulfilled by the candidate, Linda A. Macaulay.

[illegible] Morven Wilson

I hereby declare that this thesis is a record done by myself, not accepted in any previous applications for a higher degree in the University of St. Andrews or elsewhere.

(Mrs.) Linda A. Macaulay

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank my supervisor, J. MORVEN WILSON, for the exceptional guidance and practical assistance which he has given me.

I am also grateful for the generous co-operation given by PROFESSOR A.J. COLE and the kind help lent by the computer staff in his department.

Further, I wish to thank MRS. J. BROWN for the excellent and speedy manner in which she typed this thesis.

OBJECT

The object of the thesis is to illustrate the development and validation of a simulation of a paged computer system with a view to that simulation being used as a teaching tool.

The teaching tool takes the form of a computer program written in FORTRAN. Its objective is to help computer science students see the effects of paging algorithms (written by themselves) on various time-sharing system configurations and consequently to help them produce an effective algorithm.

CONTENTS

PART I

INTRODUCTION

I. 1. INTRODUCTION TO COMPUTER SYSTEM SIMULATION

a. WHAT IS SIMULATION?

Simulation is a technique for obtaining information about the performance of a system without actually putting that system into operation. A model of the system is constructed so that the results obtained by operating the model indicate the results to be expected when the corresponding real system is operated. The model may then be modified and operated to indicate the behaviour of the real system if it was also so changed.

Basically, the simulation technique is to create a model of the system by keeping lists of items at each stage in the process and transferring items from one list to another in the correct chronological order. The transferring of an item from one list to another usually represents a transition through some stage in a process and is accompanied by appropriate updating of a timing device.

b. WHY SIMULATION?

The thesis is concerned mainly with the investigation into the performance of a paged time-sharing system under a given set of conditions.

The two basic approaches that have been used for the investigation of existing time-sharing systems have utilized either analytic or simulation techniques.

In certain instances analytic techniques have proved quite satisfactory, for example, Scherr (1) was able to design a very simple model of the Project MAC system at MIT and Smith (2) was able to construct a model reflecting a paged time-sharing system. Analytic techniques, however, require a large number of simplifying approximations and assumptions whereas simulations require relatively few. This enables the simulation of more complex computer systems thus giving the method a great applicability.

In general, analytic models lack sufficient flexibility to allow a number of different systems or algorithms to be investigated without a great deal of extra effort. However as Nielsen has demonstrated in his 'Simulation of time-sharing systems' (3) simulations do exhibit the necessary degree of flexibility.

The disadvantages of simulation arise in the debugging of the simulation program and in deciding to what extent the simulation results are valid. The latter problem can be considerably eased if statistical measurements from a real system are available with which to compare the results from the simulation.

Paged computer systems are generally considered to be too complex and non-deterministic in nature for analytical methods of study. Thus the alternative chosen is simulation, validated by subsequent comparison with a real system.

c. SIMULATION THROUGHOUT THE GENERATIONS

The first generation of computers employed relatively simple hardware configurations which could be "investigated" without the use of a simulation model. The need for simulation developed with the advent of the second generation of machines when configurations became more complex. Simulations helped to give a general picture of overall system performance and provided an inexpensive and relatively easy way of investigating new design ideas.

Hardware performance was one of the first areas of the computer to which computer simulation techniques were applied. In 1957 W.E. Smith (4) developed a simulator for the internal logic of computer hardware components. This program was used for testing actual designs and also as a training device for designers. At a higher level, in 1964, M.S. Zucker (5) developed a simulator called LOCS (logic and control simulator) which simulated the components collectively, thus giving an overall view of the performance of the circuitry.

Simulations have been further developed to investigate the performance of the computer under various combinations of variables. One of the earlier programs of this type was published in 1964 by Statland (6) who considered such variables as equipment capabilities and I/O block sizes.

As software increased in importance it was realised that simulations were less likely to be reliable unless software and hardware-software interactions were taken into account.

Many simulations were constructed after the software system in question and were used to help evaluate proposed changes to that system, e.g. Katz (7) study of the IBM 7090/7040 Direct Coupled Operating System. Others were developed before the system in question had been built or programmed and were used in the construction of the system as well as subsequent modification, for example, IBM's 7090 time-sharing system.

With the appearance of the third generation of computer systems more comprehensive simulators were developed. For example, in 1967, Nielsen (3) published a simulation model with a general purpose design which can be used to study a variety of time-sharing systems. It can analyse performance characteristics for such varied purposes as hardware configuration, software modification and parameter adjustment, algorithm design and system development. In 1969 Seaman and Soucy (8) developed a Computer System Simulator (CSS) model package. CSS provides the user with a language and structure with which he can model a large variety of computer systems at differing levels of detail.

A recent test (9) (1970) was done on the validity of the simulation technique using the ATLAS computer at Manchester University. The operation of the ATLAS was simulated using a next-event type of simulation model and good agreement was found between the simulated results and those of the real system.

It can be seen that computer system simulations have developed hand in hand with the development of the computer, modelling proposed hardware and software architectures and configurations quickly and cheaply before any full scale commitment to their implementation. They continue to be a vital and valid tool in the investigations into the design of to-day's complex systems.

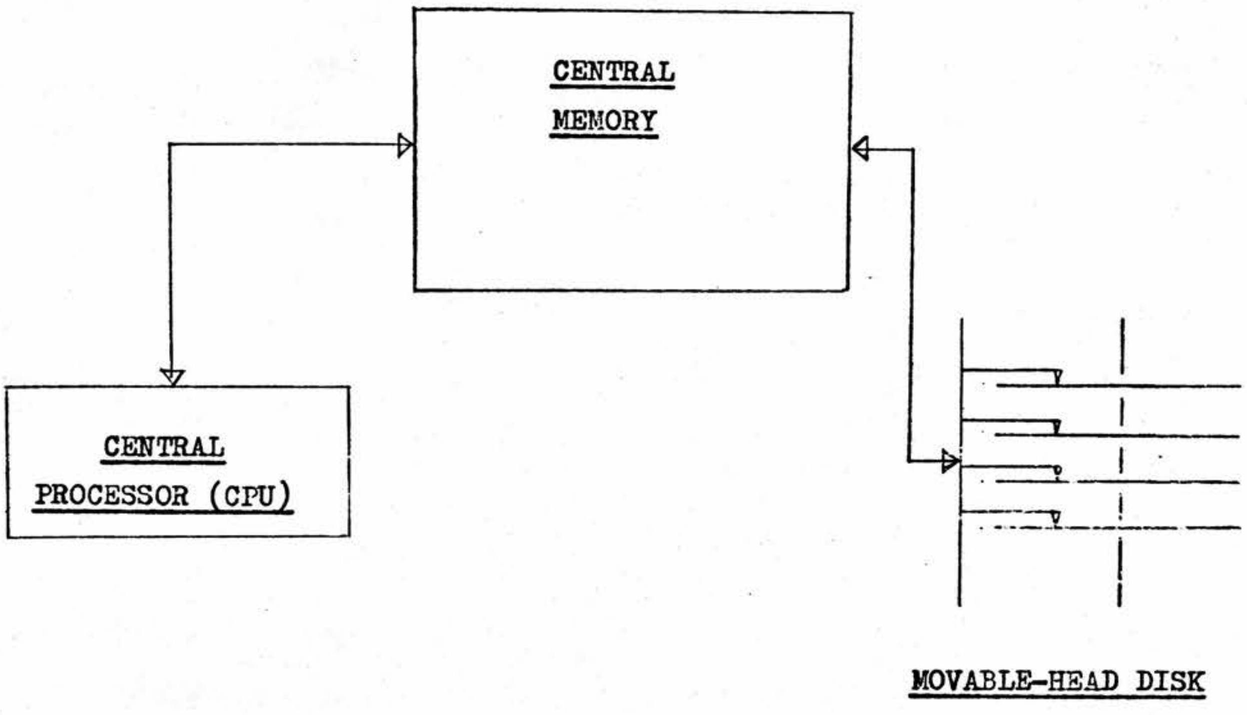
I. 2. THE CONSTRUCTION OF A BASIC SIMULATION MODEL - BASYS

BASYS, as formulated by Macdougall (10), helps to establish the basic notions of a simulation model and to illustrate how even a simple simulator can be used to see the effects of varying certain system parameters.

BASYS is a basic simulation model for a disk-based multiprogrammed computer system, whose configuration is shown in fig.1.

When a job arrives at the system it requests central memory space. If sufficient space is available then it is assigned to the job, otherwise the job is entered into a queue (the central memory queue) until enough space becomes available. Once it has been assigned central memory space (note that the entire program is in core) it can then begin execution. It requests the central processor. If the processor is free it is assigned to the job otherwise the job is entered in the central processor queue. When the job has been assigned the processor and starts executing it may issue I/O requests. At the point of issuing a request the job loses control of the processor and requests the use of the disk. If the disk is free it is assigned to the job otherwise the job is entered into the disk queue. Once the job has completed the I/O request it will probably require more CPU time. On regaining control of the CPU it may issue other I/O requests or continue executing until completion. On completion the job releases the processor, frees the central memory and leaves the system. Note that several jobs are in the system at the same time i.e. it is multiprogrammed.

FIG 1 HARDWARE CONFIGURATION OF BASYS



The simulation model

BASYS is a 'next-event' type simulator i.e. the simulated time clock is advanced to the time of the predicted next event. The events represent transition points between activities. Seven events are simulated (shown in fig.2) and four queues are dealt with, namely, the queue for central memory space, the queue for central processor attention, the queue for execution of drum transfers, and the event list. The job mix may either be read in directly by the simulator or generated within the simulator program using various known probability distributions. The flowchart of the simulator is shown in fig 2.

The Simulator Structure

The structure of the simulator is as shown in fig.3.

In the simulator, a job is represented by an entry in a job table. This entry contains characteristics established for the job as well as various counters for accumulating job related statistics. As the job moves through the system - enters queues, is assigned to the central processor, etc - its movement is reflected by moving a pointer to this job table entry, rather than by moving the entry itself.

The progress of the job through the system is marked by the occurrence of a series of events. Each event routine essentially does two things: it simulates the operations whose initiation corresponds to the occurrence of this event and it predicts, for the job for which the operation was performed, which event is to occur next and at what time it is to occur.

FIG 2 THE SIMULATION MODEL FOR BASYS

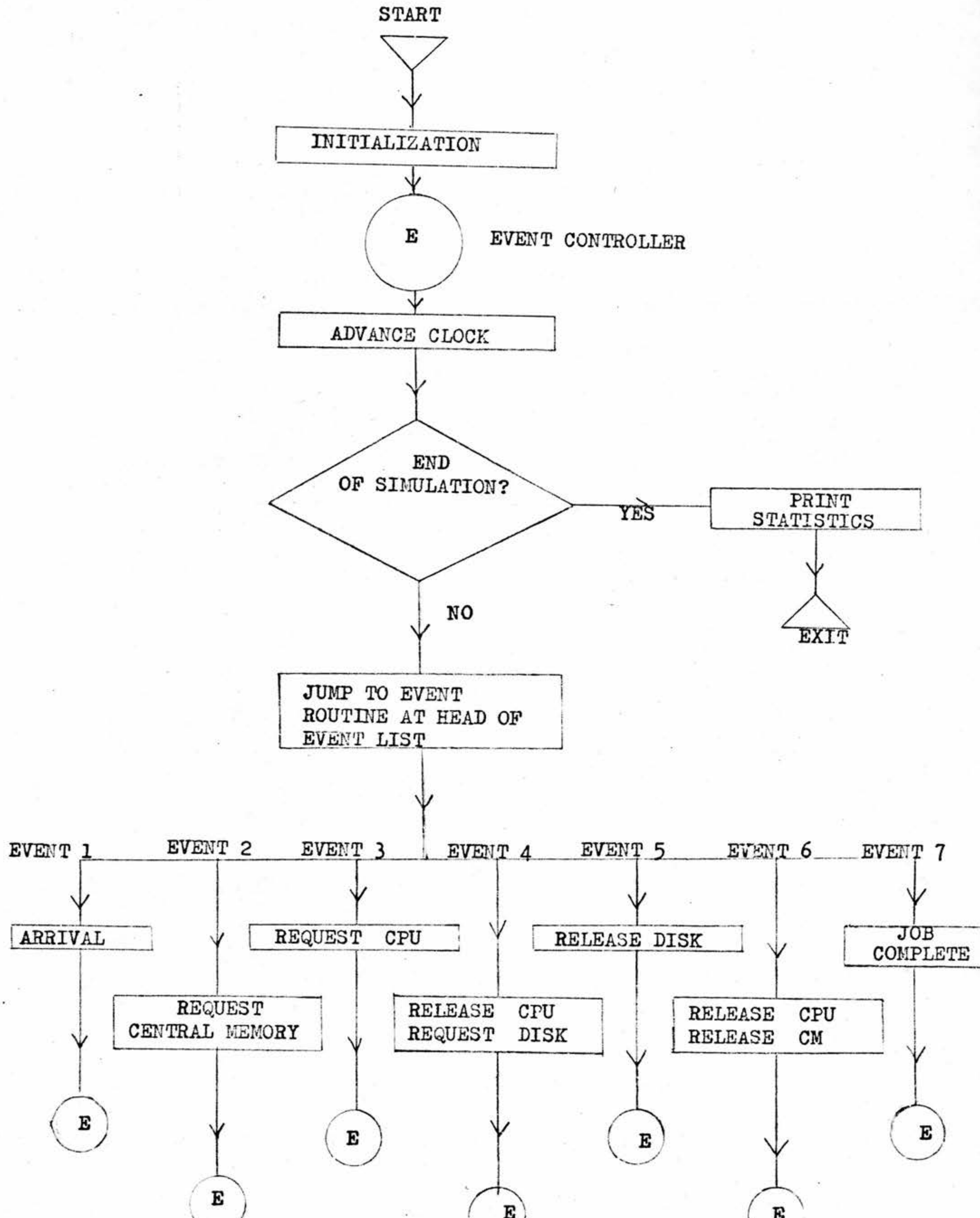
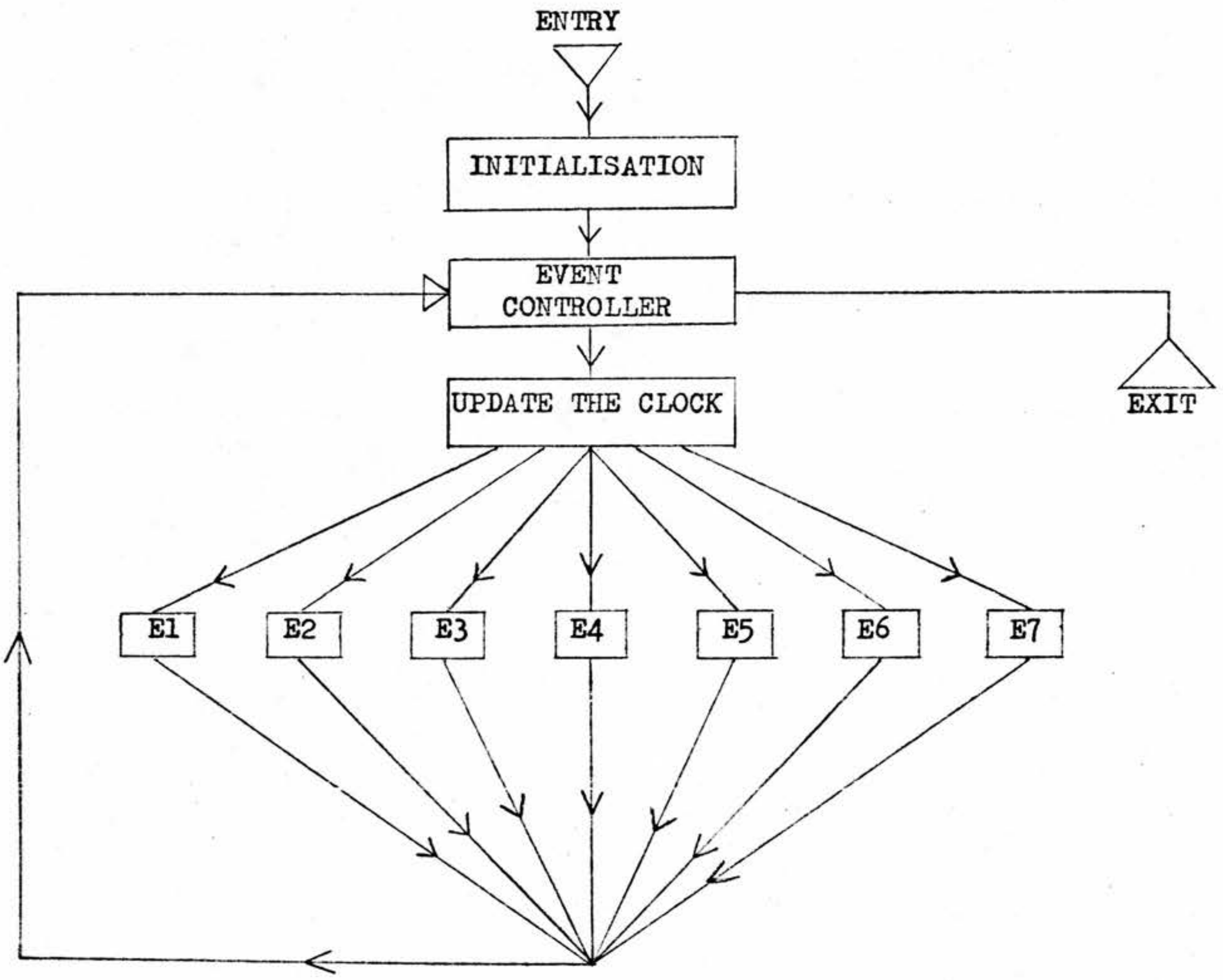


FIG 3 THE SIMULATOR STRUCTURE



E1 to E7 ARE THE EVENTS

An event list facilitates the ordering of events. This is a linked list ordered with respect to the clock time at which the next event is to occur. Thus the head of the list is the job whose next event is to occur at the earliest point in simulated (clock) time. A typical snapshot of the list at some point in time might appear as follows:

CLOCK TIME = 100

NEXT EVENT	EVENT TIME	JOB	
RELEASE DISK	101	20	← HEAD OF LIST
REQUEST CPU	109	18	
JOB ARRIVAL	117	22	

All event routines in the BASYS simulator make entries in the event list, but only one routine (the Event Controller) removes entries from this list. The structure of the event list is shown in fig. 4.

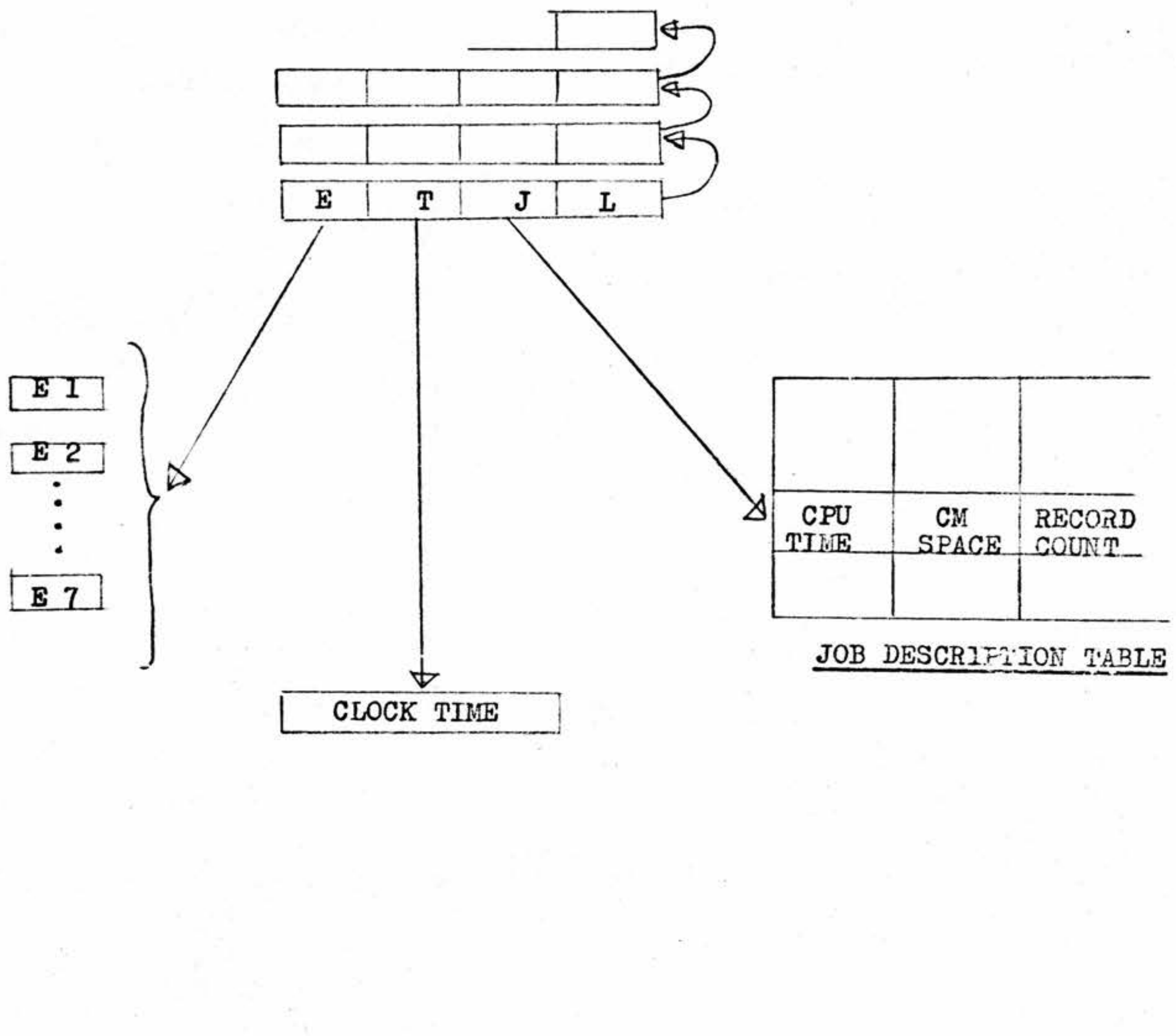
The event controller controls the occurrence of all events in the simulator and always transfers control to the event routine specified by the head of the event list. When the corresponding event has been completed and the job returned to the event list or entered into a queue, then control is always returned to the event controller.

The basic steps in the event scheduling are:

1) The event controller removes the entry at the head of the event list. This entry specifies an event time T, an event identifier E, and a job table pointer J.

2) The clock is updated to time T.

FIG 4 THE EVENT LIST



KEY

E	–	EVENT IDENTIFIER
T	–	THE PREDICTED TIME AT WHICH EVENT E IS TO OCCUR
J	–	THE JOB TABLE ENTRY POINTER
L	–	LINK TO NEXT ENTRY IN EVENT LIST

3) The event controller transfers control to the event routine E.

4) The event routine E performs the required processing for the job, and if possible determines its next event (and inserts the event identifier E^1, event time T^1 and a job table pointer J into the event list). If such a determination is not possible the event routine E enters the job into a queue.

5) Control is then returned to the event controller.

If, as is sometimes the case, the next event for the job cannot be predicted, no entry for the job can appear in the event list. This situation arises when the job has to be entered into a queue, for example, when the disk is busy or when there is a shortage of central memory space. Once the facility becomes available, and the job reaches the head of the queue, an entry is inserted into the event list to make the 'next event' a request for the facility which was previously unavailable, for example, request disk, request central memory space etc.

All queues in BASYS are represented in the form of linked lists. BASYS lends itself to a straightforward implementation in GPSS and SIMSCRIPT, and with the addition of a few elementary list processing routines can be effectively implemented in FORTRAN.

The main advantage of BASYS is that it allows for extensions and additions to the basic model. Thus it can be used as a basis for more extensive simulations of computer systems with greater complexity.

EXPERIMENTS AND RESULTS from implementation of BASYS in FORTRAN. Two main experiments were carried out. Both were to measure the utilization of the central processor in respect of

1) increasing the amount of central memory available

2) replacing the disk with a faster model

Results are shown in figs 5A and 5B.

CONCLUSIONS

Experiments show that after a certain point increasing the amount of core memory available has no effect on the percentage usage of the CPU. This point is reached when the total core memory requirements of all the jobs can be satisfied simultaneously. Results further show that increasing the disk speed by a factor of two gives a proportional increase in the usage of the CPU. This can be seen by comparison of the curves given in fig.5A.

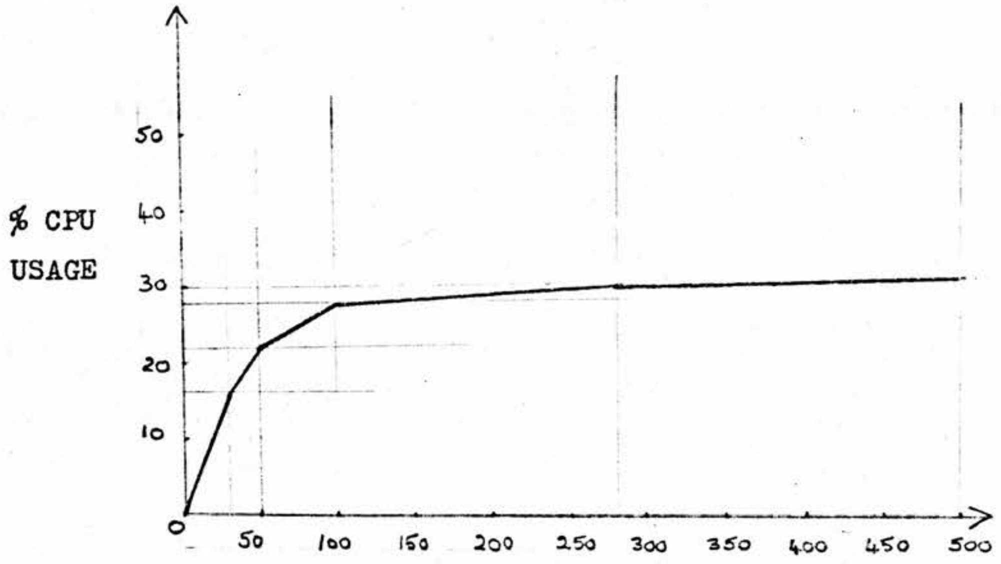
The BASYS simulation model has thus shown how even a simple model can be used to investigate the effects of varying certain parameters on the simulated computer system.

From the point of view of the thesis the implementation of BASYS was an exercise to aid familiarization with the principles of a simple computer system and some basic simulation techniques. The BASYS model is still available as a teaching tool if required, but the thesis now progresses to the more challenging problem of simulating a paged multiprogramming computer system.

FIG 5A RESULTS FROM BASYS EXPERIMENTS

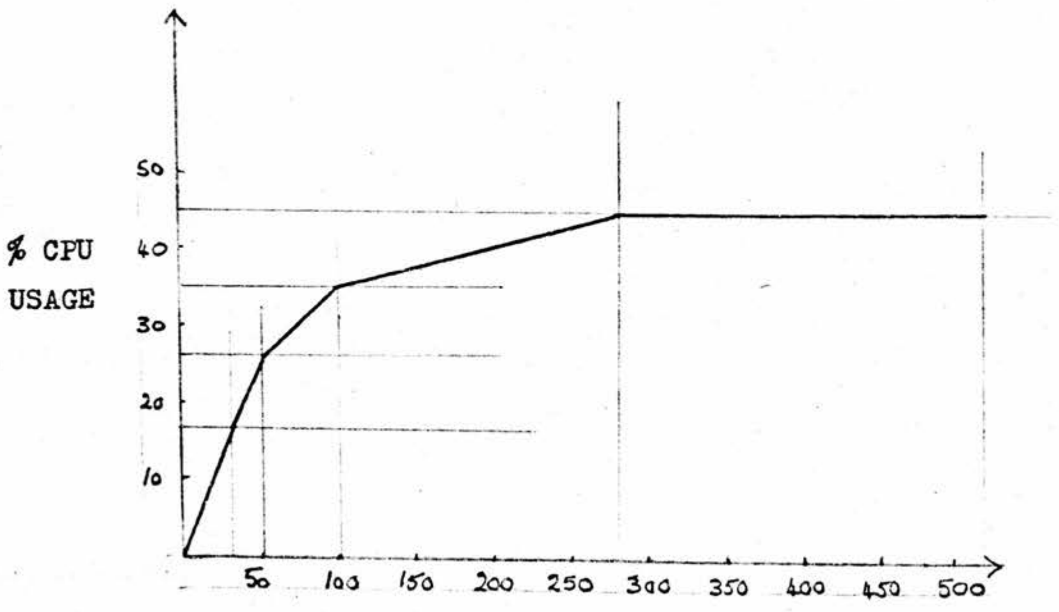
GENERAL TRENDS FROM RESULTS

ONE DISK REV. IN 25 MILLISECS



CORE SPACE AVAILABLE IN K-BYTES

ONE DISK REV. IN 12 MILLISECS



CORE SPACE AVAILABLE IN K-BYTES

THE MAXIMUM CORE SPACE REQUIRED BY THE 20 JOBS
WAS 356 K-BYTES

FIG 5B RESULTS FROM BASYS EXPERIMENTS

K SIZE	% CPU
25	16
50	26
100	34
275	44
510	44

RVTIM = 12

K SIZE	% CPU
25	14
50	21
100	28
275	30
510	30

RVTIM = 25

RVTIM	-	TIME TAKE FOR ONE DRUM REVOLUTION
KSIZE	-	SIZE OF CORE MEMORY IN K-BYTES, $K = 2^{10}$
% CPU	-	% CPU USAGE

Figure 5C

JOB DESCRIPTION OF 20 JOBS RUN THROUGH BASYS

Job No.	Central Memory Space Required	CPU Time Required	Number of I/O Requests	Mean Inter-Request Interval	Record Size
1	7520	50	10	5	360
2	6480	30	10	3	360
3	10100	60	20	3	360
4	15210	70	10	7	360
5	6110	30	10	3	360
6	20360	100	20	5	360
7	12220	80	20	4	360
8	4140	10	10	1	360
9	17770	70	10	7	360
10	9080	30	10	3	360
11	20200	60	10	6	360
12	10000	40	10	4	360
13	12100	50	10	5	360
14	8800	30	10	3	360
15	6070	10	10	1	360
16	51610	140	20	7	360
17	72130	200	40	5	360
18	7120	30	10	3	360
19	31010	90	10	9	360
20	36700	110	20	5	360

Inter-Arrival Time of Jobs

All jobs arrived at the system at fixed equally spaced intervals.

PART II

THE SIMULATOR

II 1. DESCRIPTION OF THE SYSTEM TO BE SIMULATED

a. GENERAL FEATURES OF THE SYSTEM

The system under investigation consists of a paging memory, paging auxiliary storage (drums) and one central processing unit. The complete configuration is shown in fig.6.

Basically, the simulator implements a time-sharing, multiprogramming system with provision for logical-to-physical address mapping by either simple paging or demand paging. It has several special features which are discussed in some detail below.

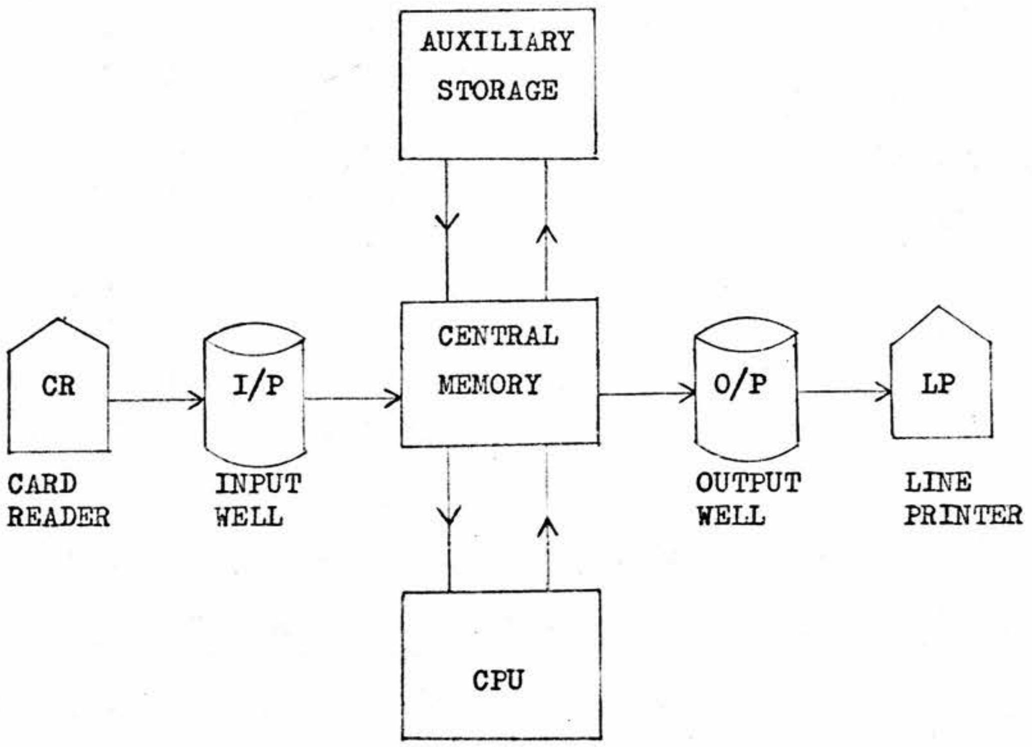
These are:- 1. Paging and demand paging
2. Working Set strategy
3. Ready List

1. PAGING AND DEMAND PAGING

The object of the study is to produce a simulator which will be sensitive to changes in the paging algorithm; thus it is essential that a paging core memory and a paging drum are included in the system.

Systems which incorporate paging are troublesome from a simulation point of view (as Boote et al. remark in their simulation (9))since the page turning events take place much more frequently than program-swapping events on a non-paged computer. The real time necessary to complete a simulation is therefore longer. Scherr's simulation (1) of a non-paged IBM 7094 had a simulated-time to real-time ratio of

FIG 6 THE SYSTEM



approximately 24, whereas Nielsen's simulation (3) of the paged IBM 360/67 on an IBM 360/50 had a ratio close to 2.

APPROACH TO PAGING

Paging is a set of techniques whereby programs and main memory are broken into small units and the program pieces are located in corresponding sized blocks anywhere in main memory. The paging techniques incorporated in our system allow a straightforward implementation of a logical-address space larger than the physical-address space.

In our paged system, physical memory is considered to be broken up into "blocks" of a fixed size. The term "page" refers to units of logical space, while equal-sized units of physical space are called blocks. The programs are also considered to be split into "pages" of a size equal to the block size of physical memory. Thus the address is such a system is considered to be represented by two numbers:

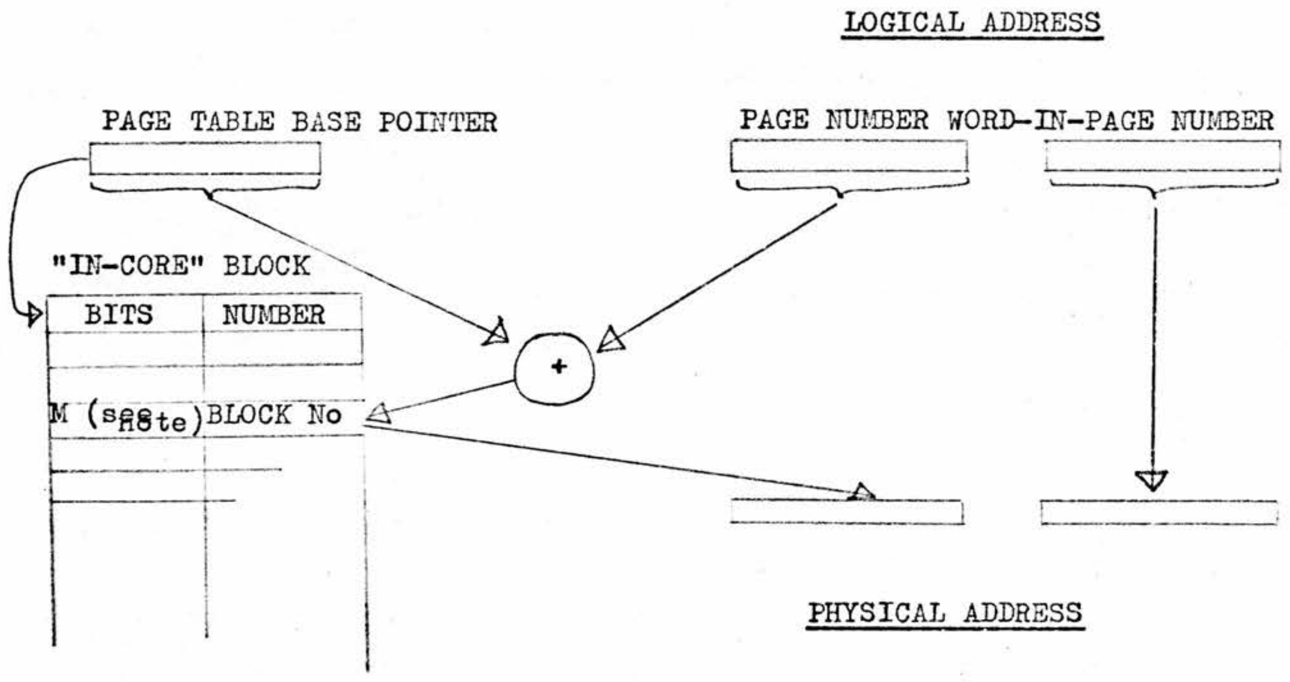
(1) a page address or number

and (2) a word-within-page address.

A paging mechanism requires a table, called a page-table, or map with one entry for each page in order to perform address translation from logical to physical space. The complete memory map used in our system is shown in fig.8 and the page table in fig.7.

One page table exists for each process. The physical-block number corresponding to a given page is found by a table look-up

FIG 7 A PAGE TABLE

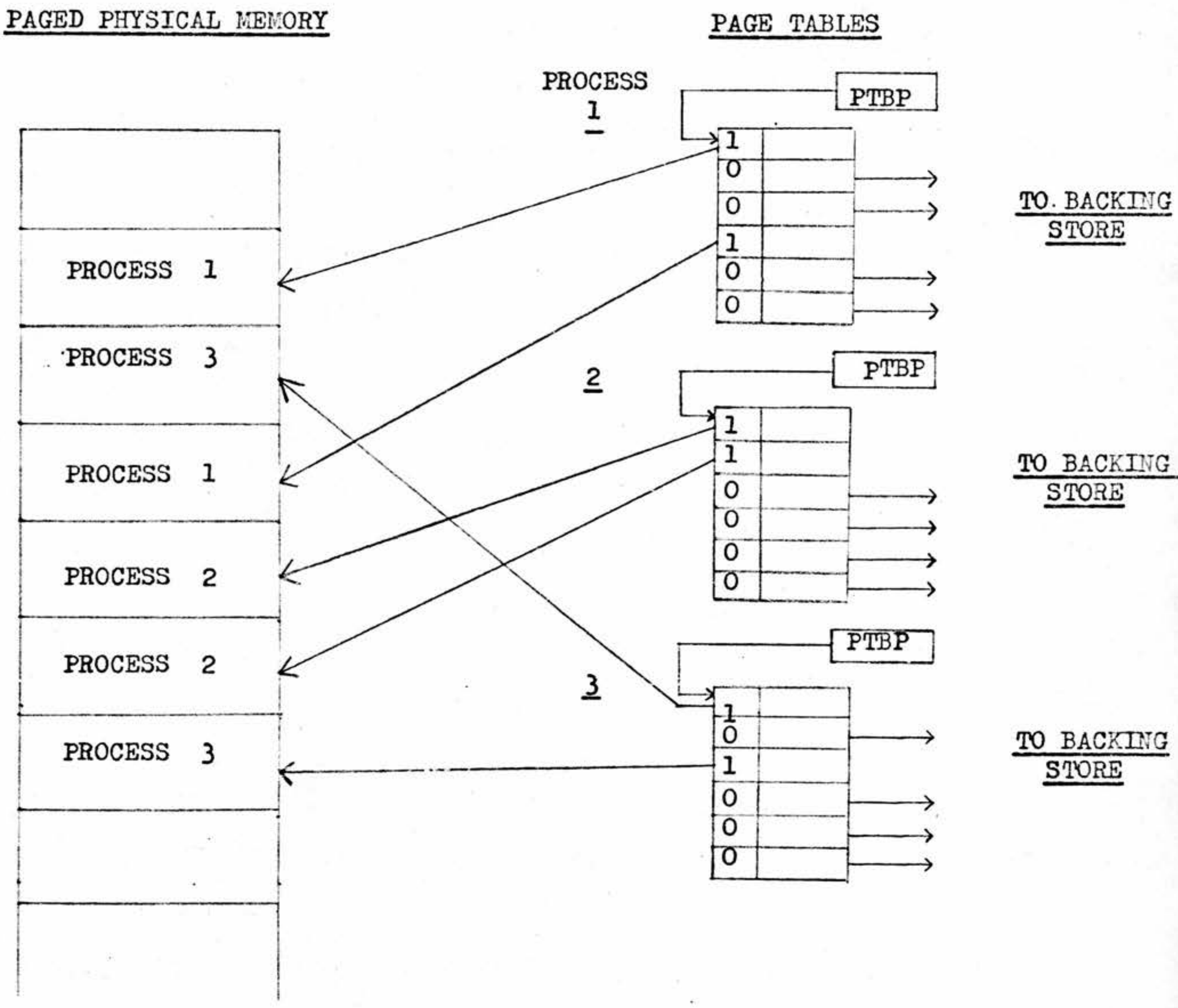


note

IF M=1 THE PAGE IS IN CORE: THUS BLOCK No ENTRY GIVES ACTUAL PHYSICAL ADDRESS OF THE PAGE.

IF M=0 THE PAGE IS NOT IN CORE: THUS BLOCK No GIVES ADDRESS ON AUXILIARY STORAGE. THE PAGE MUST BE BROUGHT INTO CORE BEFORE THE PHYSICAL ADDRESS CAN BE CALCULATED

FIG 8 THE MEMORY MAP



PTBP - PAGE TABLE BASE POINTER

in this page table using the page number as index. The control bits ("in-core" bits) in each table entry are used to indicate whether the page represented by that entry resides in memory or on an auxiliary storage device. The page number from the logical address when added to the contents of the page-table base register indicates which word in the page table contains the block number where the page resides. The figure in the block number portion of the table indicates an actual starting address for the page in main memory or a location on auxiliary storage where the page can be found. If the control bits indicate that the latter case holds, a call to the system, referred to as a page fault, is generated to fetch the page to memory before resuming computation. Using this approach, the logical-address space can be smaller, equal to, or larger than the physical-address space.

At the start of computation only a single "starter" page is loaded into main memory, not the entire process. Then as references are made to pages not currently in main memory, the page-table would indicate the fact by generating a page fault which causes the supervisor, to bring in the page. This approach is known as demand paging.

Thus paging and demand paging are incorporated in the system by means of the page table and memory map. It should be stressed, however, that no attempt has been made to introduce the segmentation concept into the system.

2. THE WORKING-SET STRATEGY

Several recent studies on the behaviour of programs in a paging environment (11,12,13,14,15) lead to the conclusion that over short periods of time instruction and operand references are confined to a subset of the set of pages comprising the logical address space, and that once this subset is established its content varies only slowly. Thus it seemed desirable to include some method by which information about the behaviour of programs could be made available to the students paging algorithm.

Denning (11) defines this subset of pages as the 'working-set'. He shows how the working set can be detected and suggests an algorithm which makes use of this information. Our simulation model detects the working set but the paging algorithm supplied by the user may use or ignore the information gathered. (see 'The Simulation Model').

Description of the Working Set

The working set of information is the smallest collection of information that must be present in main memory, at any instant, to ensure efficient execution of a program.

The operating system is responsible for determining on the basis of page reference patterns, which pages constitute the working set at any instant and for detecting those that leave the working set. In practice the operating system considers the working set of information, associated with a process, to be the set of most recently referenced pages within some arbitrary period of time.

To initiate a process on the processor a "starter page" is loaded and subsequent pages are demanded until the working set of pages is built up. When a page has not been referenced for a measured period (see later) then it leaves the working set and may be rolled out of core.

Formal definition of the Working Set

The working set $W(t,\tau)$ of a process at time t is the collection of information referenced by the process during the process time interval $(t-\tau,t)$.

Thus the information a process has referenced during the last τ seconds of its execution constitutes its working set, see fig.9.

The working set consists of information referenced during the last τ seconds; however, in our system we are usually interested in the pages which contain this information. Thus the pages themselves constitute our measure of the working set since the information required is only accessible in page sized blocks.

Properties of the Working Set

1. size

The size of the working set $\omega(t,\tau)$ is the number of pages referenced in this interval

i.e. $\omega(t,\tau)$ = number of pages in $W(t,\tau)$.

On consideration of the working set size it is obvious that in an interval of zero length, no pages will have been referenced. It is further clear that in longer intervals of time more pages will be referenced. Thus the general curve suggested by $\omega(t,\tau)$

is monotonically increasing as shown in fig.10. (See (11) for further details)

2. prediction

We would expect intuitively that the immediate past page reference behaviour of a program constitutes a good prediction of its immediate future page reference behaviour. That is to say, that for small time separations α , the set $W(t,\tau)$ is a good predictor for the set $W(t+\alpha,\tau)$.

To see this more clearly, suppose $\alpha<\tau$.

Then $W(t+\alpha,\tau) = W(t,\tau-\alpha) \cup W(t+\alpha,\alpha)$. Since references to the same page tend to cluster in time, the probability

$Pr\ (\ W(t+\alpha,\alpha) \cap W(t,\tau)\ )$ tends to be small.

Therefore some pages of $W(t,\tau)$ will still be in use after time t i.e. pages in $W(t+\alpha,\alpha)$, since also

$$W\ (t,\tau-\alpha) \subseteq W(t,\tau) \cap W(t+\alpha,\tau)$$

$W(t,\tau)$ is a good predictor for $W(t+\alpha,\tau)$.

On the other hand, for large time separations α (say $\alpha >> r$) control will have passed through a great many program modules during the interval $(t,t+\alpha)$, and $W(t,\tau)$ is not a good predictor for $W(t+\alpha,\tau)$.

3. τ -sensitivity and re-entry rate

It can be seen from fig.10, that as τ is reduced, $\omega(t,\tau)$ decreases. If the number of pages in $W(t,\tau)$ decreases, the probability that there are useful pages still in $W(t,\tau)$ also decreases. Consequently the rate at which pages are recalled to $W(t,\tau)$ increases. This means that if τ is decreased then the re-entry rate of pages will increase.

FIG 9 THE WORKING SET OF INFORMATION

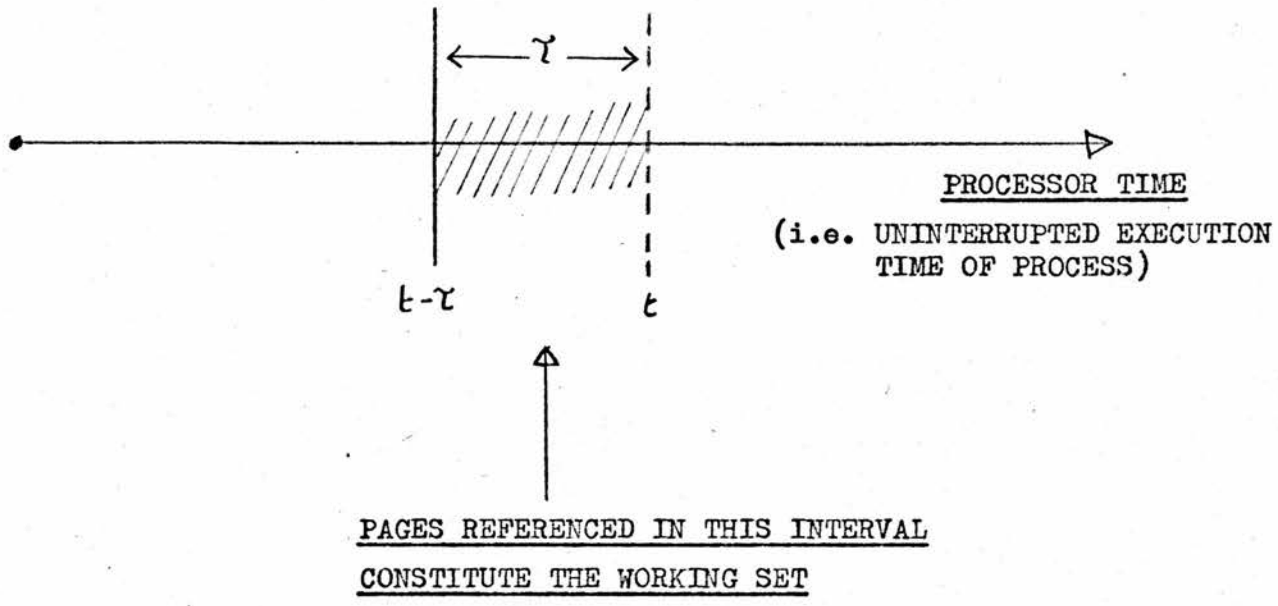
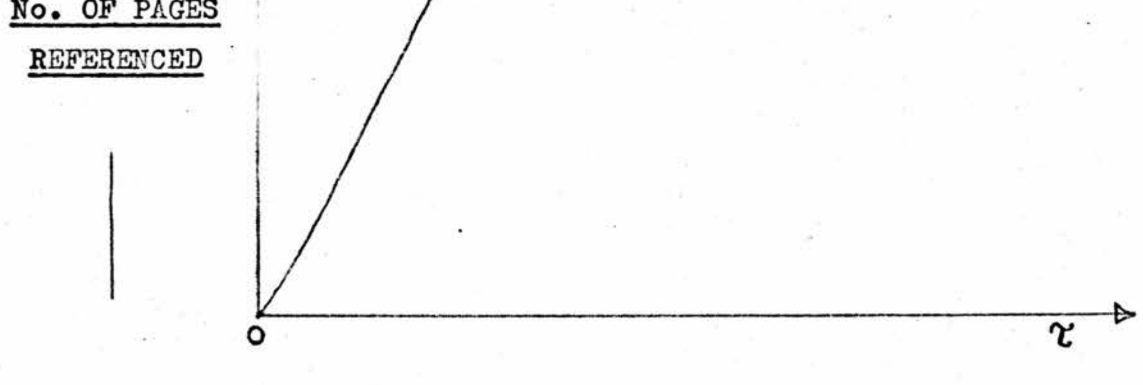

τ
PROCESSOR TIME
(i.e. UNINTERRUPTED EXECUTION TIME OF PROCESS)
$t-\tau$
t
PAGES REFERENCED IN THIS INTERVAL CONSTITUTE THE WORKING SET

FIG 10 VARIATION OF $\omega(t,\tau)$ WITH CHANGES IN τ



The value ultimately selected for τ will thus be of great importance to the effectiveness of using the working set strategy. Should τ be too small, pages may be removed from main memory while still useful, resulting in a high traffic of returning pages. Should τ be too large, pages may remain in main memory long after they were last referenced, resulting in wasted memory. Thus τ must be carefully chosen to strike a balance between excess page traffic and too much wasted memory.

In the system which we are simulating, a page is not rolled out of main memory immediately it leaves the working set. Instead it is 'marked' as a candidate for removal from core and will only be removed if the space it occupies has been demanded by a page of some other process. (Should the paging algorithm so decide). Note however that in the case of a read-only page (where a valid copy already exists on backing store) the page is still marked as a candidate for removal. Thus the space it occupies may be taken over by some other process in the usual way, however the page is not rolled out to backing store. The page table entries and other relevant information about the page is simply updated.

<u>Detection of the Working Set</u>

Detection procedures similar to those suggested by Denning (11) are implemented in the system. As detailed above, each process has a related page table which provides a map from the logical address space to the physical address space of each page belonging to that process. Along with the 'in-core' bits and the 'block number' entry there is a further entry which contains a string of 'use-bits' U_0, U_1, ..., U_K. (see fig.11)

The sampling interval σ is defined to be τ/K where K is an integer constant chosen to make the sampling intervals as 'fine grain' as desired. On the basis of page references during each of the last K sampling intervals, the working set $W(t,K\sigma)$ can be determined.

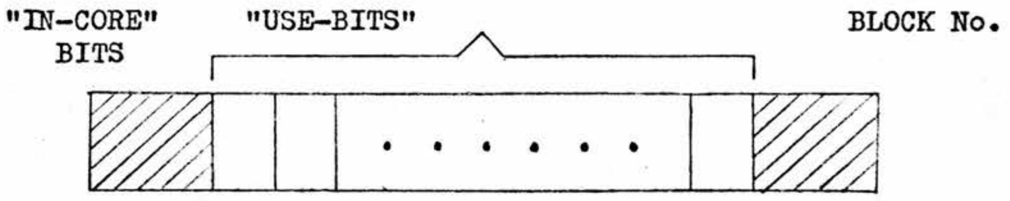
Each time a page reference occurs, U_0 is set to 1 (whether 0 or 1 already). At the end of each sampling interval o the bit pattern contained in U_0, U_1 ...., U_K is shifted one position to the right, a 0 enters U_0, and U_K is discarded: see fig.12.

The logical sum U of the use-bits is computed:

$$U = U_0 \cup U_1 \cup U_2 \ldots \cup U_K$$

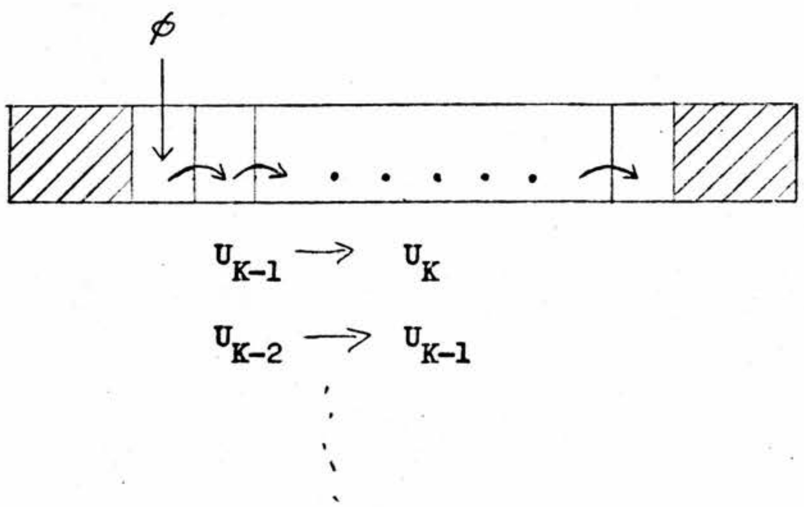
so that U = 1 if and only if the page has been referenced during the last K sampling intervals. Of all the pages associated with a process, those with U = 1 constitute the working set $W(t,K\sigma)$. If U = 0 and M = 1 (i.e. the page is in core), then the page is no longer in the working set and is marked as a candidate for removal from main memory.

FIG 11 TYPICAL PAGE TABLE ENTRY



(K IS THE No. OF SAMPLING INTERVALS)

FIG 12 SHIFT AT END OF SAMPLING INTERVAL



$U_{K-1} \longrightarrow U_K$

$U_{K-2} \longrightarrow U_{K-1}$

$\vdots$

$U_0 \longrightarrow U_1$

$\phi \longrightarrow U_0$

3. The Ready List

The Ready List is a list of processes ready to run on the central processor when it becomes available.

The Ready List has two quantum levels, a short quantum level (SQL) and a long quantum level (LQL) (see fig.13). A process is always allowed to run for a short quantum, and if at the end of this time no other process is ready to run, it can continue. The purpose of the short quantum is to assure that some useful computation takes place, in order to justify the expense of swapping the process in. This scheme also allows higher priority processes to pre-empt the processor if they appear on the SQL during or after a short quantum. When a process is dismissed after a short quantum or because a higher priority process has become ready, it is placed on the short-quantum level.

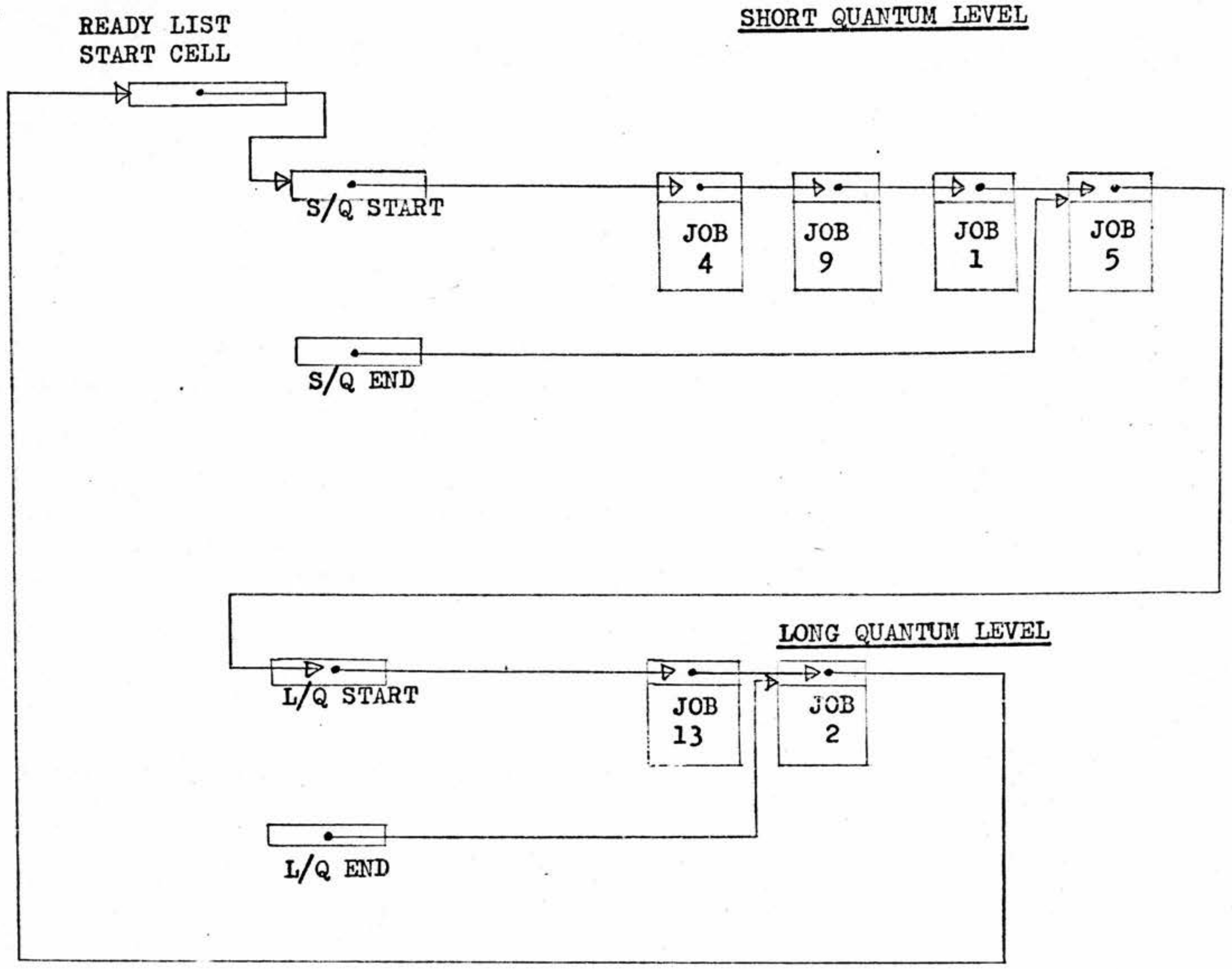
Each time a process completes a short quantum, a number, called the long quantum count is decremented. Once this count is reduced to zero the process is moved to the lowest-priority level, the long quantum level.

This method ensures that all processes will run with a reasonable response to each and it limits the number of times a process can appear on the high-priority level of the ready list.

PRE-EMPTION

A higher priority job may arrive on the ready list while a lower priority job has control of the processor. At that point pre-emption occurs and the pre-empted job is returned to

FIG 13 THE READY LIST



the short quantum level of the ready list. However this pre-empted job is not assigned a fresh short quantum but on regaining control of the processor will complete the remainder of its previously assigned quantum.

I/O REQUESTS

A similar situation arises when an executing job issues an I/O request. A record is kept of the time quantum still to be completed when the I/O request is issued. The job is placed on the SQL of the ready list until the request is serviced and the job reaches the head of the list. On regaining control of the processor the job does not begin a fresh quantum but completes the remainder of its previously assigned quantum.

b. BASIC QUEUEING AND PROGRESS OF A JOB THROUGH THE SYSTEM

The basic queueing incorporated in the system is illustrated in fig.14.

When a job arrives at the system it is assigned a priority and proceeds to be loaded page by page onto auxiliary storage from the input device. Compiling, assembling and linkage editing phases are ignored for simplicity in our model. Unit record I/O (e.g. card reader, line printer) are assumed to have little system overhead and to be spooled in any case.

If there is enough room in core for the job's working set of pages (see Ch. on "the Simulation Model" for an explanation of how this is decided) then a 'starter' page is loaded into core and the job is ready to begin execution.

The number of blocks necessary to contain the job's working set are reserved. This is a basic requirement of the working set strategy, since it insures that there will be enough space in core for the working sets of all jobs currently in the execution phase. Thus no job need demand blocks previously assigned to a page of another job's working set, thus minimising page traffic.

Thus a 'starter' page is loaded and the job is ready for execution. The job is placed on the short quantum level of the ready list. When it reaches the head of the list the central processor is assigned to the job and begins execution.

The job executes until one of four possible events arise.

1. it issues an I/O request
2. it references a page not in core, i.e. a page fault occurs
3. it completes the time quantum allocated
4. the job completes

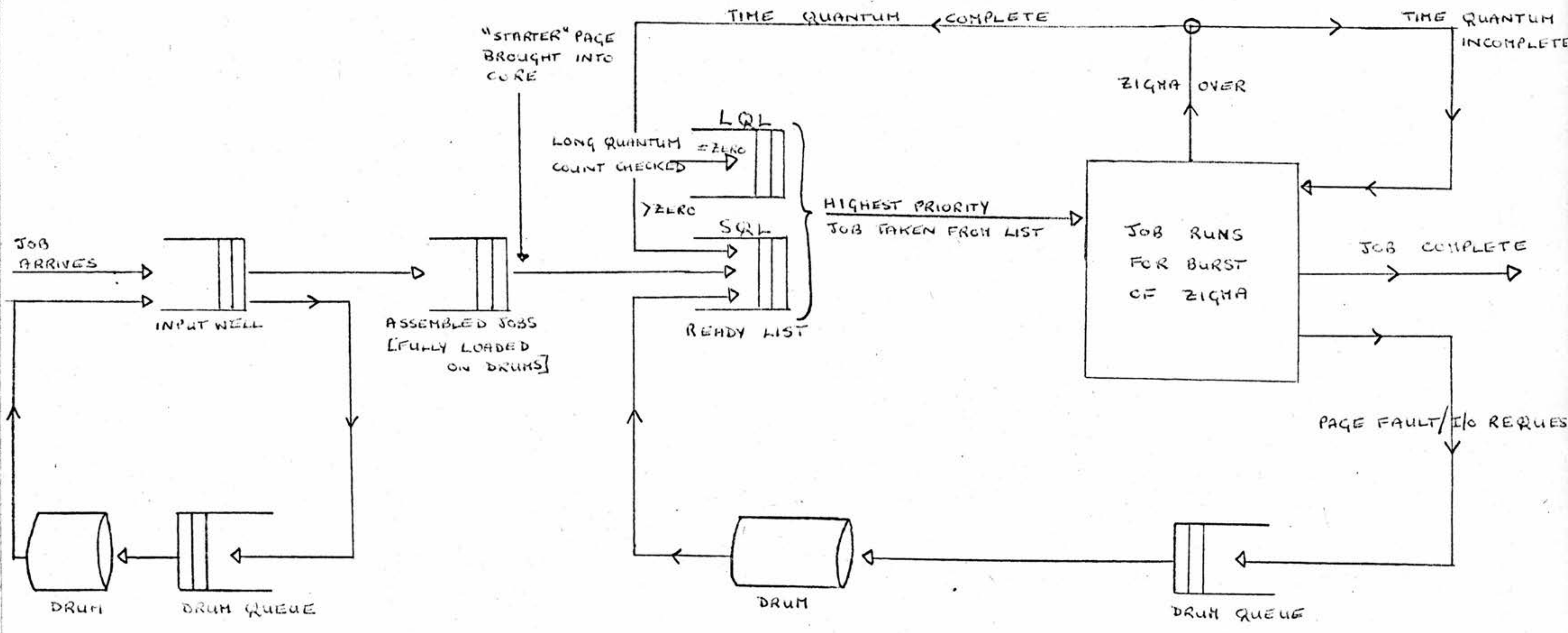


Fig 14 BASIC QUEUEING IN THE COMPUTER SYSTEM

In all four cases the job loses control of the processor to the job now at the head of the ready list.

In cases 1 and 2 the job enters the queue for drum attention. In case 1 the required I/O processing takes place and the job is returned to the ready list. In case 2 the paging algorithm is consulted and makes the decision as to which block of physical memory is to be allocated to the page being brought into core. Once the page is in core the job is returned to the ready list.

When the job completes the time quantum allocated (case 3) the long quantum count for that job is decreased by one. If this count is positive the job is placed back onto the short quantum level queue; if it is zero or below then the job is placed on the long quantum level queue. If there are no other jobs on the ready list the job is simply allowed to continue processing for a further quantum.

If the job completes it relinquishes all core memory space and auxiliary storage space, all related tables are cleared and all references to the job in the system are removed, and appropriate statistics compiled.

II 2. THE SIMULATION MODEL

a. REQUIREMENTS

In order to allow the user to observe the effect of paging algorithms which he may have written for this reasonably complex time-sharing system, the model had to fulfil three requirements.

First, the model was to serve as a test vehicle for very diverse paging algorithms. It had, therefore, to be responsive to changes in these routines. Further, the paging algorithm had to be relatively isolated from the rest of the model so that changes could readily be programmed and incorporated. An efficient interface had to be developed.

Second, the model had to be responsive to changes in the configuration of the system, for example, to changes in core size, drum size and drum speed. Such changes would help determine the efficiency of the paging algorithm under differing conditions. Further, the configuration had to be easily adjustable, such as by appropriate modification of parameter values.

Third, the model was to serve as a means of determining the effect of various job mixs and loads upon the performance of the paging algorithm, the performance of the system and throughput of jobs. Thus it had to be responsive to adjustments in the requirements of particular jobs. Further, the job stream had to be easily adjustable, such as by the change of a few parameters.

b. LEVEL OF DETAIL

Since the paging algorithm was to be of such great importance the model had to keep track of every page in the system at all times. The identity and current state of jobs had to be retained over time slices, I/O waits and page fault waits.

Care had to be taken not to include disproportionately scaled activities in the model, since this would lead to an inefficient model, for example, all I/O for user programs is assumed spooled onto the drum. Activities, such as delays due to dynamic address relocation, the effects of associative memory operation, compilation, linkage editing were ignored. Compilation and linkage editing are assumed to be merely the processor and I/O operations of another job, (i.e. the compiler and the linkage editor). Those activities with a level of detail finer than the activities at the paging level were not included. Similarly, activities with a level of detail more gross than those at the paging level were excluded. Thus, for example, the amounts of drum storage space required for job's pages are to be specified as parameter values, and the presumption is made that enough blocks will be available on drums. A simple pencil and paper calculation will enable the user to ensure that this is so.

Consequently, the basic unit of time was chosen to be 100 μsec and of storage to be the page, the size of the page being a variable parameter.

When including information about the working set in the paging algorithm, results will be affected by the values of the sampling interval, σ, and the number of sampling intervals, . Thus at the level of detail catered for here, it was decided to include σ and κ as variable parameters.

c. LANGUAGE SELECTION

Since this simulation model is to be used as a teaching tool, it is important that the program implementing the model should be readily understood. Thus it was decided to write the program in a language that is commonly known and which has a high degree of portability. FORTRAN appeared the best choice to fit the requirements.

Since the program is a simulation, it might normally be expected that it be written in a simulation language. However these languages generally have a poor execution speed relative to general purpose languages and often utilize memory space rather inefficiently. Speed and efficient memory utilization were relevant to the simulation since it has to be within the range of time and space allowed to students' everyday jobs. Although simulation languages have built-in queueing facilities, these can be easily implemented in FORTRAN subroutines. Further FORTRAN enables a closer approximation to the actual workings of the system than would be possible using the simulation languages generally available to computer users, since different types of queuing techniques may be used at different points throughout the model. The queuing techniques may be readily created to satisfy the particular requirements of this model.

d. REPRESENTATION OF JOBS

As with many simulation models (e.g. Nielsen (3),(16) and simulations used for design purposes) we are dealing here with a model of a computer system which does not yet exist. Consequently there is the immediate disadvantage of not knowing exactly what the job mix will be and how the jobs will behave during execution.

Nielsen's simulation of time-sharing systems acts as a simulator both for existing time-sharing systems and for the design of such systems. Thus the job mix will vary from system to system and in some cases may be entirely unknown. He has developed a job description language in which eight instruction types are used to indicate the desired behaviour of a job during its simulated execution. Description sequences for a particular job are constructed from a set of master sequences which represent a prototype for each different job type.

Katz's (17) simulation for System /360 machines used a Job Generator. Frequency distributions and tables of information giving the overall statistical properties of the user's job population were used as input to the Job Generator. The latter was designed so that the set of jobs produced reflected the actual jobs of a particular user's installation.

In a simulation study of the optimization of performance of time-sharing systems (18) the job-stream is generated using Monte-Carlo techniques. This method was adopted in an attempt to reduce the number of parameters required to describe the job mix.

The approach used in our simulation model is based on characteristic equations for the paging behaviour of jobs and random number generators and frequency distributions for other aspects of the jobs' behaviour. All the information used is based on results of various studies on the behaviour of programs in a paging environment (18, 12, 13, 14, 15).

Following is a description of how this simulation handles the job-description parameters and the justifications for the methods chosen.

<u>prediction of I/O requests</u>

I.F. Freiberg, in a paper entitled 'Dynamic Behaviour of Programs' (15) presents results obtained from an instruction by instruction interpretive execution of different classes of programs on an IBM 7044. He claims that the data obtained can be used as realistic input to simulation models of multi-programmed and fixed page size computer systems. Part of the data showed that most of the supervisor calls occurred for I/O operations and further that a program does not execute very many instructions between successive supervisor calls. From this it was concluded that it would be desirable to include the time taken between successive I/O requests as a parameter for individual jobs. It appears that the value of this parameter should generally be small relative to the total execution time of a job. Being a parameter however it may be varied from job to job so as to make some jobs virtually I/O bound and others relatively free from I/O activity.

The actual parameter of the interarrival time of successive I/O requests is expressed in terms of a maximum i.e. the parameter represents the maximum interval between successive requests. The time between individual requests is generated by means of a random number generator which generates values between 1 and the parameter specified.

prediction of page faults

Fine, Jackson and McIsaac (12) did an empirical study in which programs were executed in an interpretive manner on an AN/FSQ-32 computer. Their results illustrate, among other things, a page demand as a function of time as shown in fig.15.

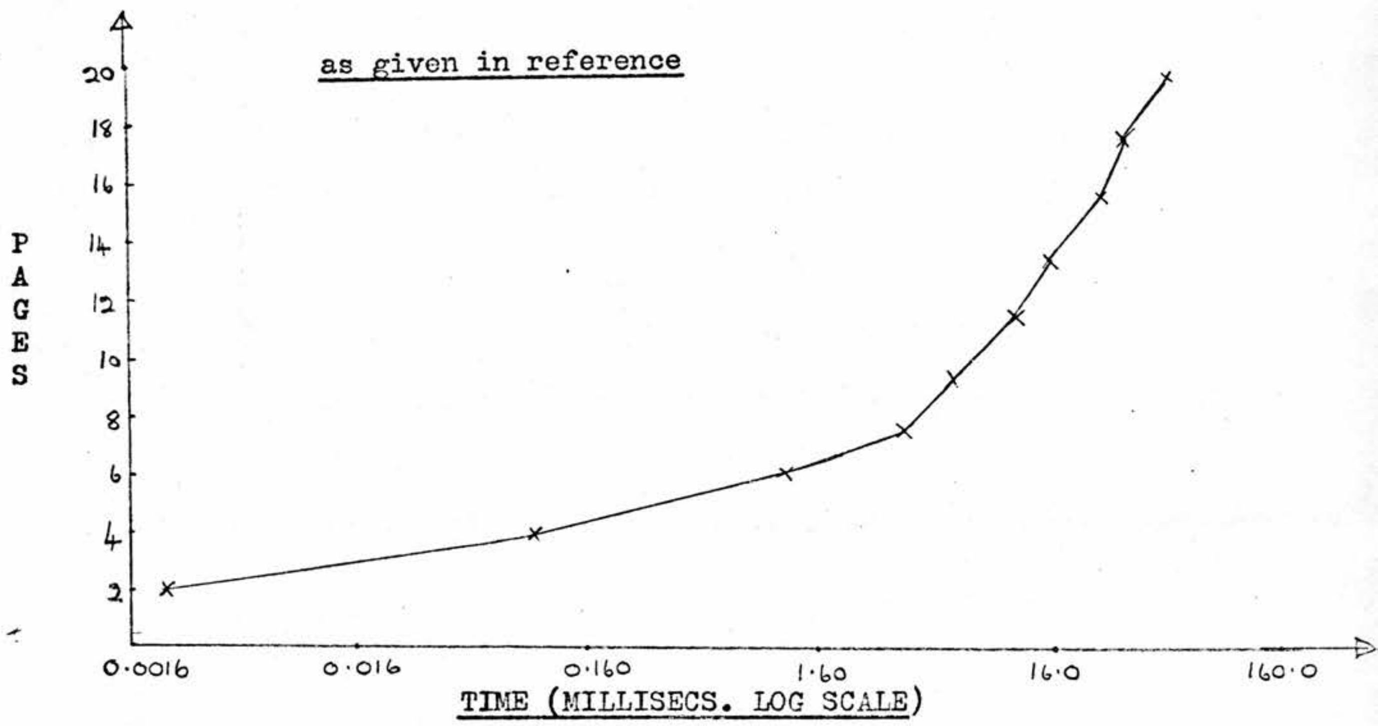
The number of pages accessed initially is extremely high. On average, the first 10 pages were required in less than 5.6 ms; in half of the cases, these first 10 pages were required in less than .8 ms. In this paper five rather large programs were studied, namely LISP, 44 pages; META5, 14 pages; GPDS, 41 pages; TINT, 23 pages; SURE, 30 pages. The page size was taken to be 1^k words. The programs were not in any way designed for a paged machine. Even with the difference in the functions of the programs considered, the over-all pattern of page demands was shown to be fairly consistent.

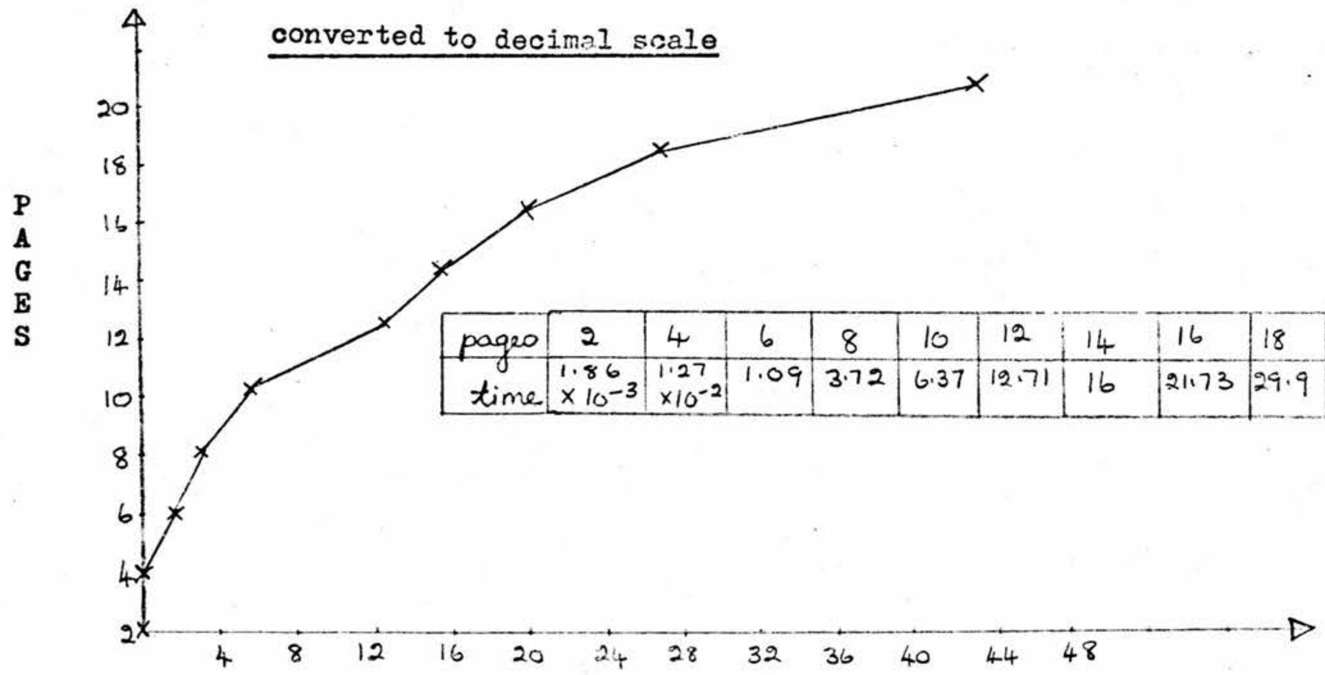
The conclusions of the study express three basic points:

1. In general programs demand pages at very rapid rates until they have 'sufficient' pages in core.
2. Frequently programs do not run for very long even after having acquired a sufficiency of pages.

FIG 15 PAGE DEMAND

DYNAMIC PROGRAM BEHAVIOR UNDER PAGING, FINE et al (See Ref. 12)





pages	2	4	6	8	10	12	14	16	18	20
time	1.86×10^{-3}	1.27×10^{-2}	1.09	3.72	6.37	12.71	16	21.73	29.9	46.5

fig 15

3. For those programs which do run for a time after acquiring a sufficient number of pages, this number is usually a considerable proportion of the total number of pages associated with the program.

In an empirical interpretive study of programs on the IBM 360/50 computer, Varian and Coffman, in 1967, produced similar results concerning page faulting activities. In 1968 they published a further study which included an experiment in which they varied the number of pages of a job allowed to remain in core during execution. This concluded that programs operating with substantially less than half their pages in core caused excessive page turning.

The studies of Freiberg and Varian and Coffman show that once a process begins execution, the page-access characteristic tends to that given in fig.15. They further agree with Fine et al in the evidence that excessive page turning takes place when programs are made to operate while substantially less than core-resident. This implies that the subset of information necessary for efficient execution of a program must be relatively large. Consequently the 'working-set' of pages (as defined earlier) must consist of a large proportion of the program's total pages.

Thus it was decided to develop equations which would predict the page fault rates using the empirical evidence discussed so far. It was seen to be desirable to have different equations for differing points in the job's execution. The curve in fig.15 was thus approximated by two straight lines whose point of intersection was taken to be the point where the working set had been reached, see fig.16. Note, however, that we are concerned only with the <u>number</u> of pages in the working set and not their individual identity.

Since demand paging is operable in this system it is possible that the paging algorithm will permit active pages of one process to be removed from core to make room for those of another process which is currently in control of the central processor.

In the simulation model the equations are such that at any point in time they can always predict when the next page fault is going to occur for a particular process. The rate of the page faulting for that process will be affected by the number of its active pages which have been removed since it last had control of the processor. Thus the rate of page faulting of a process at any point in time can be any of the five possible conditions following:

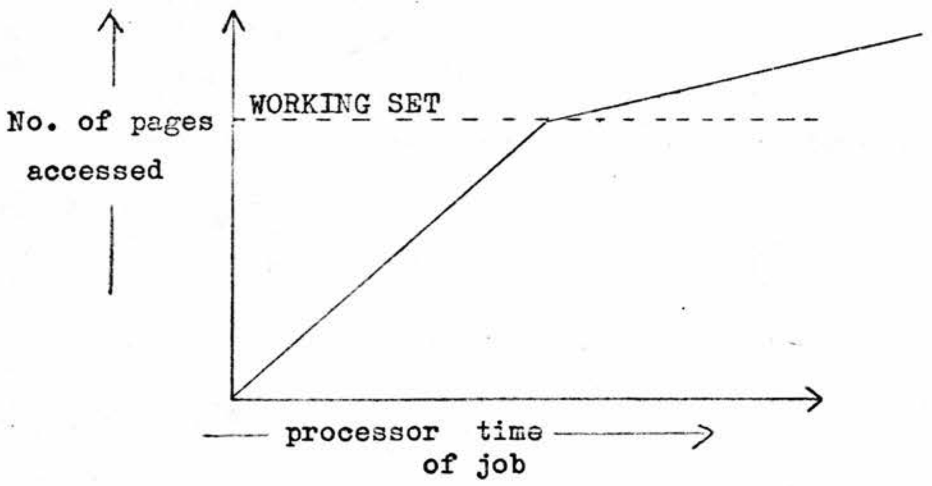
let the current set of pages in core = CS

let the working set of pages for the process = WS *

let the number of active pages removed = NPR

* as discussed on P31 the working set size should be a large proportion of the process's pages. We fix this proportion at $\frac{2}{3}$ rds. (Though this may be altered)

FIG 16 APPROXIMATION OF PAGE DEMAND CURVE



(1) CS WS-1 and NPR = 0 (fig.17)

(2) CS WS-1 and NPR = 0 (fig.18)

(3) CS WS-1 and NPR 0 (fig.19)

(4) CS WS-1 and NPR (CS-WS) (fig.20)

(5) CS WS-1 and NPR (CS-WS) (fig.21)

The point in time that is of most interest is that point at which the process regains control of the processor, since the equations must be able to predict when the next page fault is going to occur.

The actual rates at which the page faults are to occur are given by the slopes of the lines, i.e. g_1 before the working set is in core and g_2 after the working set has become core-resident. The values of g_1 and g_2 will be parameters of a particular process, as is the value for WS, thus enabling different job types to be assembled.

The values of the constants c_1, c_2, c_3 in the equations are found by simple geometric and arithmetic calculations.

<u>details of the equations</u>

<u>Case (1)</u> <u>CS < WS-1 and NPR = 0 see fig.17</u>

On regaining control of the central processor the current set of pages is less than the working set and no active pages have been removed. Thus the process continues to issue page faults at the initial rate for that process i.e. $y = g_1 x$. Consequently if the process regains control at some point (PROTIM) in the uninterrupted processor time of the job and that job currently has CS pages in core, the next page fault will occur at $x = (CS+1)/g_1$.

Case (2) $CS \geqslant WS-1$ and $NPR = 0$ see fig.18

In this case the current set of pages in core is greater than the working set. No active pages have been removed and the page-demand rate continues at the second rate i.e. according to $y = g_2x + c_1$ where $c_1 = WS(1 - g_2/g_1)$ (c_1 is calculated from the fact that the lines $y = g_2x + c_1$ and $y=g_1$ intersect at WS).

Thus if the process regains control at PROTIM the next page fault will occur at $((CS+1) - c_1)/g_2$.

Case (3) $CS < WS-1$ and $NPR > 0$ see fig.19

The current set of pages is less than the working set but some active pages have been removed. Thus the process will continue to issue page faults at the first rate i.e. according to g_1, until the working set number of pages are in core. There is however a displacement from $y = g_1x$ to consider, thus the page fault rate will be according to $y = g_1x - c_2$. (see fig.22 for calculation of c_2) If the job regains control of the processor at PROTIM then $c_2 = g_1 \times PROTIM + NPR - CS$ and the next page fault will occur at $x = (CS - NPR+1+c_2)/g_1$.

Case (4) $CS \geqslant WS-1$ and $NPR \leqslant (CS-WS)$ see fig.20

The job regains control at a point (PROTIM) where the number of pages in core is greater than the working set of pages, some active pages have been removed but these do not interfere with the working set. Thus the rate of page faulting will now follow the line given by $y = g_2x - c_3$ i.e. will be at the second rate. $c_3 = g_2 \left(PROTIM + \frac{NPR - (CS-WS)}{g_1}\right) - WS$

see fig. 22 for calculation. The next page fault will occur at $x = (CS-NPR + c_3)/g_2$.

FIG 17 CASE (1)

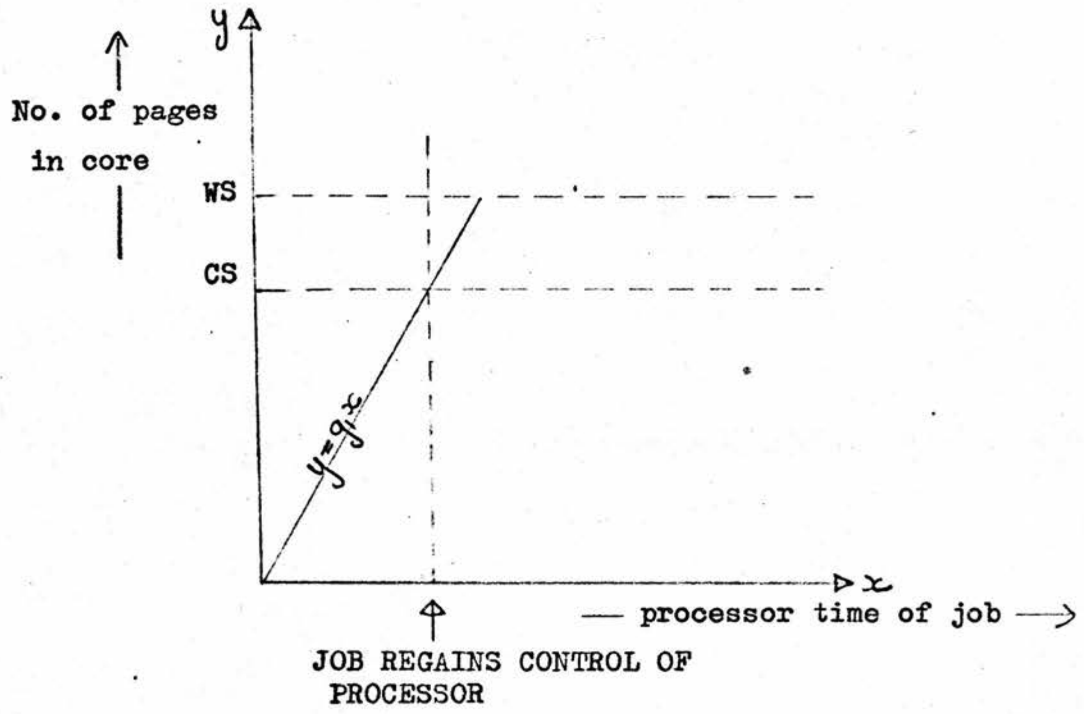


FIG 18 CASE (2)

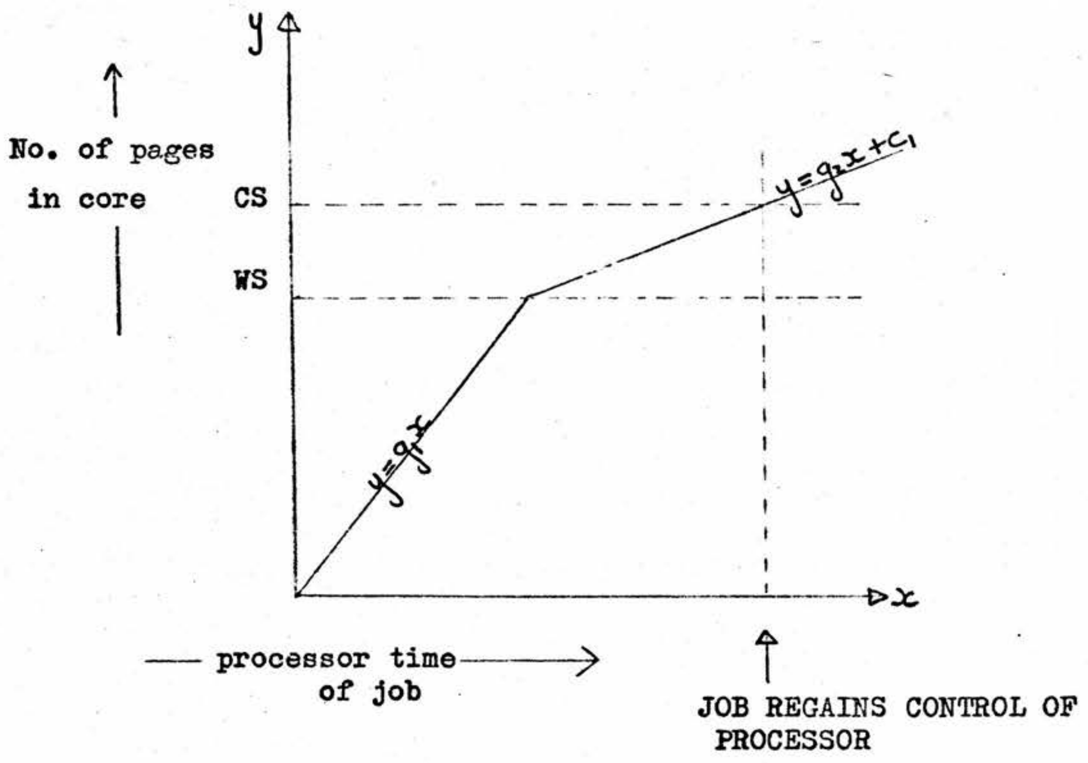


fig 17, 18

FIG 19 CASE (3)

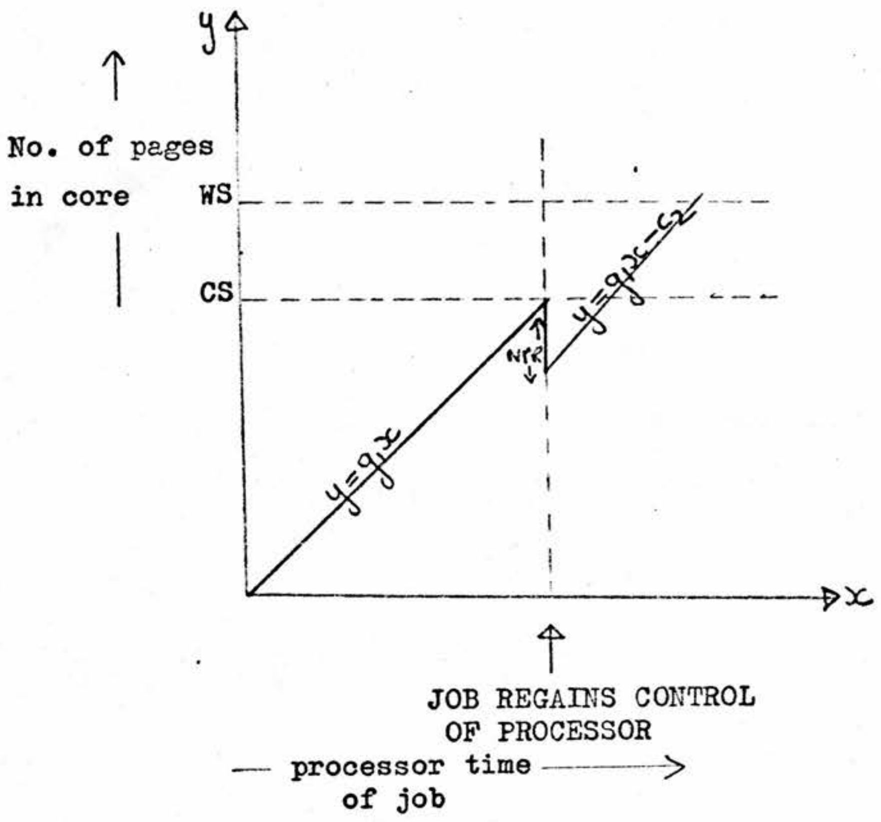


FIG 20 CASE (4)

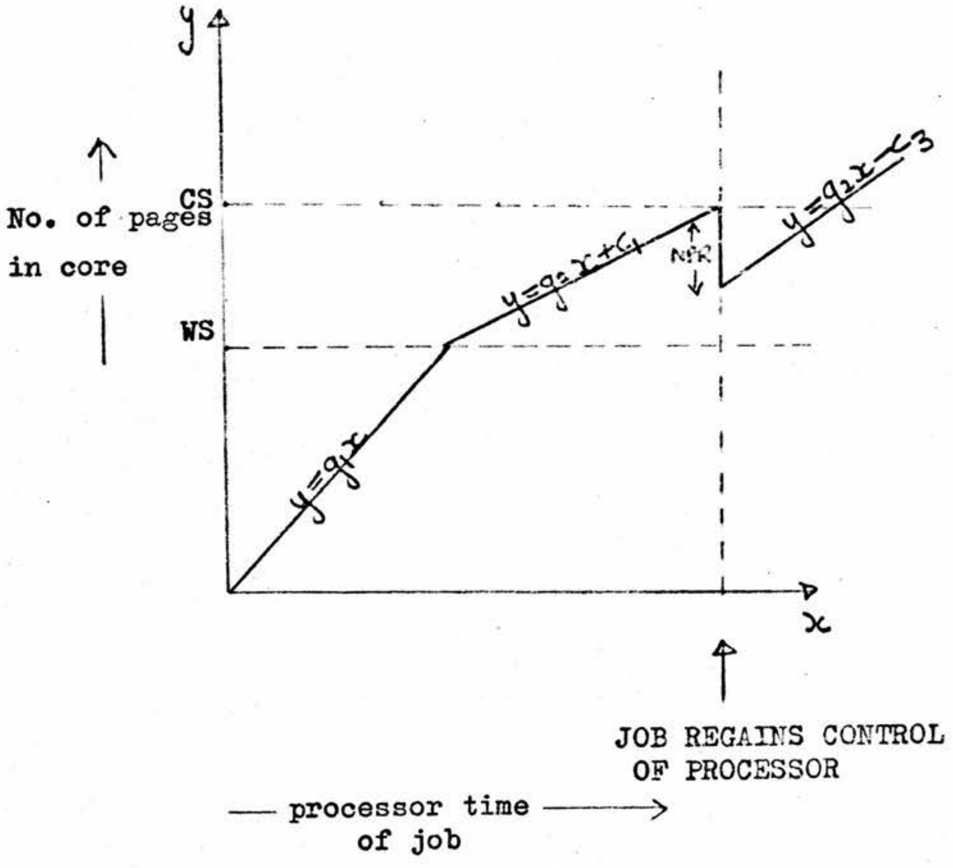


fig 19, 20

Case (5) CS $\geqslant$ WS-1 and NPR$>$(CS-WS) see fig.21

The job lost control of the processor when the number of pages in core was greater than the number in the working set. On regaining control the job finds that so many of its active pages have been removed that there has been interference with the working set. Thus the job now page-faults at the initial rate g_1 until the working set is restored i.e. along the line $y = g_1x - c_2$ where $c_2 = g_1 \times \text{PROTIM} + \text{NPR} - \text{CS}$.

Thus if the job regains control at PROTIM the next page fault will occur at $x = (\text{CS-NPR} + 1 + c_2)/g_1$.

Other aspects of job description

So far we have discussed the methods by which the model handles prediction of I/O requests and page faults. The remaining job description variables are:

Priority

Core size requirements

Long quantum count (for working set strategy)

CPU time required

Record size of job.

These may be specified by any method the user of the model requires. The subroutine which generates pseudo-random numbers is made available to the user at the time of specification. Further details about job parameter specification may be obtained in Part IV Ch 1, pages 70 -77.

FIG 21 CASE (5)

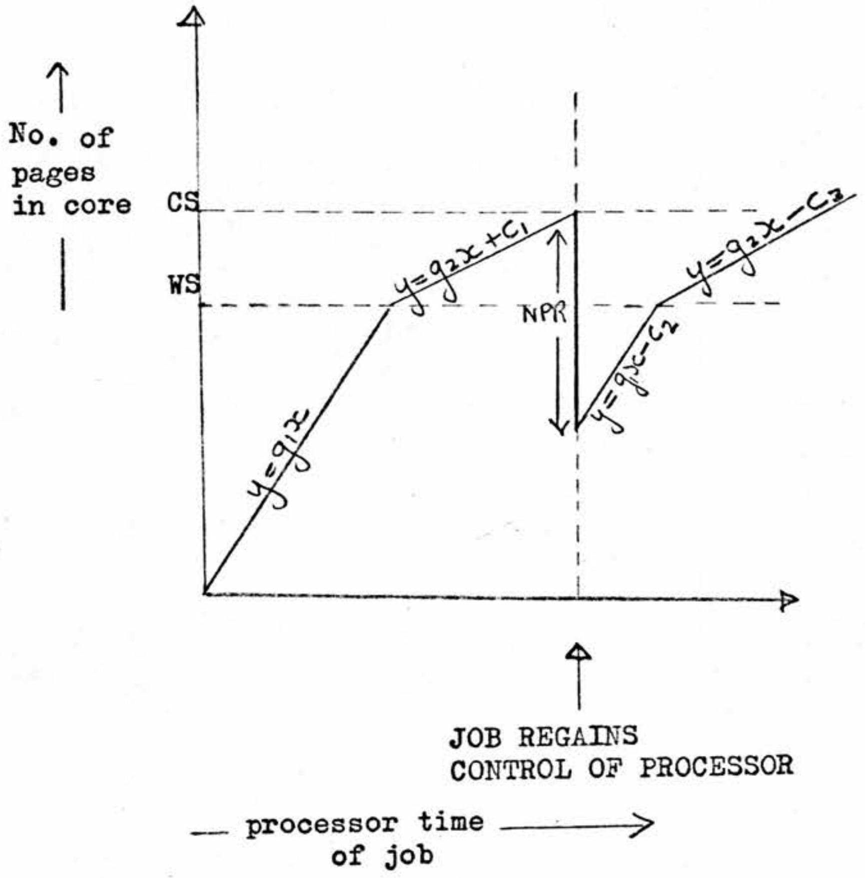
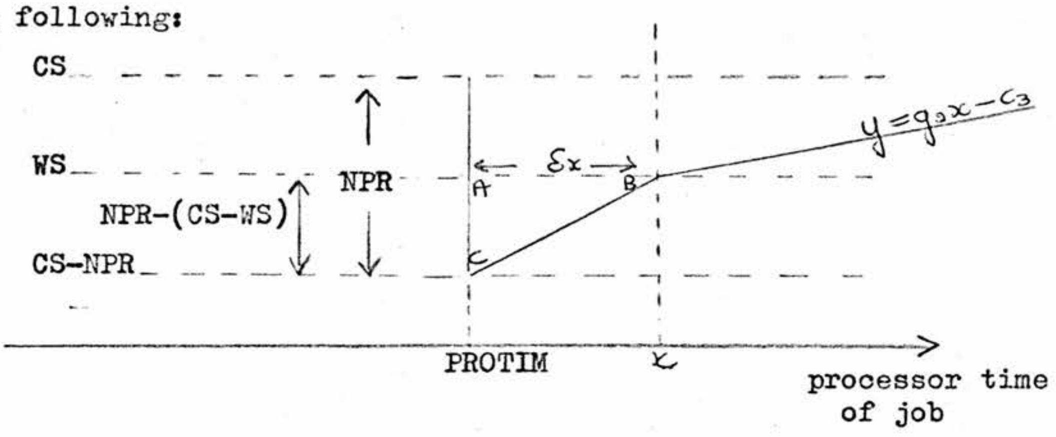


fig 21

FIG 22 CALCULATION OF C_2, C_3

It is clear from fig.21 that the lines $y=g_1x-c_2$ and $y=g_2x-c_3$ intersect at y = WS, x unknown. A closer look at fig.21 shows the following:



In the triangle ABC, les δx be the unknown side AB

then x = PROTIM + δx

from the diagram:

$$\text{the gradient of CB} = \frac{AC}{AB}$$

$$\text{i.e. } g_1 = \frac{NPR - (CS-WS)}{\delta x}$$

$$\text{thus } \delta x = \frac{NPR - (CS-WS)}{g_1}$$

Thus the lines $y = g_1x-c_2$, $y = g_2x-c_3$ intersect at

$$y = WS, \quad x = PROTIM + \frac{NPR - (CS-WS)}{g_1}$$

to find $c_2 c_3$

$$y = g_1x-c_2$$

$$\therefore WS = g_1 \left(PROTIM + \frac{NPR - (CS-WS)}{g_1}\right) - c_2$$

$$\therefore \underline{c_2 = g_1 x PROTIM + NPR-CS}$$

$$y = g_2x - c_3$$

$$\therefore WS = g_2 \left(PROTIM + \frac{NPR - (CS-WS)}{g_1}\right) - c_3$$

$$\therefore c_3 = g_2 \left(PROTIM + \frac{NPR - (CS-WS)}{g_1}\right) - WS$$

e. STRUCTURE OF THE MODEL

The model is similar to that used for BASYS since it is a next event type simulation model and simulates seven events. The events contain considerably more detail and are more closely interrelated than in BASYS but the basic structure is the same as fig.3. see fig.23

These seven events are ordered by means of an event controller which is basically a linked list ordering events with respect to the time at which they are due to occur. Each event simulates some process which a job may go through whilst in the computer system and predicts when the next event for that job will occur. It is sometimes necessary for a job to enter a queue e.g. awaiting central memory space, awaiting use of the drum to complete an I/O transfer or to wait on the ready list. The event list does not contain an entry for a job while it is in a queue but once the job reaches the head of the queue it is removed from that queue and placed on the event list.

The events which are simulated are described below;

Event 1 Simulation of Job Arrival

The event 1 routine samples the job mix distributions for job N for which the event is taking place. It has the following functions:-

1. determines maximum interarrival times of I/O requests for the job N
2. determines the page fault rates g_1, g_2 for the job N

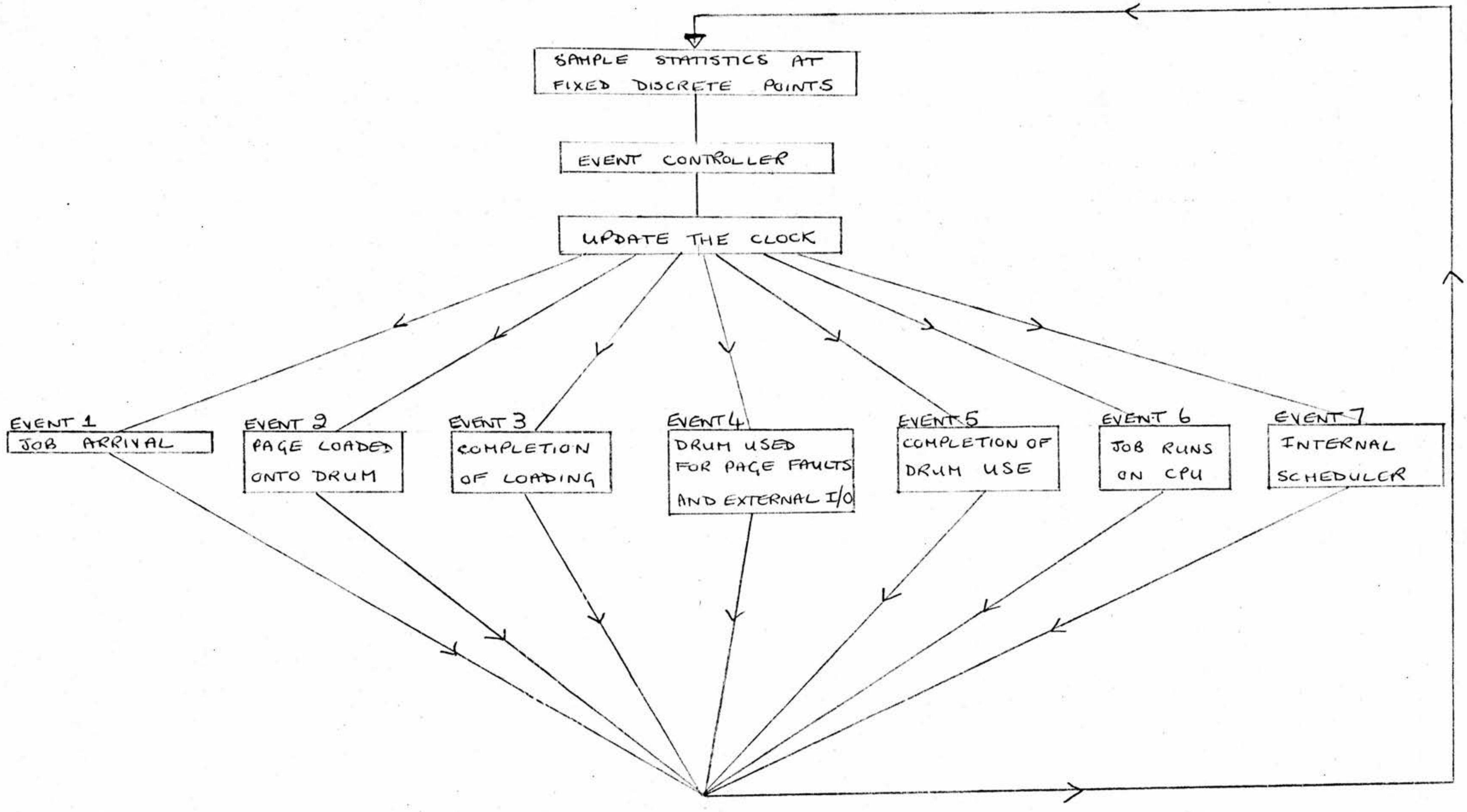


fig 23 THE SIMULATOR STRUCTURE

3. assigns a priority to the job
4. generates the central memory requirements for the job
5. generates a long quantum count associated with the job
6. predicts the approximate central processor time required by the job
7. predicts the approximate size of the working set of pages for that job
8. generates the record size for the job.

The routine then calculates the number of pages in the job (this will vary according to the current page size in the system), and determines the size of the working set (for prediction of page faults). Event 2 is then scheduled for job N and the arrival time of job N + 1 is predicted. Event 1 is scheduled for job N + 1 and control is returned to the event controller.

An event is "scheduled" by linking up the associated job entry on the event list. That is, the event number and the time at which the event is due to occur are entered into the event list along with the job number with which they are associated. The event then occurs when this entry reaches the head of the event list i.e. when it is the first event in time due to occur. When this entry reaches the head of the list the time associated with the event is closest to the time on the simulation clock. The simulation clock is then advanced to the time associated with this event.

The clock is initially set to time zero at the start of the simulation and is advanced only when an event takes place. The clock allows for several events to occur at the same time since it is only advanced when the time associated with an event is greater than that of the clock. The time at which an event is "scheduled" to occur is always greater than or equal to the clock time.

<u>Event 2 Page Loaded onto Drum</u>

The Event 2 Routine simulates the loading of a page of job N onto the drum from an external device. If the drum is busy then the request is entered into a drum queue. If the drum is free it is assigned to job N for a time that is a function of the drum speed, read/write rate, number of records associated with job N and traverse time. If, however, all the pages of job N have been loaded onto the drum then job N is assigned to the queue for central memory space and event 7 the internal scheduler is scheduled for it. If pages are still being loaded event 3 routine is scheduled, representing the delay time in completing the loading of a page.

<u>Event 3 Completion of Loading</u>

Event 3 represents the completion of the loading of a page. The drum is freed and the entry at the head of the drum queue for job H is examined. If it was in the drum queue to to load another page onto the drum then event 2 is scheduled for job H. However if it was in the queue to carry out an I/O transfer or deal with a page fault then event 4 is scheduled for job H.

Job N has completed the loading of a page so the page table for job N is updated to keep account of the position on the drum that the page is stored. The list of free pages available on the drum is also updated. Event 2 is scheduled for job N.

<u>Event 4</u> <u>Drum Request</u>

The Event 4 routine simulates the use of the drum for I/O transfers and page faults. If the drum is busy job N is entered into a queue otherwise the drum is assigned to job N and event 5 is scheduled.

<u>Event 5</u> <u>Drum Completion</u>

Event 5 signifies the completion of a drum transfer for I/O or a page fault for job N. Job N is then placed on the Ready List since it is once more ready to run on the CPU and event 7 is scheduled for this job. The drum is now free and if there is anything in the drum queue then the head of the queue is assigned to the drum and event 2 or event 4 scheduled as appropriate for job H.

<u>Event 6</u> <u>CPU Execution</u>

The event 6 routine simulates the actual running of a job on the central processor.

The processor is reserved and pointers set up indicating which job has control of the processor (CPUJOB) and what time it gained control (CPUST). The job now in control of the processor continues executing as if it had never lost control and all its counters and associated statistics are continued and updated. Firstly, if the job has completed a multiple of σ (zigma) (the sampling interval) units of time in execution then the use-bits of the job are shifted one place to the right.

A) Secondly, if the job has completed its estimated CPU time then event 7 is scheduled for the job and control returned to the event controller.

Thirdly, if an I/O request has not already been predicted to occur at a certain time then one is predicted and a flag set up to say that this I/O request is waiting to be carried out. Similarly if a page fault has not already been predicted then one is predicted and a further flag set up to say that this page fault has still to be satisfied.

A test is done to see whether the I/O request or the page fault is to occur first. Suppose it is the page fault then a further test is done to see if this page fault is to occur within the next burst of zigma on the CPU. If it is then event 7 is scheduled to deal with the page fault and control is returned to the event controller. Similarly with the I/O request.

If neither the page fault nor the I/O request is to occur within the current burst of zigma then the job completes zigma.

The use-bits are shifted and the whole process (from A) above) is repeated provided the job still has some of its time quantum left. If the job has completed the time quantum allocated to it then event 7 is scheduled where the job will be put onto the ready list. Control is then returned to the event controller.

<u>Event 7</u> <u>Internal Scheduler</u>

The Event 7 is a simulated combination of a high level and low-level scheduler. This routine selects the next "suitable" job to put onto the ready list and also decides which job is to run next on the CPU. Thus it replenishes the ready list and keeps the CPU busy.

Assume Event 7 has been called to deal with job N.

If job N is in the central memory queue then all its pages have been loaded onto drums and it is now requesting that its first ("starter") page be loaded into core so that it might begin execution. Suppose the working set strategy is being adopted, then the "starter" page is loaded into core provided that there is enough room for its working set (the size of which has been predicted in the event 1 routine). The working set number of pages are reserved out of those available in core. Thus event 7 initiates the loading of the "starter" page into core, event 4 is scheduled for job N and control returns to the event controller. If the scheduler decides not to load the initial page then control is simply returned to the event controller.

If job N is not in the central memory queue and it is on the ready list then it is requesting use of the CPU. If the

CPU is free then it is assigned to job N and job N removed from the ready list. Event 6 is scheduled and control is returned to the event controller.

If, however, the CPU is not free then the scheduler must test if the priority of job N is greater than that of CPUJOB (i.e. the job currently in control of the CPU). If its priority is greater then pre-emption occurs, job N is removed from the ready list and is scheduled for event 6. CPUJOB releases the CPU and the appropriate statistics are updated.

If CPUJOB is complete all its table references are deleted and blocks occupied by it on the drum and in core are freed. The central memory queue may now be advanced if it is possible (same argument as earlier).

If CPUJOB is not complete then in the current implementation a random bit pattern is put into the "use-bits" to simulate the page reference patterns during its last run on the processor (the bit pattern is put into "working set" number of pages only, the random numbers lying between 1 and 2^K where K is the number of sampling intervals). The pages which have been written to or updated during the last run on the processor are also generated randomly and their page tables updated accordingly. The pre-empted job CPUJOB is then placed on the ready list and control returned to the event controller.

If, however, the priority of job N is not greater than CPUJOB but the CPU is still busy then job N is placed on the ready list, and control returned to the event controller.

Suppose now that job N has just been executing on the CPU and has been blocked for some reason.

If the job is complete all references to the job are deleted and the sequence of instructions carried out as when CPUJOB completed (see earlier).

If the job is not complete "use-bits" are updated, pages "written-to" are indicated and page tables updated as before. Now the reason for blocking must be determined.

If the job has blocked for an I/O request to be carried out, related statistics are updated, the CPU is released and event 4 scheduled for job N.

B) If K sampling intervals of processing have been completed by job N, each page in job N is tested to see if it is still in the working set (i.e. logical sum of "use-bits" equals 1 when the page is in core). If it is found that the page is no longer in the working set then it is marked as a candidate for removal from main memory, as an aid to the paging algorithm. Then the scheduler selects the head H of the ready list to run next. If its quantum has run out then it is assigned a further quantum according to the level of the ready list from which it is taken. If it is taken from the short quantum level, it is assigned a short quantum and its long quantum count is decreased by one. Event 6 is scheduled for job H and control is returned to the event controller.

If the job N has not blocked for an I/O request it may have blocked for a page fault. If so related statistics are updated and the CPU is released. The paging algorithm is consulted to determine which block in core is to be allocated for the demanded page. Related statistics are updated and event 4 is scheduled for job N. The sequence of decisions are then the same as if it had blocked for an I/O request (i.e. from B)) and the head of the ready list is run next.

A further possibility is that the job has blocked because it has completed the time quantum allocated to it. In this case the long quantum count is decreased by one and the job returned to the appropriate level of the ready list. If there is no other job on the ready list this job continues with control of the CPU, otherwise the head of the ready list is selected to run next.

PART III

VALIDATION AND EXPERIMENTATION

III 1. VALIDATION OF THE MODEL

INTRODUCTION

The question posed in this chapter is:
Is the model a valid representation of the type of system we are trying to model?

In some ways this is a philosophical question and a problem common to all modeling and simulation experiments. There are several accepted approaches to this problem discussed in (19,20). The approach employed here assumes that the model is valid if it satisfies the following three conditions:

(1) That the logical and mathematical relations employed in the frame-work of the model closely approximate those in the system.

(2) That the input parameters and variables compare favourably with known historical data.

(3) That the simulation model's predictions of the behaviour of the real system correspond closely with that actually observed.

INVESTIGATION

The model under consideration was built to simulate paged, multiprogramming computer systems. It is impossible to prove the validity of the model for all such systems. However, we choose one typical machine for which known historical data is available. Further, for the machine we have chosen empirical output data is also available with which to compare the results from the simulation model.

By experimentation with a model of this particular system we can see whether or not the three conditions are satisfied. If they are then we have made some progress towards proving that this is a valid model.

No attempt is made to provide conclusive evidence of the validity of the model; however, the experiment to follow and the general trends indicated in subsequent chapters should provide strong indications that the model does fulfil its purpose.

METHOD

The system chosen for comparison is the ATLAS computer once located at Manchester University since statistics are available concerning its operation in (9,21).

We first consider condition (1).

A brief description of the ATLAS system is given here and indications of how the logical and mathematical relations in the system are approximated by the model.

The basic queueing in the system is shown in fig.24 and by comparison with fig.14 can be seen to have the same basic structure as that of the system represented by our model. However, the model does not cater for user tapes and a discussion on the approximations used follows later.

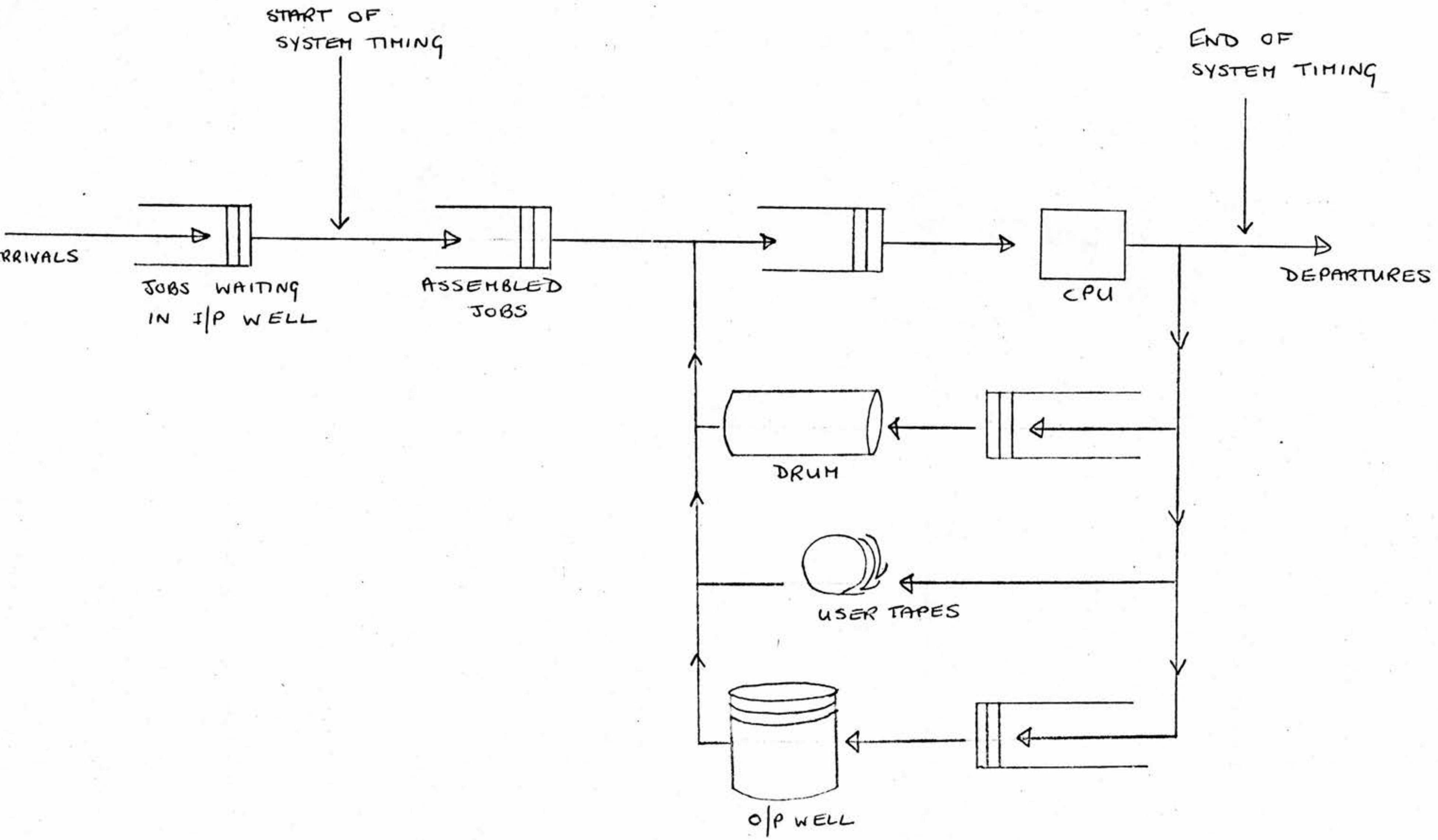


fig 24 BASIC QUEUEING IN THE ATLAS COMPUTER SYSTEM

Once jobs leave the input well they are assembled according to a priority scheme which tries to maintain a tape job and a non-tape job in the execution phase at the same time. This is implemented in the model. Jobs queue to enter into the execution phase and the number of jobs simultaneously in the execution phase is limited to two. In our model jobs queue on the ready list to await execution and the number of jobs simultaneously in the execution phase is variable according to the length of the ready list. Thus, we set this at two.

In the execution phase pages are transferred to or from the drum, tape transfers may be made to one or more magnetic tapes assigned to a job, and output may be created on the output well located on disk. In the simulator transfer of pages and creation of output information all takes place as if on drums with appropriate timing considerations. This does not have a detrimental effect on the balance of the model of the ATLAS operations since no queueing takes place for transfers to or from user tapes, (each tape is connected to core via a separate channel).

The queue discipline for drum transfers and for the use of the output well is first-in-first-out. In the queue for CPU attention, however, tape jobs are given priority over non-tape jobs. These queue disciplines are mirrored in the simulation model.

Thus the logical structures in ATLAS are reflected in the model without any major adjustments.

The mathematical relations in the system are mainly represented by the parameters given for ATLAS, including

core size = 32 pages of information
mean tape transfer time = 0.062 secs/page
mean drum transfer time = 0.014 secs/page
drum size = 133 pages
supervisor overhead to transfer control to a job = 0.002 secs
supervisor overhead to locate a page on the drum = 0.006 secs

The paging algorithm is described as a "one level store learning program" which is based on information held in "use digits". For each page of core store there is a use digit which is set when the page is accessed. All the use digits are scanned and reset at regular intervals by the central executive and a pattern of use is established. The selection of the page to be rolled out of core is made with respect to this pattern of use. This learning program is very similar to the one described in our original system. The "use-bits" in the original system being the "use digits" described here. Thus to give the paging algorithm in our model the degree of efficiency experienced by the one in ATLAS, the selection of the page for removal from core is based on the condition of the use-bits. Further, the page whose use-bits are furthest to the right will be the one selected. This is effectively a Least Recently Used paging algorithm.

Thus the logical and mathematical relations employed in the frame-work of the model closely approximate those in the system and condition (1) is therefore satisfied.

We now consider condition (2). Here a set of input parameters is described. These are based on historical data given in (9).

The parameters together with variables described earlier are used as input to the model, the final proof of the validity of the model lies in the results obtained as output. These are discussed later, (see page 50B)

<u>input parameters</u>

class 2 is considered from (9), with compute time range 1-8.

<u>class 2</u>

maximum processor requirements = 8 secs

maximum byte requirements 208000 (page size 4K)

mean no. of tape transfers mean no. of I/O requests = 344

max. inter I/O request interval = 8/344 secs for tape jobs 0.23×10^{-1} secs

page size = 4K

The input parameters actually used in the simulation are as follows:

<u>GENERAL</u>

a) core size = KSIZE = 32 pages = 128K in subroutine SYSTEM

b) page size = KPAGE = 4K in subroutine SYSTEM

c) 2 levels of priority were given to jobs to account for tape jobs and non-tape jobs:

```
IF (PJ.EQ. (PJ/2) *2) JOBDES(PJ,1) = 1
IF (PJ.NE. (PJ/2) *2) JOBDES(PJ,2) = 2
```

in subroutine JOBSIM

d) traverse time for drum transfers

$$\text{TRAVT} = \frac{\text{mean tape transfer time + mean drum transfer time}}{2}$$

$$= \frac{0.062 + 01014}{2}$$

$= 0.038$ secs

$\simeq 4$ time units (1 time units = $\frac{1}{100}$ sec)

in subroutine SYSTEM

e) NJOB = 2 in subroutine JOBSIM so that not more than two jobs may be in the execution phase at the same time.

The supervisor overheads mentioned earlier are considered small enough to be neglected and the drum size of the system is maintained at its maximum value (KSIZE = 744) since there will thus be sufficient pages for both tape and non- tape jobs.

FOR CLASS 2 JOBS

f) maximum processor requirements = 8 secs
MAXCPU = 800 time units

g) maximum byte requirements = 208000
MAXBYT = 208000

h) maximum inter I/O request interval = 0.023 secs
IOMAX (PJ) = 2 time units

The Least Recently Used paging algorithm was used in both cases. NEWLIM was set to 10000.

RESULTS

Comparison of Observations from ATLAS and Results from Simulator

ATLAS	Simulator
mean compute time =380 units	estimated CPU time of 2 jobs =269 and 564 units
mean elapsed time =3770 units	response time of 2 jobs = 1879 and 4313 units

CONCLUSION

The results obtained show that the simulation model's predictions of the behaviour of the real system correspond closely with that actually observed. Thus, the model satisfies condition (3) and proves that the simulator is capable of modelling a particular system.

III 2. A QUESTION OF BALANCE

Our model has been shown to be valid for a particular system configuration, but we require it to be valid for all systems with the properties of the system described in Part II Ch.1.

Care has been taken to ensure that the frame-work of the model reflects the logic of such systems. The user of the model, however, is responsible for the choice of input variables and parameters describing the system. His choice of system must be made carefully since it will have considerable effect on the performance of the model and consequently on the predicted performance of his chosen system.

It is possible that the user may choose "unreasonable" input parameters which will result in an unbalanced computer system. Such an unbalanced system may produce distorted simulated results and could render the system and consequently the model completely insensitive to a change in paging algorithm.

In this section, we propose some simple tests, inspired by J.H. Saltzer (22) which will help the user determine whether or not he is working with a balanced system.

The balance problem we shall consider is whether or not the core memory and processor are balanced relative to each other and to the presented job load.

THRASHING

Before meaningful conclusions can be drawn about system balance it is necessary to convince ourselves that balance measurements are not distorted by "thrashing", that is, excessive overhead caused by quantum runout or page swapping.

Consider first the case of QUANTUM RUNOUT. This concerns the values of the two parameters QANTUM (1) and QANTUM (2) which upperbound the amounts of continuous processor time allocated to a job when it leaves the short quantum and long quantum levels of the Ready List respectively. These quanta of CPU time may be given values which are too small, in which case the prime cause of processor switching will be quantum runout rather than the job blocking itself.

In addition to causing extra overheads in the system, excessive processor switching may also reduce the average response times of jobs in the system. For an intuitive notion why this is true, consider 10 jobs each of which need 5 secs of processor time. If each is to run to completion, followed by the next, the first job will be served after 5 secs, the second after 10 etc... and the last after 50 secs. On the other hand, suppose that each job is served for only 1 sec, then the processor is switched to the next, etc. in a round robin. In this case, the first job to enter the system will not leave until 46 secs have passed, the last still leaving at 50. (Since processor switching causes some overheads delay times would probably be even greater).

It is difficult to decide conclusively that processor-thrashing is being caused by quantum runout but a reliable guideline may be obtained through a comparison of the response time of jobs to their respective CPU requirements. If their response times are very great compared with their CPU requirements then it is likely that processor-thrashing has occurred and that the parameters QANTUM (1) and QANTUM (2) need adjusting to increase the processor time quanta allocated to the jobs.

Consider next, thrashing caused by excessive PAGE SWAPPING.

This situation arises when pages of a job are being rolled out of core before the job has finished with them. That is, pages of other jobs are demanding core space and are getting it at the expense of removing pages which are still in use (i.e. still in some job's working set). In such circumstances pages are rolled out of core only to be rolled in again almost immediately upon a page fault.

Excessive page swapping has three possible causes

1) the paging algorithm
2) the size of core
3) the size of jobs in the system

The paging algorithm determines which block of core and under what conditions that block of core is to be allocated to a "demanded" page. Its decision-making policy could cause a page to be removed from core when still in use. Thus a page may be rolled out of core only to be referenced again almost immediately necessitating another page replacement decision and further roll-out/roll-in hence core thrashing will occur.

The number of pages of memory available may be insufficient for the total number of job pages in the system.

When the competition for physical memory becomes very high due to over commitment vigorous page-swapping will take place i.e. thrashing will occur. Hence either the size of core or number and hence total storage demand of active jobs in the system must be altered.

Thus if thrashing caused by excessive page swapping is detected then alteration to one or more of the above three factors may be necessary.

In the simulator KSIZE represents the size of core in K bytes, KPAGE the page size in K bytes. The paging algorithm is the SUBROUTINE ALGORI and the size of jobs in the system is governed by a maximum job size MAXBYT which is the maximum total storage requirement in bytes in any one job.

Finally, how are we to decide when core-thrashing is occurring and in fact being caused by excessive page swapping?

No hard and fast answer can be given but a good guide line is a comparison of the total number of pages in the system (A)* to the total number of page faults (B) that have occurred when all the jobs have completed. If (B) is very large in comparison to (A) then thrashing must have occurred during that run.

* The total number of pages in the system is the sum of all the pages belonging to the jobs run through the system.

Thus we must assure ourselves that thrashing is not occurring in our system.

It is suggested that the system is adjusted using a "good" paging algorithm, ideally the BOR algorithm (23) but possibly the LRU (see Part VI Ch.1) will be quite satisfactory. (Different types of paging algorithms will be discussed briefly in the next chapter).

BALANCE

Once we are convinced that "thrashing" is not occurring then we can consider the question of system balance. The measurements of prime importance here is that of processor idle time.

The processor may be idle for one of two reasons

1. There is actually no work to do
2. The Ready List contains work but the low-level scheduler** refuses to allow any more processes to be loaded.

When the processor is idle for the first reason, there is a potential case of processor overcapacity. If however the processor is idle for the second reason, then we have evidence that the allowable load is being limited by the amount of core memory available. Reducing processor capacity will have very little effect on total system capacity or service quality under these conditions. On the other hand increasing only memory size will increase total system capacity.

** The low-level scheduler decides whether or not a process will be loaded. It is loaded if there are enough free blocks in core to hold the process' working-set.

The user has the option within the simulator program to remove or exchange the statements which carry out this test, should he so desire. (They are clearly marked by FORTRAN COMMENT statements).

The related problem of detecting core memory overcapacity provides more difficulties since a paged core memory tends to use up all available memory, no matter how much there is. On the other hand, the fact that the memory is paged is of considerable assistance in the problem. We can reduce the size of core memory by removing a block of memory at a time from consideration in the system. As the appropriate memory size reached, processor idle time will begin to mount and the desired information of where memory "undercapacity" begins will have been found.

We thus have several simple tools available for detecting whether or not the resources of the system are well matched for the job they are trying to do. First, simple measurements indicate whether or not thrashing is being caused either by quantum runout or by excessive page swapping. Second, once we are sure that thrashing is not occurring then we may consider whether or not the system is in a state of balance.

The user may then proceed with any experiments on the system which he may require to carry out.

ONE JOB NO I/O (FIXED SYSTEM AND PAGING ALGORITHM)

fig.25A

DRUM USAGE/CLOCK TIME

CORE SIZE = 32 PAGES

	N° OF JOB PAGES	
TIME	25	49
160	58%	65%
320	31%	61%
480	20%	57%
640	15%	53%
800	12%	53%
960	10%	53%
1120	8%	
1126	8%	

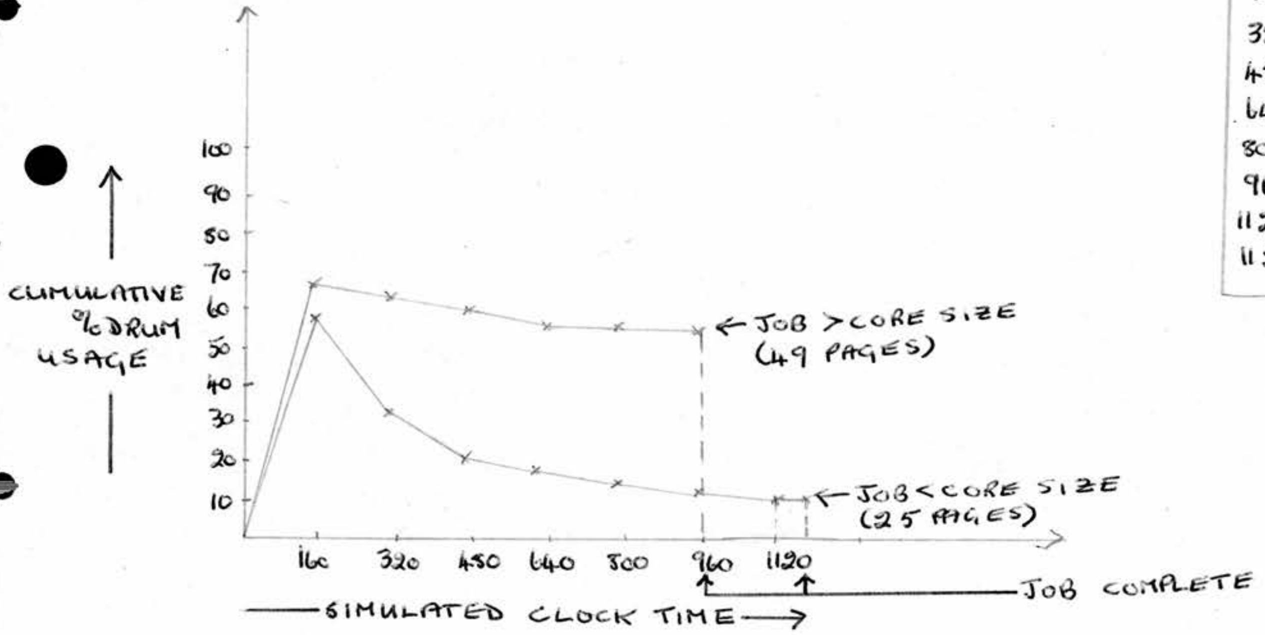
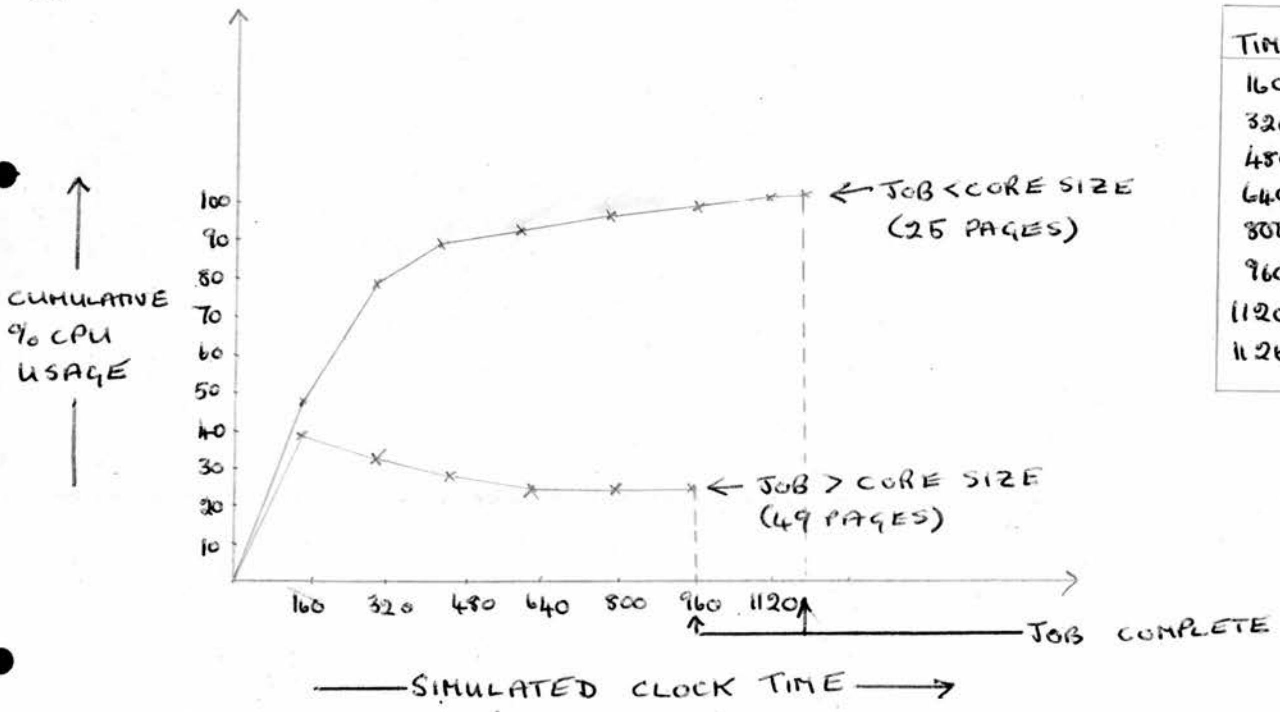


fig.25B

CPU USAGE/CLOCK TIME

CORE SIZE = 32 PAGES

	N° OF JOB PAGES	
TIME	25	49
160	48%	39%
320	79%	31%
480	87%	27%
640	90%	26%
800	92%	25%
960	93%	25%
1120	94%	
1126	94%	



III 3. INVESTIGATION OF PAGING BEHAVIOUR

In this section we illustrate that the general trends of the results obtained from the simulation are of the type expected from the system we are simulating.

Several runs were made initially to establish "reasonable" input parameters. For example, it was found that when the maximum processor requirements of a job were equal to 1000 simulated time units (MAXCPU= 1000), processor time quanta allocated to any job of 40 and 100 time units (QANTUM(1) = 40 and QANTUM(2) = 100) avoided excessive overheads due to processor switching at quantum runout.

a) Response to different job parameters

Consider first, two runs in which only one job is run through the simulator using a fixed system configuration and paging algorithm. In the first run we allow the job to be greater than core size (job size = 49 pages, core size = 32 pages) and on the second to be less than core size (job size = 25 pages, core size = 32 pages). No I/O requests are issued in either case. A comparison of the percentage drum usage during the two runs is given in fig.25A and a comparison of the percentage CPU usage in fig.25B.

It can be seen from the graphs that a job which is greater than core size causes a great deal of drum activity and allows very little actual processing to occur. The job with 25 pages had a total of 24 page faults and had no pages removed from its active working set. Whereas the job with 49 pages had a total of 209 page faults and had 51 pages removed from its active working set over an equal period of simulated clock time.

Both jobs had a predicted execution time of 1000 units and after 958 units of simulated clock time the job of 25 pages had been in the execution phase for 826 units and the job of 49 pages for only 212 units. It seems evident therefore that a great deal of page faulting activity is occurring when the job is greater than core size.

(In fact, the job with 49 pages had not completed in 30 mins of actual execution time on the IBM 360/44, whereas the job with 25 pages completed in 7 mins of actual CPU time).

Consider, next, a comparison of two runs in which the execution of four jobs is simulated. The sum total of the pages of these four jobs adds up to less than core size, (29 pages in the 4 jobs, 32 pages in core). In the first run none of the jobs issue I/O requests, but in the second all jobs are I/O bound, (IOMAX(PJ)=2 and MAXCPU = 500).

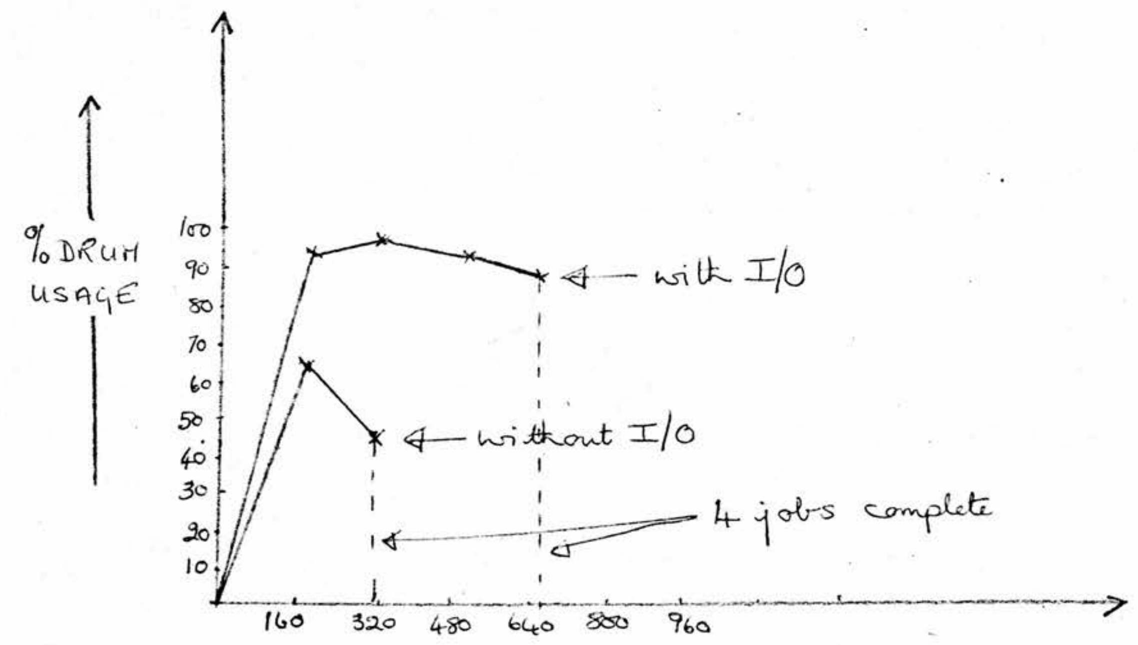
Fig.26A shows the comparison of the percentage drum usage and fig.26B a comparison of the percentage CPU usage. Jobs not issuing I/O requests finish executing in a shorter time than those which do. Fig.26C shows a comparison of the response times of the jobs with and without I/O. The jobs are identical in all other respects, for example, the actual time spent on the CPU is identical in both cases. For instance, although job 2 is only executing for 32 time units, when I/O requests are issued it takes 137 units to complete in comparison with only 73 units when no I/O is issued.

4 JOBS WITH TOTAL PAGES LESS THAN CORE

fig.26A

DRUM USAGE/CLOCK TIME

core size = 32 pages, total pages = 29

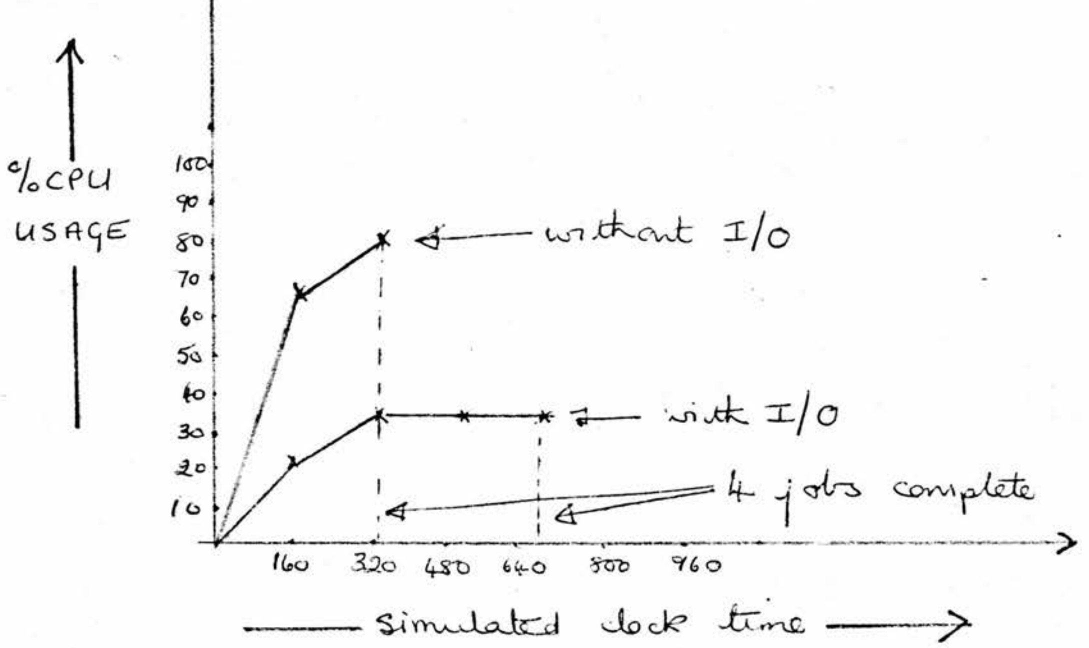


time	I/O	no I/O
160	91%	65%
320	94%	44%
480	86%	
640	84%	

fig.26B

CPU USAGE/CLOCK TIME

core size = 32 pages, total pages = 29



time	I/O	no I/O
160	21%	67%
320	33%	77%
480	33%	
640	33%	

fig.26C

RESPONSE TIMES FOR 4 JOBS LESS THAN CORE SIZE

	No I/O	I/O	actual CPU time used
Job 1	45	65	10
Job 2	73	137	32
Job 3	134	330	97
Job 4	61	118	29

It can be seen from fig.26A that when the jobs are issuing I/O requests there is a high degree of drum activity throughout their execution. Whereas when no I/O requests are issued there is high drum activity while the initial pages of the jobs are loaded onto drums which falls off rapidly once jobs start executing. Fig.26B shows that the percentage CPU usage remains very low throughout the jobs' execution when the jobs are issuing I/O requests. This illustrates the fact that I/O bound jobs have a low computing demand on the system, thus creating a situation in which the CPU is idle for a large amount of the time.

Next we investigate two runs in which the total number of pages required by jobs is greater than the number of pages available in core. One set of jobs issue I/O requests and the other does not.

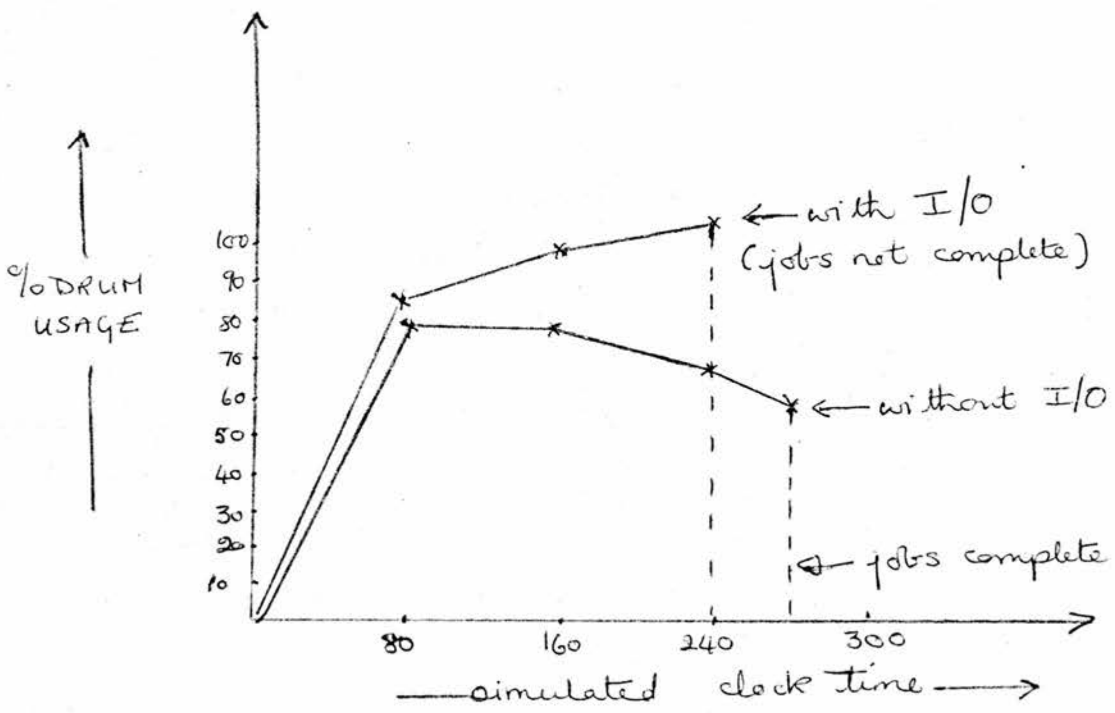
Fig.27A illustrates that the percentage drum usage for jobs issuing I/O requests is always higher than when they do not. However, drum activity is high in both cases, due to the excessive paging which is taking place. For the 4 jobs with no I/O 40 page faults have occurred after 240 units of simulated time with 20 pages having been removed from active working sets and 3 jobs completed. When I/O requests are being issued, the same jobs under the same conditions have issued only 22 page faults with 5 pages being removed from active working sets and only 1 job completed after the same 240 units of simulated clock time. The percentage CPU usage is consistently less (see fig.27B) for jobs whose total pages are greater than core size than for

4 JOBS WITH TOTAL PAGES GREATER THAN CORE SIZE

fig.27A

DRUM USAGE/CLOCK TIME

core size = 16 pages, total no. of pages = 29

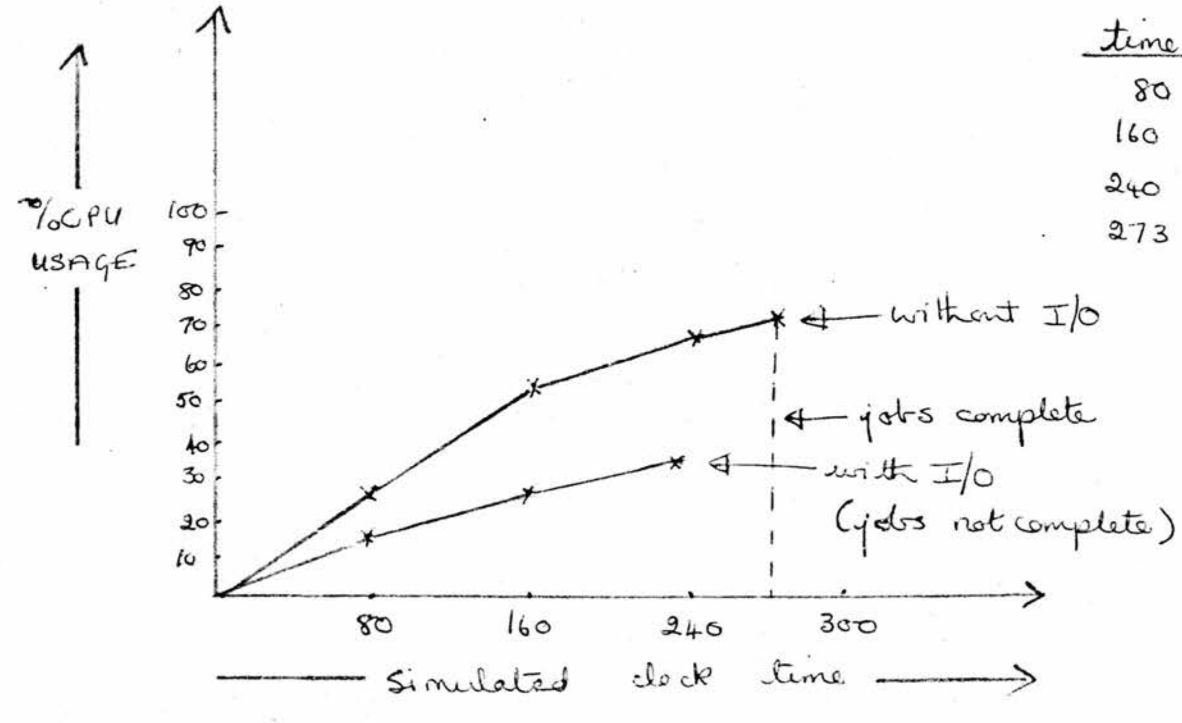


time	I/O	no I/O
80	84%	79%
160	92%	78%
240	93%	60%
273		52%

fig.27B

CPU USAGE/CLOCK TIME

core size = 16 pages, total job pages = 29



time	I/O	no I/O
80	14%	27%
160	22%	52%
240	28%	62%
273		66%

those jobs whose total pages are less than core size (see fig.26B). It can also be seen from fig.27B that jobs which issue I/O requests and have total pages greater than core size have a very low CPU usage.($< 30\%$)

b. Response to different machine configurations

Lastly, figs. 28A and 28B show a comparison of 2 runs in which no I/O requests are issued and all conditions are identical except core size. In both cases there are a total of 29 pages in the system, one run has 16 pages of core and the second has 32 pages of core. As one would expect the run with only 16 pages has a high percentage drum usage (see fig.28A) and a lower percentage CPU usage (see fig.28B) throughout the whole run, since vigorous page-swapping is occurring. The comparison further illustrates the fact that when jobs fit comfortably into core their response times are lower than when they do not (see fig.28C). In the case with 32 pages of core the 4 jobs completed in 243 units of simulated clock time whereas with only 16 pages it took 273 units to complete the same 4 jobs. Further with 32 pages of core 22 page faults occurred and no pages were removed from active working sets whereas with 16 pages 40 page faults occurred and 20 pages were removed from active working sets.

4 JOBS RUN WITH DIFFERENT CORE SIZES

DRUM USAGE/CLOCK TIME

fig.28A

total pages in 4 jobs = 29, no I/O issued

	CORE SIZES	
time	16	32
80	79%	
160	78%	65%
240	60%	44%
273	52%	

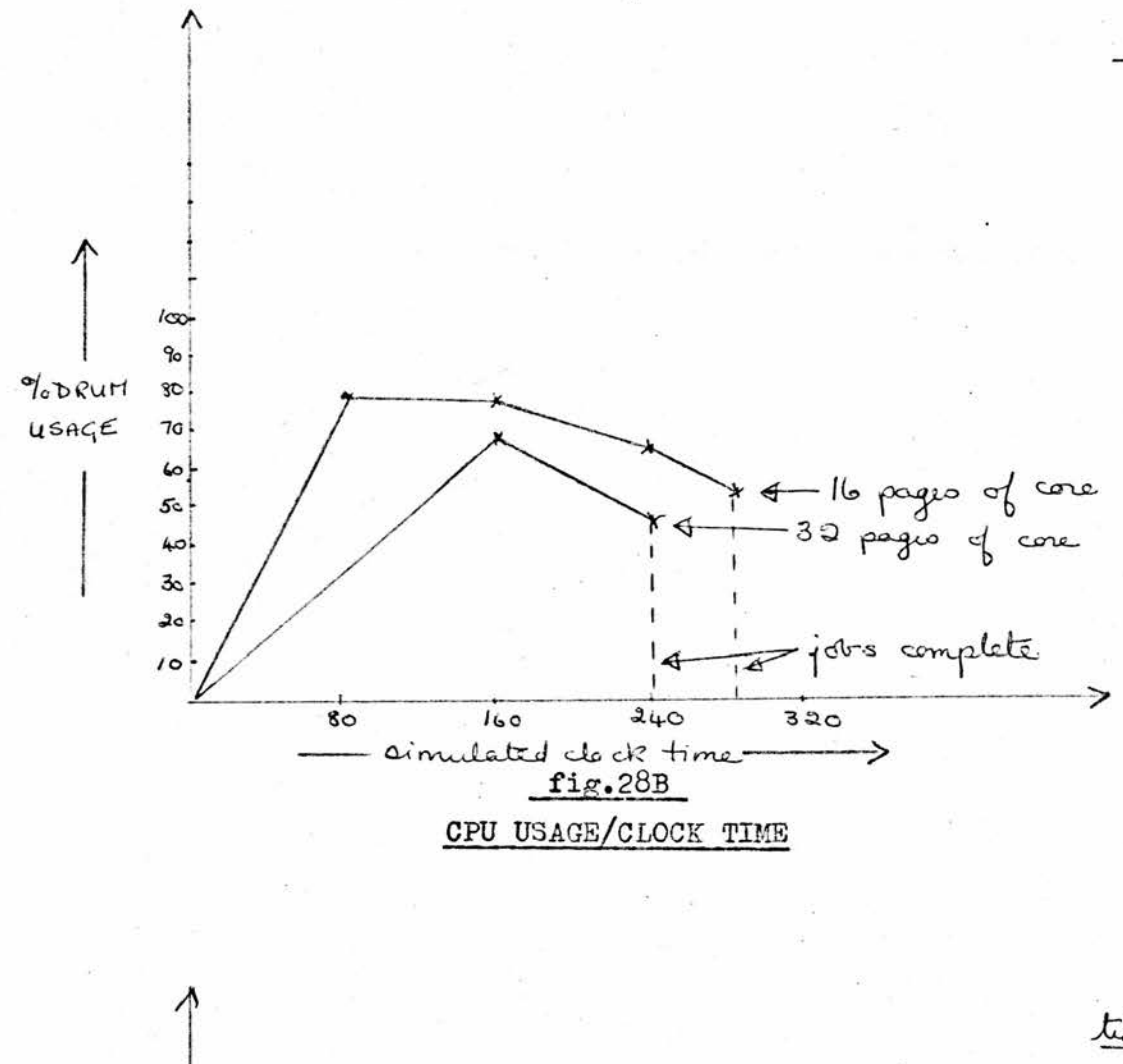


fig.28B

CPU USAGE/CLOCK TIME

	CORE SIZES	
time	16	32
80	27%	
160	52%	67%
240	62%	77%
273	66%	

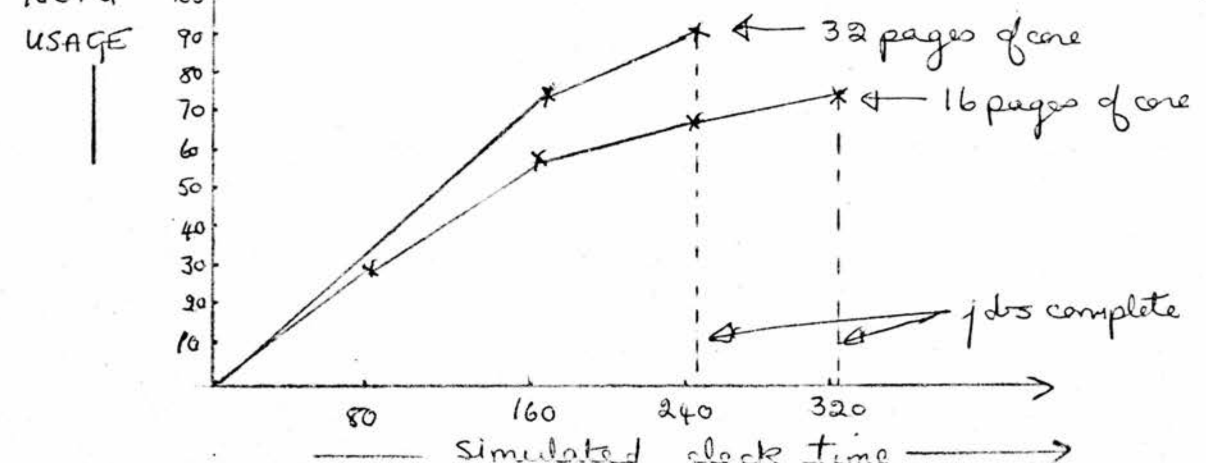


fig.28C

4 JOBS RUN WITH DIFFERENT CORE SIZES

RESPONSE TIMES

	16 pages	32 pages of core
Job 1	45	45
Job 2	91	73
Job 3	155	134
Job 4	77	61

c. Response to a paging algorithm

The paging algorithm used in the examples in fig.25A to fig.28B was the Random Selection Algorithm described in Part IV 1.c. By considering the effects of altering core size on page traffic and CPU utilization we can see that the algorithm is having the expected effect. By reducing core size on the same job stream it has been shown that drum traffic is increased and CPU utilization reduced. Further, the number of page faults and the number of pages removed from active working sets increase as core memory size is reduced.

III 4. A Comparison of Two Paging Algorithms

Here we consider a comparison of two runs which differ only in the paging algorithm used. These are the Least Recently Used (LRU) algorithm and the Random Selection algorithm. Both these algorithms are described and listed in Part IV 1.(c).

It has been shown that the LRU algorithm reduces drum activity. This appears to be caused by the reduction in page traffic since only 13 pages were removed from active working sets when the LRU algorithm was used compared with 20 pages using the Random Selection algorithm. The LRU algorithm does not cause as great a demand on the system facilities as does the Random Selection paging algorithm.

The system under consideration did not issue I/O requests, it had only 16 pages of core memory and had a total of 29 pages belonging to jobs. The execution time on the IBM 360/44 for the LRU run was 8.23 mins compared with 9.23 mins for the Random Selection run.

It appears, therefore, from the runs described in this chapter that the simulation is, in fact, sensitive to changes in system parameters, job types and paging algorithms. Further the results obtained from these changes are consistent with those expected from a paged multiprogramming computer system when subjected to similar changes.

PART IV

THE TEACHING TOOL

IV 1. HOW TO USE THE SIMULATOR

INTRODUCTION

The simulator was seen from the very beginning of its development as a teaching tool. One of its basic requirements was therefore that it should be easy to use. This chapter will illustrate the fact that the simulator is straightforward to use, and define the method of use.

The system was developed using punched cards, the subroutines and main program were then precompiled and stored in a private library of object modules, (on a disk). The user is presented with a set of Job Control Language (JCL) cards for linking his subroutines with the main program.

In order to test the user's system it is sufficient to slot the FORTRAN subroutines into the JCL, which will then compile it, link it to the private library to get the rest of the program, load the whole system and start execution.

There are three subroutines required from the user. These are:

1.	ALGORI	- the paging algorithm
2.	SYSTEM	- the system configuration parameters
3.	JOBSIM	- the job description parameters.

Details and examples of these subroutines may be found later in the chapter.

A factor contributing towards the ease of use of the simulator is that it is written in the high level langauge FORTRAN which is generally known among students of computer systems and is generally supported on a wide range of commercially available computer systems. Thus it is expected that anyone with a working knowledge of FORTRAN will be able to use the simulator.

The teaching tool endeavours to fulfil the needs of a user who requires one or more of the following:-

1. to test a paging algorithm on various configurations of a paged time-sharing system, with a fixed job stream
2. to test a paging algorithm on a fixed configuration but with a varying job stream
3. to test the effects of various paging algorithms on a fixed system configuration with a fixed job stream.

Thus there are three sets of information available to the user so that he has the facility to do any of the above, these are:-

1. the system configuration parameters
2. the job stream parameters
3. the data necessary to write his own paging algorithm.

A variable parameter which does not fall into any of the above sets of information is the length for which the simulation is run. The relevant parameter NEWLIM is the number of units of time after which the simulation will cease and the program terminate. This may be set to any value but it is usual to set it sufficiently high so that all the jobs in the system can run to completion. Further discussion on the use of NEWLIM is given later in this chapter under the heading of OUTPUT.

A discussion on the type of computer system configurable in the simulator has already been held in chapter 3. The remainder of this chapter will therefore be devoted to the description of the three sets of information available to the user.

a. THE SYSTEM CONFIGURATION

There are 10 parameters concerned with the system configuration: these are specified in the subroutine SYSTEM. These parameters are

1. KSIZE - this is the size of main/core memory in K-bytes where K = 1024. see note 1
2. KPAGE - is the size of a page and consequently of a block of memory within the system. This is also in K-bytes, K = 1024. see note 1
3. KSTORE - this is the actual number of blocks of store available on auxiliary storage, (i.e. on drums). see note 2
4. KSAMP - is the number of sampling intervals allowed to pass before the use-bits are shifted (see discussion on the working set in chapter 3 for further details).
5. ZIGMA - is the length of the sampling interval (see chapter 3 discussion on working set).
6. RVTIM - this is the time taken for one drum revolution and is used to help calculate the time taken to complete a drum transfer.
7. RWRATE - is the read/write rate of the drum in bytes/unit of time.
8. TRAVT - is the traverse time of the drum i.e. the time taken to transfer the page from the drum (once it has been located) into main memory. Although it sometimes takes less time to store into

auxiliary memory, using a "first-free-block" algorithm only, than to read from it, TRAVT is regarded to be the same no matter in which direction the page is moved.

9. QANTUM (1) - is the short quantum length which is assigned to a job when it leaves the short quantum level of the ready list. (see chapter 3 on ready list for further details)

10. QANTUM (2) - is the long quantum length which is assigned to a job when it leaves the long quantum level of the ready list. (see chapter 3 on ready list for details)

Note that the parameters involving length of time (namely ZIGMA, RVTIM, TRAVT, QANTUM (1), QANTUM (2))are all standardised to multiples of $^1/100$th sec. That is, the unit of time is chosen to be $^1/100$th sec.

The above parameters may thus be changed by the user within the restrictions indicated in the notes.

<u>note 1</u> Because of the limitations of fixed length declarations in FORTRAN, the size of the page tables and memory map must be fixed. Consequently the total number of blocks of main/core memory allowed in the system at any one instant had to be fixed. The maximum number of blocks is fixed at 256. Thus the result of the division of KSIZE by KPAGE must not be greater than 256.

<u>note 2</u> KSTORE is limited to a maximum of 744 owing to the restrictions of fixed length declarations.

An example of the system subroutine SYSTEM follows:

The declaration and common statements must appear in the subroutine.

```
SUBROUTINE SYSTEM (NEWLIM)

REAL GONE(8), GTWO(8)
INTEGER NOPF(8), BYTREQ(8), NOPAG(8), IOMAX(8)

     COMMON  GONE, GTWO, NOPF, NBLOC, KSTORE, NJOB, MAXINT,

1 MAXBYT, MAXCPU, MAXREC, BYTREQ, NOPAG, IOMAX, TT, ACTIVE, CURRST,
2 NOPR, PTAB, PTBP, WORKST, KSIZE, KPAGE, RVTIM, RWRATE, QANTUM,
3 KSAMP, ZIGMA, MMAP, JOBDES, NFB, FBLST, FBLEND, TRAVT

INTEGER*2  NEWLIM, TT(8), ACTIVE, CURRST(8), NOPR(8),
1 PTAB (1024,4), PTBP(8), WORKST(8), KSIZE, KPAGE, RVTIM, RWRATE,
2 QANTUM(2), KSAMP, ZIGMA, MMAP(1280), JOBDES(8,5), NFB,
3 FBLST, FBLEND, TRAVT


C KSIZE is the size of core memory in K-BYTES (K=1024)
     KSIZE = 128
C KPAGE is the size of a  page in K-BYTES
     KPAGE = 4
C KSTORE is the no. of blocks of auxiliary memory ( ≤ 744)
     KSTORE = 744
C KSAMP is no. of sampling intervals before sampling of use-bits
     KSAMP = 10
C ZIGMA is length of sampling interval
     ZIGMA = 2
C RVTIM is time taken for one drum revolution
     RVTIM = 1
C RWRATE is read/write rate of drum, bytes/millisec
     RWRATE = 156
C TRAVT is traverse time of drum
     TRAVT = 1
C QANTUM (1) is length of a short time quantum
     QANTUM(1) = 40
C QANTUM(2) is length of a long time quantum
     QANTUM(2) = 100
C NEWLIM is no. of units of simulation time required
     NEWLIM = 2000
     RETURN
     END
```

b. THE JOB STREAM

The job stream description parameters are divided into two types. Namely, those which are specified as maximum values for the whole job stream and those which may be specified as individual values for each job. The subroutine JOBSIM incorporating the complete set of job description parameters is called from EVENT 1 in the simulator.

First, the parameter which specifies the total number of jobs to be simulated is NJOB. The number of jobs capable of being held in the present version of the simulator is limited to 8. Thus NJOB may have any value from 1 to 8 inclusive.

Now, the job description parameters which are specified as a maximum value are:-

1. MAXINT - this is the predicted maximum inter-arrival time of jobs to the system.*
2. MAXBYT - this is the maximum number of bytes required by the largest job i.e. the predicted maximum size of jobs.
   (The number of pages required by the job is calculated within the program according to the value of KPAGE)
3. MAXCPU - this is the maximum number of units of time required by the longest job i.e. the predicted maximum length of jobs.
4. MAXREC - this is the maximum record size of a job. This is used when calculating the total time taken for a drum transfer. Drum transfers are often of fixed length records and the option is available to the user to make the records of variable or fixed length.

The individual byte requirements and CPU times of a job may also be specified, details are given in the examples later in the chapter.

* The inter-arrival time to the next job is uniformly distributed between 1 and MAXINT.

The following parameters must be specified as individual values for each job:-

1. IOMAX(PJ) where PJ is the number of the job (see note 1)

   IOMAX(PJ) represents the maximum interval between which I/O (input/output) requests are made by a job, i.e. this is used to predict how much I/O a job is to do whilst in the system. Thus if IOMAX(PJ) is small then a lot of I/O is done, whereas if it is large compared with the total CPU time required by the job then very little I/O is done.

This does not mean, however, that the I/O requests will occur at evenly spaced intervals since IOMAX(PJ) is a <u>maximum</u> value for job PJ and the actual intervals between I/O requests will vary. This is possible since every time an I/O request is predicted within the model, its predicted time of occurrence is taken to be the present value of the processor time (see note 2) of the job PJ plus a random number generated between 1 and IOMAX(PJ).

Note that if the user wants all jobs to do roughly the same amount of I/O and 50 units of time is a suitable interval, then simply specify

IOMAX(PJ) = 50

(IOMAX(PJ) may have any non-zero integer value).

2. GONE(PJ), GTWO(PJ)
   $0 \leqslant$ value $\leqslant 1$ - These specify the rates at which page faults will occur for the job PJ. They represent the gradients of the two lines along which page faulting is predicted to occur as described in Part II chapter 2 section (d).

   For example

   GONE(PJ) = 1
   GTWO(PJ) = 0.25

   gives a page faulting rate along the line whose gradient is 1 until the working set of job PJ is fully core resident then a page faulting rate along the line whose gradient is 0.25 after the working set is in core.

3. JOBDES(PJ,1) - This variable holds the priority of the job, which is such that 1 is the highest priority and larger integer values represent lower priorities. JOBDES(PJ,1) indicates the priority of job PJ in the central memory queue and within each level of the ready list.

   If all the jobs are given equal priority then the queues are ordered in a first-in-first-out basis.

   See later for example of subroutine JOBSIM.

Thus the user has intimate control over the I/O, page faulting and priority of individual jobs giving him the ability to model various types of job stream.

Now that we have seen how to vary the system configuration and the job stream the next topic to consider is the paging algorithm.

<u>note 1</u> Each job is allocated a number from 1 to NJOB according to its order of arrival to the system i.e. JOB(1) arrived first, JOB(2) arrived second, and so on.

<u>note 2</u> The processor time of a job is the time of the job on the processor seen without any interruptions i.e. CPU time actually used by the job.

```
SUBROUTINE  JOBSIM(PJ)
REAL GONE(8), GTWO(8)
INTEGER NOPF(8), BYTREQ(8), NOPAG(8), IOMAX(8), PJ
INTEGER JM(101)
INTEGER*2 TT(8), ACTIVE, CURRST(8), NOPR(8), PTAB(1024,4),
1 PTBP(8), WORKST(8), KSIZE, KPAGE, RVTIM, RWRATE, QANTUM(2),
2 KSAMP, ZIGMA, MMAP(1280), JOBDES(8,5), NFB, FBLST, FBLEND,
3 TRAVT

COMMON GONE, GTWO, NOPF,NBLOC, KSTORE, NJOB, MAXINT, MAXBYT,
1 MAXCPU, MAXREC, BYTREQ, NOPAG, IOMAX, TT, ACTIVE, CURRST, NOPR,
2 PTAB, PTBP, WORKST, KSIZE, KPAGE, RVTIM, RWRATE, QANTUM,
3 KSAMP, ZIGMA, MMAP, JOBDES, NFB, FBLST, FBLEND, TRAVT
```

```
C NJOB is number of jobs to be simulated in 1 run of program
     NJOB = 1

C MAXINT is maximum interarrival time of jobs
     MAXINT = 250

C MAXBYT is maximum byte size of any job in the system
C BYTREQ(PJ) is byte requirements of a particular job
             MAXBYT = 100000
             MAX    = MAXBYT
             CALL RANN2(JM,MAX)                ( note (a) )
             BYTREQ(PJ) = JM(PJ*2+10)

C MAXCPU is maximum processor requirements of any job in the
       system
C JOBDES(PJ,4) is processor requirements of a particular job
             MAXCPU = 1000
             MAX    = MAXCPU
             CALL RANN2(JM,MAX)                ( note (b) )
             JOBDES(PJ,4) = JM (PJ+5)

C MAXREC is maximum record size of any job
C JOBDES (PJ,5) is individual record size of a particular job
             MAXREC = 200
             MAX    = MAXREC
             CALL RANN2(JM,MAX)                ( note (c) )
             JOBDES (PJ,5) = JM(PJ+7)

C IOMAX(PJ) is maximum time interval between I/O requests for
           a particular job
             IOMAX (PJ) = 1000                 ( note (d) )

C GONE(PJ) is page fault rate upto the working set
                                (O   GONE(PJ)   1)
             GONE(PJ) = 1

C GTWO(PJ) is page fault rate after working set is reached
             GTWO(PJ) = 0.25

C JOBDES(PJ,1) is the priority of a particular job (integer 1-9)
             JOBDES (PJ,1) = 1                 ( note (e) )

C JOBDES(PJ,3) is the long quantum count of a particular
             job (integer 1-9)
             CALL RANN2(JM,10)
             JOBDES(PJ,3) = JM(PJ*2)           ( note (f) )

             RETURN

             END
```

note (a)

MAXBYT must be specified. The individual byte requirements in the above example for a particular job will be some random number between 1 and MAXBYT.

Further examples:

1. Suppose the user wishes all jobs to be the same size then the instructions

```
  MAXBYT = 10000    say
  BYTREQ(PJ) = 10000  will have this effect.
```

2. For half the jobs to be large and half to be small

```
  MAXBYT = 10000

IF (PJ.EQ.(PJ/2)*2)BYTREQ(PJ) = 10000
IF (PJ.NE.(PJ/2)*2)BYTREQ(PJ) = 500
```

note (b)

MAXCPU must be specified. The individual processor requirements in the above example for a particular job will be some number between 1 and MAXCPU.

Further examples:

1. Suppose the user wishes one job in the system to be very long and the rest to be short jobs; e.g. job 1 is to be long

then

```
  MAXCPU = 10000
IF (PJ.EQ.1)JOBDES(PJ,4) = 10000
IF (PJ.NE.1)JOBDES(PJ,4) = 50.
```

2. For all the jobs to be the same length

```
  MAXCPU = 1000
  JOBDES(PJ,4) = 1000
```

<u>note (c)</u>

MAXREC must be specified. The example shown gives individual record size of some number between 1 and MAXREC, to a particular job.

Further example:

Often drum transfers are of fixed length records, this situation is catered for by the statements

```
          MAXREC = 200
          JOBDES(PJ,5) = 200
```

This will make all jobs have record lengths of 200.

<u>note (d)</u>

the statement IOMAX(PJ) = 1000 means that every job in the system will have the same maximum time interval between I/O requests. Two further examples follow:-

<u>example 1</u>

Suppose the user wishes half the jobs to be I/O bound and the other half relatively free from I/O, then for MAXCPU of say 1000 the statements

```
   IF (PJ.EQ.(PJ/2)*2)IOMAX(PJ) = 20
   IF (PJ.NE.(PJ/2)*2)IOMAX(PJ) = 500
```

will have this effect.

<u>example 2</u>

Suppose the user wishes the maximum time interval between I/O requests to be some random interval between 1 and the MAXCPU requirements then the statements

```
      MAX = MAXCPU
      CALL  RANN2(JM,MAX)
      IOMAX(PJ) = JM(PJ*4)
```

would have this effect.

note (e)

The statement JOBDES(PJ,1) = 1 gives all the job the same priority. Two further examples follow:-

example 1

Half the jobs have high priority and the other half with low priority. This could be required in a system where the program is simulating tape jobs and disk jobs.

```
IF(PJ.EQ.(PJ/2)*2)JOBDES(PJ,1) = 2
IF(PJ.NE.(PJ/2)*2)JOBDES(PJ,1) = 8
```

example 2

Assigning random priority to jobs may be done by the statements

```
CALL  RANN2(JM,10)
JOBDES(PJ,1) = JM(PJ*3)
```

note (f)

The long quantum count in the given example will be some random number between 1 and 9, for a particular job.

further example:

For all jobs to have the same long quantum count, the statement

JOBDES(PJ,1) = 4, say, will have this effect.

c. THE INTERFACE WITH THE PAGING ALGORITHM

The principle consideration of this interface is that the user should be able to write his own paging algorithm with a minimum of programming effort.

First of all, we must make clear what, in fact, the function of the paging algorithm is:-

The paging algorithm is invoked when a process makes a reference to a datum in a part of its logical address space that does not immediately map onto the physical main memory of the machine. The task of the paging algorithm is then to find some "unused" physical space, load the appropriate section of logical address space into the freed physical area and specify the mapping (i.e. cause the page tables to be suitably modified to show the mapping).

It may be that there are some free (unused) blocks still in core, in which case the page is allocated to the first free block without reference to the paging algorithm. However, if there are no free blocks then the paging algorithm must be referenced. The function of the paging algorithm is to decide which block in core may be overwritten or must be rolled out onto drum in order that space may be allocated to the demanding page.

The user decides on the strategy upon which the paging algorithm will base its decisions. The paging algorithm is in the form of a subroutine and various sets of information may be accessed by it. Thus, the strategy within the paging algorithm may be based upon information about individual processes or of the general condition of the system. Although much information is available to the user via the subroutine none of it should be altered by him. The only value that the user is free to change is that of the variable BLCKNO which represents the positioning of a block in core. That is, the block which has been chosen to be rolled out to drum (if it has been written to) to make way for the incoming page.

Next follows a list of all the information available to the paging algorithm and consequently to the user. Later follow two examples of typical paging algorithms.

The subroutine has the following information available to it:-

ACTIV	- is a count of the total number of pages removed from active working sets upto the present time. (i.e. the total for all jobs in the system)
NOPAGR(J) (J=1,8)	- is a count of the number of pages removed from job J's working set upto the present time
CURSET(J) (J=1,8) (CURRENT SET)	- is a count of the number of pages of job J which are presently resident in core.
WOKSET(J) (J=1,8) (WORKING SET)	- is a count of the number of pages presently in job J's working set.
NBLOCK	- is the maximum number of blocks of core available

PAGTAB(I,K)
(I=1,1024,K=1,4)
(PAGE TABLE)

K =	1	2	3	4
	"IN-CORE" BITS	"USE-BITS"	BLOCKNO	DRUMNO
1024				

The table has one set of entries for each page of a job.

"IN-CORE" BITS = 0 or 1, 1 if page is in core
0 otherwise

"USE-BITS" - this is an ageing mechanism for pages resident in core.
(see notes on working set for details)

BLOCKNO - location of the block in core in which the page is held

DRUMNO - location of the block on auxiliary storage on which the page is held.

PTABBP(J) - represents the page table base pointer for
(J=1,8)
job J. This points to the base address of information concerning job J in the page table.

For example, to find the information about page 4 of job 3

calculate I = 4+PTABBP(3)

then PAGTAB(I,K) ,K = 1,4 contains the information about this page.

MYMAP(I)

I = 1,1280
(MEMORY MAP)

BLOCKNO	No.OF JOB	DRUMNO	0 or 1	NEXT POINTER
this is the block in which the page is held	this is the number of the job to which the page belongs	this is the position on the drum from which the page was taken	this indicates whether or not a page has been "written to" since it was taken from the drum. 1 if it has 0 otherwise	for free-blocks list also used to indicate if the block is a candidate for removal from core. (set to -10 if it is)

The memory map table contains a set of 5 entries for each block of core. This set of entries, though presently at 5, is variable and is held in a parameter called WIDTH.
In the current model, the memory map has a maximum dimension of 1280 which represents the product of the maximum number of blocks of core available and the WIDTH of the memory map.

WIDTH - is also given as a parameter in case it is required to hold more information in the memory map, (this is also a parameter of the main simulation)
(at present WIDTH = 5 so maximum no. of blocks = 256)

BLCKNO - this is the variable most essential to the user since it is used to pass back to the main simulation the location (number) of the block chosen by the paging algorithm i.e. the block which will be rolled out to make way for a demanding page.

EXAMPLE PAGING ALGORITHMS

1. RANDOM SELECTION STRATEGY WITH WORKING SET CONSIDERED

The general philosophy of this strategy is that if there is a candidate for removal* from core (i.e. an entry in column 5 of the memory map whose value is -10) then this block is chosen to be rolled out. If such a candidate does not exist then a block is simply selected at random from all the blocks of core.

*(see notes on working set for further details)

The next page contains the FORTRAN code necessary to implement this paging algorithm. Statements marked ** must appear in all paging algorithm subroutines.

```
**  SUBROUTINE ALGORI (BLCKNO, ACTIV, PAGTAB, PTABBP, WOKSET,
                       CURSET, NOPAGR, MYMAP,NBLOCK, WIDTH)

**  INTEGER*4 PAGTAB(1024,4), ACTIV, PTABBP(8), WOKSET(8),
              CURSET(8), NOPAGR(8), MYMAP(1280)

**  INTEGER  WIDTH

    C additional declarations

    INTEGER*4  ADR

    C scan column 5 of memory map to see if there is a candidate
      for removal

    C i.e. find the first entry which equals -10.

    DO  1  I = 1,NBLOCK

    ADR =   WIDTH *(I-1)+1

    BLCKNO = I

    IF(MYMAP(ADR).EQ.-10)GO TO 3

1   CONTINUE

    C if there is no candidate for removal then select a block
      at random

    C SUBALG chooses a random number between 1 and NBLOCK and
      places it's value in BLCKNO

    CALL SUBALG(BLCKNO, NBLOCK)

3   CONTINUE

**  RETURN

**  END
```

2. THE LEAST RECENTLY USED STRATEGY (LRU)

The LRU algorithm uses the information gathered through the implementation of the working set philosophy. First of all, if a page has been marked as a candidate for removal from core, i.e. if it has left the working set but is still in core, then this page is rolled out of core to make way for the demanding page. If no such page exists then the "use-bits" of each page in core are examined and the page whose "use-bits" are most right-justified is chosen. This is the page which has been in core for the longest time without being referenced, although it is still in the working set. Thus the least recently used page is selected to be rolled out of core to accommodate the demanding page.

The following page lists the code necessary to implement this paging algorithm.

2. THE LEAST RECENTLY USED STRATEGY

```
** SUBROUTINE ALGORI (BLCKNO, ACTIV, PAGTAB, PTABBP, WOKSET,
                      CURSET, NOPAGR, MYMAP, NBLOCK, WIDTH)
** INTEGER*4  PAGTAB(1024,4), ACTIV, PTABBP(8), WOKSET(8),
                     CURSET(8), NOPAGR(8), MYMAP(1280)
** INTEGER  WIDTH
   C additional declarations
   INTEGER *4  ADR
   C scan column 5 of memory map to see if there is a candidate
     for removal
   C i.e. find the first entry which equals -10
     DO L  I = 1,NBLOCK
     ADR  = WIDTH*(I-1)+1
     BLCKNO = I
     IF (MYMAP(ADR).EQ.-10) TO TO 3
1  CONTINUE
   C if there is no candidate for removal then remove the page
     that was least recently used
   C This is the page whose "use-bits" are most right justified
     i.e. numerically smallest
   C If the page is in core then compare its use-bits
     K = 0
6    K = K + 1
     IF(PAGTAB(K,1). NE.1) GO TO 6
     IF(PAGTAB(K,2).LE . 0) GO TO 6
     DO  4  J = 1,1024
     IF (PAGTAB (J,1).NE.1) TO TO 4
     IF (PAGTAB (J,2).LE.0) GO TO 4
     IF (PAGTAB (J,2).LT.PAGTAB (K,2) ) K = J
     4 CONTINUE
       BLCKNO = PAGTAB (K,3)
     3 CONTINUE
**     RETURN
**     END
```

3. THE RANDOM SELECTION STRATEGY

This paging algorithm works simply on the philosophy that if there is no block of core available for the demanding page, then a page is simply rolled out of core at random and the corresponding block allocated to the demanding page. The following code has this effect:

```
** SUBROUTINE ALGORI (BLCKNO, ACTIV, PAGTAB, PTABBP, WOKSET,
                      CURSET, NOPAGR, MYMAP, NBLOCK, WIDTH)

** INTEGER*4  PAGTAB(1024,4), ACTIV, PTABBP(8), WOKSET(8),
              CURSET(8), NOPAGR(8), MYMAP(1280)

** INTEGER  WIDTH

   C completely random strategy

   C SUBALG selects a random number between 1 and NBLOCK
            and places its value in BLCKNO

     CALL SUBALG (BLCKNO, NBLOCK)

** RETURN

** END
```

d. THE OUTPUT FROM THE SIMULATOR

Output from the simulator takes on three different forms namely, descriptive, histogram and tabular.

The output endeavours to give a step by step picture of the state of the system being simulated. Statistical information is collected at evenly spaced intervals based on the value of NEWLIM, where NEWLIM is the maximum number of units of time for which the simulator will run. NEWLIM is chosen by the user.

The first set of output is descriptive. It describes the system configuration and the job stream as specified by the user. This is output once only.

The output which is described below is output at intervals of $^{1}/10$th of NEWLIM. Thus if the simulation runs until NEWLIM is reached* then 10 sets of output will have been given.

These 10 sets of output each consist of 4 histograms, NJOB**** tables and a descriptive summary. Suppose NEWLIM = 9600** units of time then these sets of information are given at intervals of 960 units.

Let this interval of 960 be called LIMIT, then LIMIT = NEWLIM/10.

* Note that a simulation may be terminated before NEWLIM is reached since all the jobs in the system may have completed. The simulation and the program terminate either at NEWLIM or when all the jobs are complete, whichever occurs first.

** Note that NEWLIM needs to be a multiple of 80 so that the sampling can be done at evenly spaced intervals. However <u>any value</u> may be given by the user for NEWLIM and it is rounded to the nearest multiple of 80 within the program.

**** NJOB is the number of jobs in the system. NJOB is chosen by the user i.e. maximum no. of jobs active simultaneously i.e. degree of multiprogramming.

The first three histograms given at LIMIT represent the state of queues in the system over the last LIMIT units (i.e. 960 units). The queues are sampled at 80 evenly spaced intervals within LIMIT (i.e. at 0, 12, 24, ...960). The queues represented by the histograms are:

1. the queue for central memory space
2. the queue for drum attention
3. the ready list.

The height of the histogram at any one point represents the number of jobs in the queue at the end of the interval (LIMIT/80).

The fourth histogram represents the number of free blocks of central memory still available. This histogram is scaled*** down so that it conforms with the other histograms in general appearance. While it does not give an exact account of the actual number of freeblocks, it does show the general trend of the free blocks still available.

The NJOB tables given at LIMIT are such that each table represents the state of one particular job during the last LIMIT units of time. There is one table per job. This table is built up from the statistics about a job gathered at 20 evenly spaced intervals within LIMIT. (The statistics are sampled at the end of each interval).

*** The method of scaling is that the height of the histogram (H) at any one point = number of free blocks (NFB) divided by (the integer result of NFB/20) plus 1
i.e. H = NFB/I(NFB/20)+ 1

Each table contains the following information about a job:-

1) its arrival time to the system

2) the time at which its first page was loaded into core.

Then at each interval of LIMIT/20 it gives:-

1) the clock time

2) the time the job has been in the system. (this timing starts after the job's first page has been loaded into core)

3) the processor time of the job (this is the number of units of time the job has had control of the processor excluding any interruptions)

4) the number of pages of the job loaded onto drums i.e. the total logical address space demand

5) the current set of pages in core (i.e. the number of pages currently loaded)

6) the number of active pages removed (i.e. the number of pages removed from core while still in the job's working set)

7) present number of I/O requests that have been issued by the job

8) present number of page faults issued by the job

9) the time quantum still left to run on the processor, (this quantum was the one allocated to the job when it left the ready list)

10) the current value of the long quantum count.

Then at the end of each table, the following 3 pieces of information are given. These indicate the state of the job at the end of the interval i.e. <u>at</u> LIMIT. These are:-

1) whether the job is complete - if it is then the response time for the job is given
2) number of units of time for which the job was blocked for I/O
3) number of units of time for which the job was blocked for page faults.

Finally after the histograms and tables a descriptive summary is given of the state of the jobs and the system at the end of LIMIT units of simulated time.

This contains:-

1. The processor idle time upto the present time (i.e. a cumulative value)
2. The total number of units of time simulated so far
3. The total number of pages removed from active working sets (a cumulative count)
4. The number of jobs put through the system i.e. NJOB
5. The number of jobs completed so far (a cumulative value)
6. The current number of page faults issued by all jobs (i.e. a cumulative count)
7. The percentage CPU usage and percentage drum usage over past LIMIT units of time
8. The overall CPU usage and average drum usage. (This is a cumulative percentage).

These histograms, tables and descriptions are produced every NEWLIM/10 units of time. Except where indicated the statistics given are not cumulative, but are pseudo-continuous. This means that, for instance, each histogram of the drum queue placed side by side in the order in which they are output would produce a continuous picture of the conditicr of the drum queue throughout the simulation. The same may be said of all the histograms and the job tables for a particular job.

(Note that all values given are those of a sample taken at the end of the interval in question).

PART V

CONCLUSIONS and FUTURE DEVELOPMENTS

V. CONCLUSIONS AND FUTURE DEVELOPMENTS

The development of the simulation was greatly assisted by the implementation of a simple model (BASYS) which established the basic structure and notions involved in the final model.

A straight forward expansion of this simple model has however caused several problems, although on the whole, it is felt that this was a good approach.

The main problem arose from the development of the much larger FORTRAN main program which represents the model of our paged multiprogramming computer system. The program development stages could, retrospectively, have been improved. Most of the subroutines used (there are 14 subroutines in all, with an average of 44 FORTRAN statements) were debugged independently using the on-line VDU terminals available under the RAX system on the IBM 360/44. The main program, however, was written in one continuous piece of code. This was a large main program (726 FORTRAN statements) and contained the seven-event structure of the simulator, and was consequently difficult to debug. It is now felt that some way of breaking down this main program into independent modules for testing should have been developed. Perhaps, event one should have been written and tested then event two added on and tested then event three and so on. It could be argued that FORTRAN subroutines could be used to simulate each event, but since there is a great deal of interaction between the seven events the overhead involved in the subroutine parameter passing would make the program even bigger and execution time slower than at present. Alternatively, all parameters could be passed in COMMON but this raises its own problems of clarity of design and security.

The program was originally compiled under the F-level compiler which requires less storage space than the G-level but this did not contain the debugging aids required by the program. Further, to reduce the storage requirements of the program INTEGER 2 (half-words) variables were used wherever possible.

Eventually, however, the program needed to be run under the G-level compiler so that array subscript checking could be carried out. The DEBUG option SUBCHK was used to ensure that no array subscript overflow occurred during the program which rectified itself later giving apparently good results in the simulation. This compiler uses too much storage when compiling the large main program that it must run in full-core i.e. 200K, resulting in a slow turnaround of jobs. Further, it was not discovered until after nearly six months debugging the program that the FORTRAN G-level compiler contained a bug (generally unknown) which is related to the use of half-words in DO LOOPS. This caused intermittent errors and array subscript overflow errors which varied from run to run, and considerable delay was experienced in pursuing this apparent simulator error. Therefore, we ran the program under the F-level compiler whereupon it ran to completion free from subscript overflow errors but without the added confidence provided by the DEBUG SUBCHK option. At this point all half-words were removed from the program and it was again compiled under the G-level compiler, and debugging of the program could recommence.

Despite the problems encountered in debugging the program it is still felt that FORTRAN was a reasonable choice of language since it has allowed different queueing techniques to be used at differing points in the model, and it is generally known among students. In general, the simulation model satisfies the requirements for which it was originally written.

Although no attempt has been made to provide conclusive evidence of the validity of the model, it should be clear from the arguments and results in Part III Ch.1 and the general trends of the results in Part III Ch.3 that the model provides a realistic representation of paged multiprogramming computer systems. The model has been shown to be valid for one particular system since on the basis of known input data, the results from the simulation compared favourably with known output data. Further the model has been shown to be sensitive to changes in system configuration and job description parameters and to various paging algorithms. It must be realised however that the simulation is only valid to a certain level of detail, for example, a job with references scattered throughout its pages and only a small CPU time cannot be accurately represented. (see Part II, 2.b. LEVEL OF DETAIL)

The simulator has exact reproducibility since using the same parameters always gives the same results. The

IBM 360/44 CPU time necessary to run to simulation varies directly with the amount of drum activity within the simulation, for example, on two runs where jobs total pages fit into core in the run where jobs are free from I/O requests the execution time was 6 mins whereas in the run with I/O the execution time was 24 mins. The real execution time also varies directly with the length of the jobs within the simulation.

The present state of the simulation gives the user a means of developing and testing new paging algorithms under varying conditions of system configuration and job stream.

FUTURE DEVELOPMENTS

It was not possible within the time available for development of this simulation to include all the facilities that one would wish to include. However in the future it is hoped to make improvements in terms of the size and execution speed of the program. Those subroutines which are called most frequently should be rewritten in IBM PL360, the random number generator which was originally written in FORTRAN has already been replaced by one in PL360 resulting in a better execution speed.

Several of the output statistics from the simulator could be improved, histograms should be presented in percentages, and maximum and minimum values of variables could be taken as well as the present sampling which takes place at the end of sampling intervals. Further a user defined sampling interval could be created.

It is intended that the simulator should be available for general student use in the session 1974-75, with possible modifications and enhancements performed by undergraduates as part of their normal project work.

LIST OF REFERENCES

(1) SCHERR A.L.	An Analysis of Time-Shared Computer Systems MAC-TR-18 MIT Project MAC Cambridge, Mass; 1965
(2) SMITH Jnr. E.C.	Simulation in Systems Engineering IBM Systems Journal Vol.1, Sept 1962 pp30-50
(3) NIELSEN N.R.	The Simulation of Time Sharing Systems Communications of the A C M Vol.10 No.7, July 1967
(4) SMITH W.E.	A digital System Simulator Proc 1957 WJCC Institute of Radio Engineers New York pp31-36
(5) ZUCKER M.S.	LOCS: An EDP Machine Logic And Control Simulator. IEEE Transactions on Electronic Computers Vol.14, June 1964 pp403-422
(6) STATLAND N.	Methods of Evaluating Computer Systems Performance Computers and Automation Vol.13 No.2, Feb 1964 pp18-23
(7) KATZ J.H.	Simulation of a Multiprocessor Computer System Proc 1966 SJCC Spartan Books Washington D.C. pp127-139
(8) SEAMAN & SOUCY	Simulating Operating Systems

(9) BOOTE W.P., CLARK S.R., ROURKE T.A.
Simulation of a Paging Computer System
The Computer Journal. Vol.15 No.1

(10) MacDOUGALL M.H. Computer System Simulation: An Introduction
Computing Surveys Vol.2 No.3, Sept 1970

(11) DENNING P.J. The Working Set Model for Program Behavior
Comms of the ACM Vol.11 No.5, May 1968

(12) FINE G.H., JACKSON C.W., McISAAC P.V.
Dynamic Program Behavior under Paging
Proc ACM National Meeting 1966

(13) VARIAN L.C. & COFFMAN E.G.
An Empirical study of the behavior of programs in a paging environment
Proc. ACM Symposium on Operating Systs. Oct 1967

(14) COFFMAN E.G. & VARIAN L.C.
Further Experimental Data on the Behavior of Programs in a Paging Environment
Comms of ACM Vol.11 No.7, July 1968

(15) FRIEBERGS I.F. The Dynamic Behavior of Programs
Fall Joint Computer Conference 1968

(16) Personal Communication

(17) KATZ J.H. An Experimental Model of System/360
Comms of ACM Vol.10 No.11, Nov 1967

(18) BLATNY J., CLARK S.R., ROURKE T.A.
On the Optimization of Performance of Time-Sharing Systems by Simulation
Comms of ACM Vol.15 No.6, June 1972

(19) NAYLOR, BALINTFY, BURDICK, CHU
Chapter 8
Computer Simulation Techniques
Wiley & Sons Publication

(20) MARTIN F.F. Computer Modeling & Simulation
Wiley & Sons Publications

(21) MORRIS D., SUMNER F.H., WYLD M.T.
An Appraisal of the Atlas Supervisor
Procs of ACM National Meeting 1967

(22) SALTZER J.H. Traffic Control in a Multiplexed Computer System
Thesis. July 1966
Massachusetts Institute of Technology
Cambridge, Massachusetts

(23) BELADY L.A. A Study of Replacement Algorithms for a Virtual-Storage Computer
IBM SYSTEMS JOURNAL Vol.5 No.2, 1966.