PAUL MARTIN BENNETT

EVALUATION AND MEASUREMENT OF MULTIPROCESSOR SYSTEMS WITH SHARED MEMORY

SUBMITTED FOR Ph.D. DEGREE ..JUNE 1979:..

THE UNIVERSITY OF ASTON IN BIRMINGHAM

EVALUATION AND MEASUREMENT OF MULTIPROCESSOR SYSTEMS WITH SHARED MEMORY

THESIS SUBMITTED FOR Ph.D. DEGREE ..... JUNE 1979 .....

PAUL MARTIN BENNETT

ABSTRACT

This study is concerned with several proposals concerning multiprocessor systems and with the various possible methods of evaluating such proposals. After a discussion of the advantages and disadvantages of several performance evaluation tools, the author decides that simulation is the only tool powerful enough to develop a model which would be of practical use in the design, comparison and extension of systems.

The main aims of the simulation package developed as part of this study are cost effectiveness, ease of use and generality. The methodology on which the simulation package is based is described in detail. The fundamental principles are that model design should reflect actual systems design, that measuring procedures should be carried out alongisde design, that models should be well documented and easily adaptable and that models should be dynamic.

The simulation package itself is modular, and in this way reflects current design trends. This approach also aids documentation and ensures that the model is easily adaptable. It contains a skeleton structure and a library of segments which can be added to or directly swapped with segments of the skeleton structure, to form a model which fits a user's requirements.

The study also contains the results of some experimental work carried out using the model, the first part of which tests the model's capabilities by simulating a large operating system, the ICL George 3 system; the second part deals with general questions and some of the many proposals concerning multiprocessor systems.

KEY WORDS: MULTIPROCESSOR, SIMULATION, EVALUATION, MODULARITY, DESIGN

ACKNOWLEDGEMENTS

I should like to take this opportunity to show my gratitude to the various people who have assisted me throughout this study. Firstly I must thank the staff, both academic and programming, at the University of Aston Computer Centre; in particular Mr. I. H. Gould, my supervisor, for his advice and guidance throughout the duration of this research, and Mr. E. C. Richards, the systems manager, for his help with the George 3 operating system particulars I required. I should also like to thank my wife, Linda, for her spiritual support and my parents for urging me to this level of study.

CONTENTS

List of Illustrations.

List of Tables.

Chapter 1 INTRODUCTION

1.1. Background.

1.1.1. The need for performance evaluation.

In the past relatively unsophisticated methods were used to assess computer system performance. The speed of processing various types of instructions or instruction mixes or memory cycle times were compared, usually on paper (06) (G4) (J7) (S12) (C1). These methods were probably sufficient in comparing systems where programs were introduced and run to completion before another program was introduced to the system. With the advent of multiprogramming and multiprocessing, parallelism became an integral part of the functioning of systems and their complexity has grown immensely, thus making performance measurement more complex. A more systematic and rigorous approach to the task was therefore required (06). The realisation of this is reflected in the mushroom growth of the interest in computer performance and evaluation. The topic has grown from approximately ten or fewer papers in 1967-68 to the situation today with over one hundred and fifty technical papers per year and a large number of popular and semi-popular articles.

While the cost of particular hardware units within a system has decreased recently the total cost of systems has increased, thus placing more emphasis upon efficient use of resources and hence upon the techniques of performance measurement.

Performance evaluations are required for three main purposes:-

1. Selection of the best among several existing systems - comparison
2. Analysis of an existing system with a view to improvement - extension.
3. Prediction of performance of a not-yet existing system - design.

Hence, an evaluation must be delivered whether a system exists or not.

In the latter case, one of the most difficult problems in the design of large complex systems is, in fact, gaining an insight into the performance of a system before it is implemented. It seems obvious, then

that some technique for predicting the performance of system programs and for evaluating the impact of various design alternatives on that performance must be considered, and that these must be considered alongside the design. It is expensive to build machines based on an intuition of how well it will perform. Intuition can be wrong as is explained by Estrin, Muntz and Uzgalis (E9). To combat this the designer must be provided with predicted performance characteristics so that he can re-design, if necessary, in order to meet his specified performance objectives. This counteracts the tendency of designers to incorporate design features from previous systems without careful assessment of their suitability to new systems. (This is particularly valid today with the general trend away from single processor systems to multiprocessor systems being made). Calingaert (c1) stresses this, "design without evaluation is usually inadequate", and Morrison (M25) points out that the means of understanding the performance consequences of changes in design and configuration, without the expense of implementing them, has still to be improved.

In the case where the systems exist it is the manager, who is responsible for the choice of machine or for efficient and effective use of his existing resources (T4). When computer usage grows to utilize available resources and then continues to grow until the system can no longer supply the demand (as is the case in many installations (L5), the manager's usual response has been to request more financial aid to expand the hardware of the system. Thus, expenditure on computer systems has grown without any regard to whether the existing facilities are being fully utilized. In this kind of attempt to meet the differing demands of his users the manager has sacrificed efficiency.

"Many of today's systems are poorly co-ordinated and have wasteful operations with much idle time and little overlap between various system resources." (W1)

Technical development, then, has reached a point of diminishing returns. Although the hardware is fast, an efficient use of that hardware is rare. In fact, it has been suggested that improving usage on current systems will have significantly greater profit potential than upgrading to fourth generation hardware (W1). This could probably be correct in view of the various examples in which a rigorous performance evaluation effort has improved the effectiveness of computer systems. A report by the U.S. General Accounting Office (U3) provides a good example of the productivity improvements that are possible from its study of NASA's Goddard Spaceflight Centre: Performance evaluation techniques in this case resulted in:-

1. The number of jobs processed in a set time increasing by 50%
2. On another computer the number of jobs processed in a set time increasing by 25%
3. A 10% increase in hours of usage
4. A 7% increase in C.P.U. activity
5. Computer time worth $433,000 annually being saved.

This report and others on the subject (H2) (W3) (E3) (A4) indicate that performance evaluation techniques can lead to an estimated 15 - 30% productivity gain on all types of computers, and that these gains may be made through a variety of means such as a change in software, using faster i/o devices, adding more channels or adding more memory (E2).

To summarize, then, computer performance evaluation can be defined as a design and management tool which involves the process of gathering information and analysing a computer system as an aid to making decisions about that system.

It is used as such a tool in many situations (P7),

1. Systems design
2. System acquisition
3. Changes in configuration
4. Software production
5. System checkout
6. Normal operation
7. Advanced research

and should provide the user with the following facilities (M25)

1. Ability to translate objectives into component performance statements and vice versa

2. Ability to determine how changes to the machine and program architecture will affect system performance

3. Ability to determine the effect on performance of shifting a function to a different component

4. Ability to quantify, on an incremental basis, the performance effects of adding, deleting or changing system functions

5. Ability to determine the performance of a new system from the performance of existing systems

6. Ability of designers and implementers to estimate resource (hardware and software) usage of various design possibilities

7. Ability to get feedback during implementation about actual values of design parameters and resource usage, and a procedure to relate this to objectives and effect any necessary reconciliation

8. Ability to reconstruct after development the rational (performance consequences) supporting decisions during development

9. Ability to estimate the potential performance of a specified combination of hardware, software and workload

10. Ability to determine how much of the potential performance of a given system is being achieved by an installation using that system

11. Ability to assess performance of a given system relative to that of competing or alternative systems.

1.1.2. Computing power

The main goal of designers in creating new equipment is to gain more computing power. This is the same goal as that of the manager in selecting new equipment or extending his present equipment. Computer performance evaluation techniques, then, should measure computing power. But what is meant by the term "computing power"? The interpretations of this can vary and are never precise.

The earlier attempts at computer performance evaluation (O6) (G4) (J7) (S12) (C1) (M15), took hardware times to be a measure of computing power. This is now considered inadequate (T4) (M15) since computing power is a function of both hardware and software. For example, a system which does not utilize a fast machine efficiently and whose organisation results in system overhead and idle times will not be improved by replacing it with a faster machine.

The most common measure of computing power is throughput. This measure takes into account both the hardware and software functions of a system as it gives an overview of the system as a whole. However, it gives no insight into the usages of separate facilities. Edwards (E4) explains the need for a measure of the amount by which competition for particular facilities will degrade the throughput of the whole system.

In support of this from the other viewpoint, Kimbleton (K8), after noting several studies which report separate analyses of CPU usage, scheduling algerithms, i/o models and memory management models, says that performance predictions should not be made in terms of the performance of a subsystem, but in terms of the relationships between these. This is implicit in Boehm and Bells remark (B26).

"A particular difficulty is that what looks like a performance improvement at one level, actually degrades performance at a higher level."

From these observations, it can be seen that computing power cannot be measured simply, but depends upon the performances of individual facilities

in terms of their interrelationships in the context of the whole system. This view is implicit in the Kiviat graphs (K11) (M21) which represent 8 system facilities and their overlaps to give a 'picture' of the overall performance of that system.

In order to measure computing power, therefore, it is necessary to assess various system parameters (06), which are

1. Overall system efficiency (e.g. throughput)
2. C.P.U. utilization (e.g. CPU% busy, idle, system)
3. I/o channel utilization (e.g. active-idle times)
4. Storage device utilization
5. I/o device utilization
6. Queuing statistics
7. Instruction information (e.g. number executed)

in the context of their interrelationships within the whole system.

Another factor important to computing power, besides its various assessment parameters, is the relation of these parameters to the objectives of those who design or use a system. The efficient utilization of a computer system is not an end in itself but is one of a number of means towards improving performance with respect to overall organisational objectives and criteria. It is in considering this that the effect of the trade-offs between parameters becomes important. For example, in order to have 100% channel utilization, C.P.U. utilization may degrade. This example of a trade-off may be acceptable to one computing system but not to another due to the differing objectives of each system.

These objectives, therefore, are necessary in interpreting the results of a computer performance evaluation, for the computing power of a system can only be judged in terms of the degree of success in attaining them.

1.2. The trend in 3rd generation computer systems.

Third generation hardware was initially very expensive and it was this factor which prompted the system designers to utilize the hardware as much as possible. In order to do this the operating system became more complex as it had to deal with sharing resources, multiprogramming and all their consequent problems of allocation, blocking etc. In fact, many of these large systems are now suffering from problems arising from their complexity.

The advent of cheap micro processors and mini computers, however, has now prompted a trend towards multiprocessor and distributed systems, which can provide the same amount of computing power but at less cost, and which has the added bonus of giving more reliability to a system. With these considerations in mind, Whitby-Strevens (W8) (W9) has predicted that such systems will probably replace the large centralized resource - multiplexing computers in many environments. This view is also supported by Enslow (E7).

"There is no question that we will continue to see multiprocessors utilized in applications where high availability and high reliability are of paramount importance."

The primary characteristics of multiprocessor systems which makes them preferable to single processors are:

1. There can be a better utilization of the resources that the customer buys, since the system can be tailored to his needs
2. The incremental modularity of all the major functions (processing, memory, i/o) will provide the customer with a smoother price/performance curve from which to select his equipment
3. Development, production and maintenance costs can all be reduced, since a few of the models of each functional unit will provide a broad range of performance capabilities
4. For special extra large jobs, the multi-processor system organisation may be the only way to assemble sufficient power in any one system at a given point in time.

Examples of the trend towards multiprocessing are becoming more numerous. There is the twin processor Modular One system at Warwick University (W8), the Pluribus system (H6) (05), the Plessey system 250 (W15), the distributed system which Jensen of Honeywell Incorporated is developing (J3),

the system now being implemented by the Digital Equipment Corporation which is based on Wecker's theoretical model (W6) (W7), Farbers Distributed Computer System (F2), (F3), (F4), (F5) (F6) (F7) (F8) (F9) (F10) (F11) (H7), and the Hydra system which is being developed by Wulf at Carnegie-Mellon University (W24) (W25). The latter system runs on a multiprocessor PDP-11 configuration operating with a potentially vast shared memory. The envisaged applications of Hydra imply that there will be a need for multiprocessor operation and part of the Hydra project is to discover the potentials and problems of such systems.

The literature on various aspects of multiprocessors is now quite diffuse, covering the areas of the design features or possible design features of multiprocessors (L2) (B12) (B1) (P11) (A12) (C21) (L1), the interconnection of small computers to form systems (J8), and the possibility of exploiting the inherent parallelism in programs via multiprocessing (B2) (R2) (R3) (R13) (R12) (G9) (F16) (H12) (D8) (D20).

The large centralized third generation computers placed a great burden on the operating system software, as mentioned earlier, but this burden was ignored by early computer performance evaluation studies, which dealt only with hardware performance. It is not surprising, therefore, that many performance difficulties have arisen on such systems. Already software development costs have equalled hardware development costs on third generation machines and software work is still proceeding (03).

These difficulties ensued because of the transition from the simple batch processing systems to the multi-programming, resource-sharing systems and because designers assumed that the same principles would still hold. Today, the transition from multiprogramming to multiprocessing is taking place and it is essential that designers do not repeat this mistake. Thus, the development of computer performance techniques for measuring multiprocessors can be seen as a priority and may avoid, on future generations of equipment and systems, some of the problems plagueing today's.

1.3. Where should the Operating System reside?

The first major decision that must be taken concerning systems which have more than one processor is where the Operating system should be placed and how it should be shared.

There are four basic organisations that have been used in design:

a. master/slave
b. separate executive for each processor
c. distributed operating system
d. symmetric or anonymous treatment of all processors.

Master/Slave

The master/slave organisation is the easiest to implement and may often be produced by making relatively simple extensions to a uniprocessor operating system that includes full multiprogramming capabilities. Although it is simple, this type of system is usually quite inefficient in its control and utilization of the total system resources (E8). Examples of the master/slave organisation can be seen in the early TRW 400 system and currently in the SEL 86, RCA 215 and IBM S/360 TSS model 6T systems. They include the following characteristics:

1. The executive is always executed by the same processor
2. Table conflict and lockout problems are simple
3. The software and hardware is relatively simple.

Apart from the inefficient use of resources, the master/slave system also has other major disadvantages which explain why current research tends to ignore it. These are the inflexility of the system when compared to the other three types of system, the fact that the slave processor is often idle and this idle time can build up appreciably whilst it is waiting for some service controlled by the master processor and, more important, the fact that the whole system is subject to catastrophic failure should the master processor fail. In fact, this type of organisation is really only effective in assymmetrical systems where the slave processor(s) has less capability than the master processor, as in the NBS Pilot system or the CDC 6600 system (where the i/o processors can be seen as slave processors).

Separate executive for each processor

In this type of system each processor acts as a single entity, each having their own set of tables, sharing only the workload.

They have the following characteristics:

1. Each processor services its own needs.
2. This type of system requires some reentrancy or replication of supervisory code (so that each processor can have its own copy).
3. Each processor has its own tables, though some tables must be common to the whole system.
4. Not very susceptible to total system failure.
5. Each processor has its own set of i/o equipment, files etc.

One of the major problems concerned with this type of system is that the problem of table lockout becomes very complex. The shared tables can be accessed by any processor and yet each processor is separately executing its own executive. This can easily lead to two processors trying to access the same set of tables and yet being unaware that they are in competition.

Although this type of system is not very susceptible to total failure, the restart of an individual processor that has failed is a difficult operation, since it will have lost the contents of its tables and the work it was executing.

A reconfiguration of i/o facilities requires manual intervention and possible manual switching because each processor has its own dedicated i/o resources.

There is also a serious loss in cost effectiveness brought about by having memory space taken up by multiple copies of the operating system, and, if multiprocessing systems are to take advantage of the cheap mini-computers and microprocessors, the loss incurred becomes more severe due to the smaller size of each processor and its associated memory.

Distributed system

A distributed computer system can be organised in two ways. The operating system can be either statically or dynamically distributed.

In the statically distributed strategy the operating system is split into functions which are assigned to special purpose processing elements

When a particular function is not in use the processor associated with it becomes idle. Today, some large computing systems embody such an approach, for example, the ICL 2900 series.

In the dynamically distributed strategy, although the operating system is divided into natural sub-computations in a similar manner, these are distributed and scheduled amongst the processing elements, i.e. each function floats from one processor to another.

Research related to the static type of distributed system is that of Fuller (F23). His aim is to build large systems by developing architectural techniques based on computer modules as building blocks. His basic philosophy is to put microprocessors next to memory, forming a processor/memory pair and to consider each of these as a module. A system can then be built up with these modules suitably interconnected, and should possess sufficient modules for each to be assigned a particular function within the system.

Research into the dynamically distributed type of system is being carried out by Whitby-Strevens (W8) at Warwick University. Such a system will incorporate communication paths between processors so that they can pass the modules back and forth. The best practical realisation of this seems to be the ring structure developed by Farber (F2) (F3) (F4) (F5) (F6) (F7) (F8) (F9) (F10) (F11) in which no processor functions as a central authority (otherwise the system will be liable to catastrophic failure as in the master/slave organisation).

Jensens Mini-multicomputer is an example of the use of the techniques of both dedicated processors and dynamic distribution (J3).

The dynamic system has the advantages of only having a small kernel of the operating system in each processor and of degrading gracefully, since, if one processor fails, the function can be assigned to another processor. In the static system, however, the whole system would fail due to that particular function as the failed processor being no longer available.

The dynamic system also utilizes the available resources better, since the processing elements are not prone to idle when a function is not required, as they are in the static system.

The dynamic system, however, has its own disadvantages. If the operating system is to be distributed, then it must be divided into modules of approximately the same size and each must be allotted to separated processors (or distributed evenly amongst the available processors). This leads to the first problem of how to divide the operating system. Some work has been done in this area by Spier (S24) and this aspect is also reflected in the design methodology of Parnas (P2). However, if it were possible to divide the operating system effectively, there are still difficulties which might lead the designer to question the worth of such an operation:

1. Each module must be self-sufficient in that it should not need to call upon a different module to perform some sub-task and then resume the operation.
   This condition must hold in order to avoid too much communication and swapping.
2. Modules must be reentrant
3. A safe mechanism of table access and control must be devised, since two separate processors could be executing the same module independently and be unaware of the competition for the same data access.

The distributed system is also inherently inefficient in that a processor would have to check several others before it found the module it was seeking and valuable processing time would be lost whilst this operation was in progress. Processing time would also be lost whilst a copy of the module was being transferred from one processor to another. This type of system is also inefficient because at least one copy of a particular module would have to be saved and not overwritten, so that it would be directly available to the system. Therefore, a check would have to be made to ensure that a processor can overwrite a module it no longer requires with one it now needs. Also, the situation may arise, in which a processor holds several 'last copies' of modules and this would severely restrict its memory requirements and may even paralyse that processor because it has not enough space left in its memory for a module which it requires.

To summarize, then, a dynamically distributed system would seem to spend too much time communicating rather than processing. Related to this is Eckhouse's comment (W9) that while the cost of memories and central processing units are being reduced rapidly, communication costs are only decreasing slowly. On the basis of this, he advocates that economic solutions to architecture organisation should rely on communications less and less.

Symmetric processors

In this type of system the processors all share one memory and are treated as anonymous. There is no master processor but all the processors are controlled by a single operating system, accessible by all. An example of this type of system is the C.mmp. system (C14) at Carnegie-Mellon University. The characteristics of the symmetric type of system include:

1. The 'master' floats from one processor to another, although several processors may be executing supervisory routines at the same time.
2. Reentrant supervisory code is needed.
3. This type of system attains a better load balancing over all types of resources.
4. Conflicts in service requests are resolved by in-built priority schemes
5. Table access conflicts can cause delay.

The advantages of a system, using symmetric processors with a shared memory, are that it provides graceful degradation, better availability when the system is running at reduced capacity, true redundancy and the most efficient use of the resources available.

Research into operating systems and their methodology seems to reflect a preference for this type of organisation. Hoare's work on monitor-structuring assumes a memory sharing system (H13). Brinch Hansen's work on Concurrent Pascal assumes memory sharing as a fundamental concept (B38) (B41). Dijkstra's Hierarchical design methods can be applied to multiprocessing systems with shared memory and his P and V operations are particularly valuable as aids to the table lockout problems which arise in this type of system (D11) (D12) (D13). The Hydra operating system for the C.mmp. works with a potentially

vast shared memory (W9) (W25). Betourne (B20) and Davis et all (D3) also assume shared memory using pages and modules respectively, for parallel processing or multiprocessing capabilities.

The main disadvantage of symmetric multiprocessing systems is that the shared memory can cause a bottleneck. However, there are methods to overcome or alleviate this problem by a suitable choice of memory management scheme (see section 2.3.2.1.). As this is the case, it would seem that the symmetric system will prove to be the most rewarding of the organisations discussed. Its advantages outweigh its disadvantages. It does not have the complexity of communications or the need for dividing the operating system which the dynamically distributed system has. It does not waste resources which the separate executive or the statically distributed system does. Nor is it prone to total system failure as in the master/slave organisation.

1.4. Aims and Methods

The aim of this study is to develop a methodology for computer performance evaluation which will be of practical use in the design, comparison and extension of systems (see section 1.1.1.). This methodology should provide an overall view of the computing power of a system, together with component interrelationship measures, in order to assess the overall results, with a view to comparing these to the objectives of designers or system managers (see section 1.1.2.).

In order to be of general use, this methodology must be both cost effective in its use and suitable for evaluating a large class of systems. The class of systems chosen for this study is that of multiprocessors with shared memory (symmetrical systems), due to the considerations explained earlier. (See section 1.2. and 1.3.). The aim is to measure various hardware and software configurations of this type of system with a view to providing basic information for use in making future design decisions.

The method used in developing this methodology reflects the current trends in design, which is that of decomposing the operating system into modules or some form of hierarchy (W7) (P2) (P1) (D12) D3), and is termed the building block approach. This approach has been taken both to co-ordinate with design and also to allow the flexibility which is necessary if the methodology is to be of general use.

The methodology consists of constructing a skeleton structure of the basis of computer systems of the type chosen. A library of routines is then developed, each routine reflecting a particular function within the operating system. The user can then select the routines which reflect the system he wishes to evaluate and fit them onto the skeleton structure. This operation should be cost effective since most of the work in building the model has already been performed for the user.

The tool used to develop the skeleton structure and the library of routines in this building block fashion was simulation which can be procedurised and which seems to be the only tool powerful enough to carry out such a task in any detail.

1.5. Brief Description of the Subjects covered in Succeeding Chapters

Chapter 2: This chapter is concerned with defining a multiprocessor system for the purposes of this study and providing the background to multiprocessor systems. It also describes the software and hardware features which have been proposed for multiprocessor systems and raises questions concerning the effects of those features.

Chapter 3: This chapter is concerned with the suitability of various methods of evaluating computer systems and suggests that simulation is the best approach, providing a suitable methodology can be developed to overcome the difficulties outlined.

Chapter 4: This chapter proposes a methodology for the design of a suitable simulation package.

Chapter 5: This chapter proposes a method of implementing a simulation package based upon the above methodology and describes (via its associated appendices) the actual steps taken in the development of S.C.O.P.E.

Chapter 6: This chapter (and its associated appendices) comprises the User's Guide for S.C.O.P.E. and describes how to model various single and multiprocessor systems using S.C.O.P.E. and how to run those models.

Chapter 7: This chapter describes an example of using S.C.O.P.E. to model a large complex system; namely, the system installed at the University of Aston which is an ICL 1904S machine running under the George 3 operating system.

Chapter 8: This chapter shows the results of several experiments with S.C.O.P.E. which attempt to answer some of the questions raised in Chapter 2.

Chapter 9: This chapter contains a summary of the research aims achieved, the limitations of the study and suggestions for further research.

Chapter 2 PROPOSALS FOR MULTIPROCESSING SYSTEMS

2.1. Development of multiprocessor systems

2.1.1. Definition of a multiprocessor system

The American National Standard Vocabulary for Information Processing (A10) defines a multiprocessor as

" a computer employing two or more processing units under integrated control."

This definition, however, is rather vague, but does contain a valid point. The requirement that a multiprocessor has "integrated control" is extremely important, for it must have a single integrated operating system. However, the concepts of sharing and interaction, which form the basis of the techniques of multiprocessing are not mentioned in this definition. A multiprocessor must have the capability for the direct sharing of main memory by all processors and the sharing of input/output channels by all memory and processor combinations.
Figure 1 gives a conceptual view of a multiprocessor system.

In multiple-computer systems interaction is present but at the level of a physical unit, such as a complete file or data set. In multiprocessors the level of interaction must be more flexible, from complete jobs to individual job steps. It is the combination of these expanded concepts of sharing and interaction at all levels which characterizes the hardware and software required for a multiprocessor.

A more complete definition based on these considerations would be:

1. A multiprocessor contains two or more processors (it may be further qualified that these be of the same capability).
2. Main memory must be shared and accessible by all processors.
3. Input/output access must be sharable.
4. The entire system must be controlled by one operating system providing interaction between processors and their programs at the job, task, step, data set and data element levels.

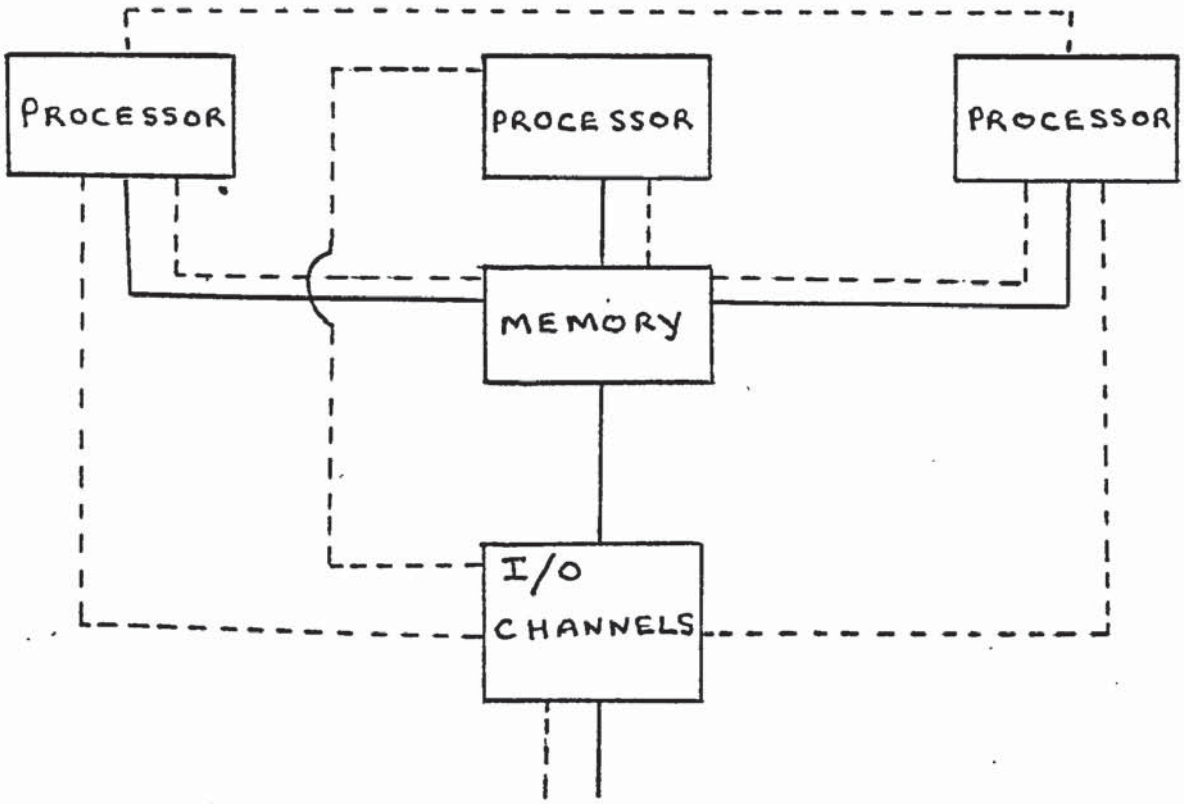
There are certain similarities between multiple-computer and multiprocessor systems, since both were motivated by the same goal of simultaneous operations within a system. However, there is a distinction between them and this lies in the degree of sharing and interaction which takes place in the system (E6). A multiple-computer system consists of several separate and discrete computers, even though they may be connected directly, whereas a multiprocessor is a single computer with multiple processing units. In general, the main aim of a multiple-computer system is to relieve the input/output load from the main processor, and it is because the processors see and treat the other(s) as i/o channels that they cannot be classed as multiprocessors.

Examples of such multiple-computer systems are given below: (see section 2.1.2. for more examples),

1. Satellite computers (peripheral, stand-alone) e.g. IBM 1401.
2. Loosely (indirectly) coupled systems.
3. Directly coupled systems, e.g. Honeywell 8200
   CDC 6600 (with 10 peripteral processing units)
4. Attached support processor, e.g. IBM ASP.

Figure 1:

A CONCEPTUAL VIEW OF A MULTIPROCESSOR.



——— DATA AND INSTRUCTIONS

- - - - - - - CONTROL SIGNALS

TABLE 1 (adapted from Enslow (E8)

DEVELOPMENT OF MULTIPROCESSOR AND PARALLEL SYSTEMS

Date	Manufacturer and Model No.	Comments	Further Reading
1958	National Bureau of Standards - PILOT	Three independently operating computers that could work in co-operation. One processor acts as the supervisor.	(L8)
1958	IBM AN/FSQ-31 and 32	Solid-state SAGE computer. Duplexed system, not multiprocessor	(E11)
1960	Burroughs D-825	First modular system with identical processors. Total memory shared by all processors. Up to 4 processors, 16 memory modules, 10 i/o controllers. One of earliest examples of modern operating system: ASOP - Automatic Operating and Scheduling program	(A12)
1960	Ramo Wooldridge TRW-400	Early TRW-400 used master/ slave configuration. Used for U.S.A.F. control and command	(P11)
1960	Univae Larc	One i/o processor and one computational processor operating in parallel. Not a multiprocessor	(E1)
1961	IBM Stretch (7030)	Contained look-ahead facilities	(B49)
1963	Burroughs B-5000	One or two processors, and up to eight memory modules. Utilized virtual memory concepts. Became the B-5500 in 1964.	(B51)
1963	IBM 704X/709X (7040 or 44 and 7090 or 94)	"Direct coupled system"	(I2) (I3) (I4)
1963	Bendix G-21 (later CDC)	A multiprocessor version of the G-20 developed for the Carnegie Institute of Technology. A crossbar system.	
1963	IBM, MSC	A custom multiprocessor system to support Manned Space Centre. Originally 7090's sharing large core; later 360/75's	
	cont		

Date	Manufacturer and Model No.	Comments	Further Reading
1964	CDC 6600	Contained multiple arithmetic and logic units each of which could execute only a small fraction of the total instruction repertoire,10 peripheral processing units to deal mainly with i/o functions. Overall system an example of an assymmetric multiprocessor	(T5)
1964	Burroughs B-5500	Upgrade of the B-5000. Used master/slave configuration	
1965	GE645	Delivered to Project Mac at M.I.T. Hardware not a standard product; however MULTICS operating system is being released	(C23)
1965	Univac 1108		(S26)
1965	Solomon I	Design only. First large scale array processor	(B8) (G21)
1966	IBM S/360 Model 67	Special dual processor time-sharing system. Master/slave configuration	(G3)
1966	Solomon II	Design only	(S19)
1967	CDC 6700	Dual CDC 6600's	
1968	CDC 7600	Very similar to 6600, but higher speed and included hierarchy of main memory as a feature	
1969	IBM S/360 Model 65MP	Dual processor version of standard model 65	(W22) (B23)
1970	Burroughs B-5700	Similar to B-5500 with capability for increased memory. Capability for four B-5700 systems to share disk storage	
1970	C11 Iris 80	True multiprocessor;processors considered as anonymous resources	
1971	Honeywell 6050, 6060, 6080		
1971	Burroughs B-6700		
1971	DEC System 10/1055, 10/1077		
	cont		

Date	Manufacturer and Model No.	Comments	Further Reading
1971	Univac 1110		
1971	SDC PEPE (Parallel Element Processing Ensemble)	Prototype for processing of radar data for ballistic missile defence system	(W16)
1971	Fairchild Symbol 2R	Seven processors dedicated to separate functions	
1972	Univac 1106		
1972	Honeywell 2088		
1972	Illiac IV	Array processor, 64 processing elements. Driven by a conventional multiprocessor used as a front-end control processor	(B12)
1972	Burroughs B-7700		
1972	CDC Cyber 72,73,74,76		
1972	Goodyear STARAN S	Parallel associative system	
1972	Texas Instruments ASC (Advanced Scientific Computer)	Embodies both multiprocessing and pipelining	(W5)
1973	Bolt Beranek and Newman, PLURIBUS	Multi-minicomputer stressing reliability	(05)
1973	CDC STAR-100	Pipeline system, Arithmetic unit consists of two floating point pipelines and a string unit	(C20)
1974	IBM S/370, Models 158 MP and 168 MP	Shared real and virtual storage	(A18)
1974	Carnegie-Mellon Univ. C.M.M.P.	Multiprocessor with 16 PDP-11's sharing memory through a crossbar. Symmetric processors	(W24) (C14)
1975	Univac 1100/20, 1100/40		
1975	Univac 1100/10		
1975	Tandem T16	Fault-tolerant multiprocessor	
1976	DEC System 10/1088	Dual processor	
1976	Cray-1	Pipeline system	
1977	Goodyear STARAN E	Parallel associative system	(M19)

2.1.2. History of multiprocessor systems

Although multiprocessing is frequently thought to be a new development and the next step in computer technology, its development has actually covered a long period within the history of digital computing. The first multiprocessor operating system, as defined previously (see section 2.1.1.), was the D-825 which was introduced in 1962, which is one of the first general operating systems. However, the roots of multiprocessor systems lie in the development of multiple-computer systems which were introduced even earlier. A brief history of this development in chronological order is given in table 1. A summary of the development of multiple-computer and multiprocessor systems in the various series of IBM computers is given by Freeman (F20).

Despite the fact that multiprocessor systems have occupied such a long span of computing history, for many years they have been rare and have been found primarily in special systems requiring high availability, such as military command and control applications. This trend has, however, recently changed, and multiprocessors can be found in many installations, working on different applications. It is this change in trend which has led to the academic interest now shown in multiprocessor systems and to the new developments in theory. This is shown by the fact that, in a recent international survey of the architectural characteristics of medium and large scale systems, the Auerbach study (A20) identified 45 out of the total 158 models included as having the hardware capability to be configured as a multiprocessor. However, it is important to note that the Auerbach study was unable to verify if there were multiprocessor operating systems available for all those 45 machines.

A suggestion for further reading concerning the development of multiprocessors is the Infotech publication, which describes the state of the art of multiprocessor systems (I9), and gives the logical rather

than the chronological development of multiprocessors, the main stages of which are:

a. The introduction of the i/o channel

The first step towards an architecture in which different parts of a program were processed by separate units was the functional division of the program into arithmetic and logical operations and i/o operations, for which the i/o channel was introduced. The channel is actually a small special-purpose computer, designed specifically to handle i/o transfers. The main processor could then pass a control command to a channel, forget the i/o operation and continue with some processing, whilst the channel dealt with the i/o operation in parallel with it.

b. Peripheral stand-alone systems.

A development from this functional division of the work was to use independent processors to prepare i/o data for processing by transferring it from slow to fast media. In this situation, there are two completely separate and often quite dissimilar operating systems controlling the computers.

c. Indirectly or loosely coupled systems.

In this type of system the processors interact via common data in shared backing store on disks or drums. In such systems it is still usual for the processors to be engaged on functionally distinct areas of the workload. Each of the systems sees the other as an i/o device and there are two separate operating systems controlling them.

d. Directly or tightly coupled systems.

The directly coupled system is an improvement on the shared peripheral store. The processors are coupled via shared memory or a channel-to-channel adaptor. There are two separate operating systems and each system sees and treats the other as an i/o device. It is also necessary to duplicate the peripheral devices since each system has its own set.

e. Non-homogeneous systems: built up from a mixture of processors with different characteristics and functions, usually under the control of a standard sequential processor.

f. Array processors: in which many data streams are processed in parallel according to a single instruction stream.

g. Pipeline processors: in which the decoding of successive instructions of a program is overlapped to give the effect of executing more than one part of a program at a time.

h. True multiprocessors: as defined in section 2.1.1.

2.2. Objectives of multiprocessors

The motivating factor behind the development of multiprocessor systems was not that of relieving the input/output load from the main processor (as in multiple-computer systems) but was, initially, the high system availability that could be obtained by having a reconfigurable system should one processor fail. It was later that designers began to capitalize on the fact that multiprocessors could also improve system performance by providing a more economic handling of exceptional jobs or peak loads by attaining a higher overall effectiveness of resource utilization.

Another motivating factor was to provide true simultaneous execution of jobs. The operating systems of multiprogramming systems and those of multiprocessing systems are similar, but the key difference is that the former only seems to perform several execution tasks at once (concurrent execution) whereas the latter supports actual simultaneous execution of two or more processes. This capability assists in the speed up of throughput and in improved utilization of resources.

Thus, the objectives were:

1. To provide high availability of utilizing:
   a. the reliability improvement attained through multiple units
   b. the ability to reconfigure the system (i.e. making it fail-soft)

and 2. To improve system performance by:
   a. providing a more economic handling of exceptional jobs or peak loads
   b. attaining higher overall effective resource utilization
   c. being able to exploit parallel execution of independent tasks
   d. improved system balance.

The above objectives together can form four general categories which multiprocessor systems aim to improve,

1. Reliability
2. Flexibility
3. Availability
4. Throughput

and the ultimate improvement of these areas should improve overall system performance.

In addition to the previously listed factors which have motivated the development of multiprocessor systems, another important aspect of the current situation is that they can be used to provide increased capacity while retaining the same type of processing units. This kind of use allows manufacturers to capitalize on investments already made in the development of a system, by adding another processor of the same type and still retain their customer population as their computational requirements increase. This saves the large expenditure and complexity incurred in changing to a larger machine. Figures 2 and 3 illustrate this cost factor.

Figure 2 is adapted from material prepared by the Quantum Science Corp.. The problem of determining exactly equivalent configurations is difficult; however, the figures presented in the original material (C.1972 unpublished), which uses IBM System 370 models 155 and 165 as the two specimen computers, is sufficient to illustrate the observation being made.

Figure 3 is intended to illustrate a general situation but it does appear that this condition exists with some of the larger Honeywell systems (E8).

The starting points for cost and performance are taken as unity for simplification.

From Figure 2 it can be seen that a fully expanded model 1 can provide almost double the performance for about 1.7 times the cost, and that a large gap exists between model 1 and model 2, which provides 3.4 times the performance for 2.2 times the cost. If model 1 processors are used in a multiprocessing configuration, the cost/performance curves for even a single processor are always below the standard model 1 uniprocessor because of the cost of the extra hardware needed to support multiprocessing. However, if the single processor system is saturated another processor can be added, together with other equipment that might be required (such

as an extra i/o channel or more memory) and the cost/performance curve continues smoothly. This process can be continued without any changeover.

The advantages of this are:

1. it is possible to provide smooth growth for increased capability
2. there is no drastic changeover involved
3. there is often a performance range in which the multiprocessor system is more cost effective
4. the multiprocessor is more reliable.

However, the disadvantage is apparent:

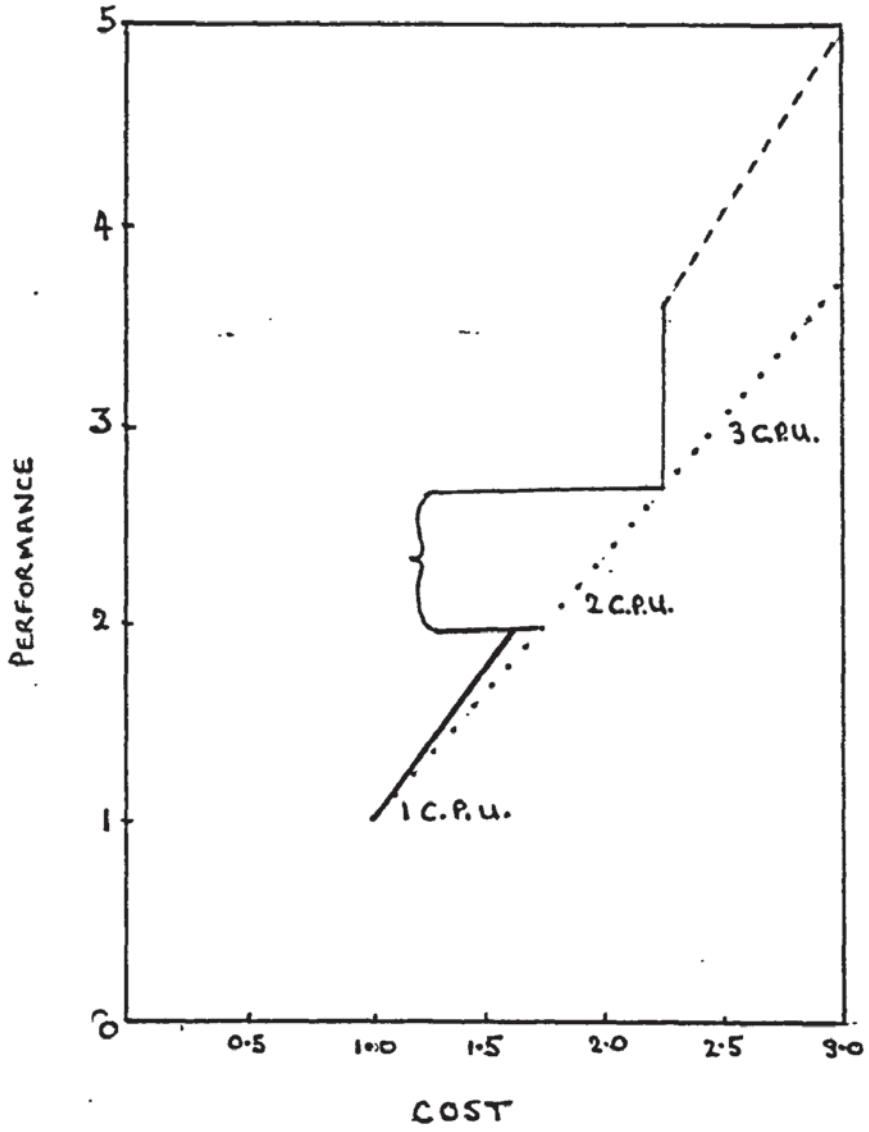
2. the multiprocessor system is a less cost effective proposition overall due to the increase in cost for the hardware to support communication, sharing etc. (This is the price paid for increased reliability/availability),

Figure 3 illustrates a different situation which has occurred with the recently introduced Honeywell systems. A single processor designed to include multiprocessor capability is still more expensive than the uniprocessor; however, it has the capacity to expand pas the limits of the uniprocessor. Also, as it expands its cost effectiveness is better than the next larger uniprocessor model.

This situation greatly favours the multiprocessor configuration and will probably continue to be the case since the cost of small C.P.U.'s is being reduced at a rapid rate and because memory (which needs to be added when a processor is added) is following suit.

Figure 2:

COST/PERFORMANCE COMPARISON



——— MODEL 1

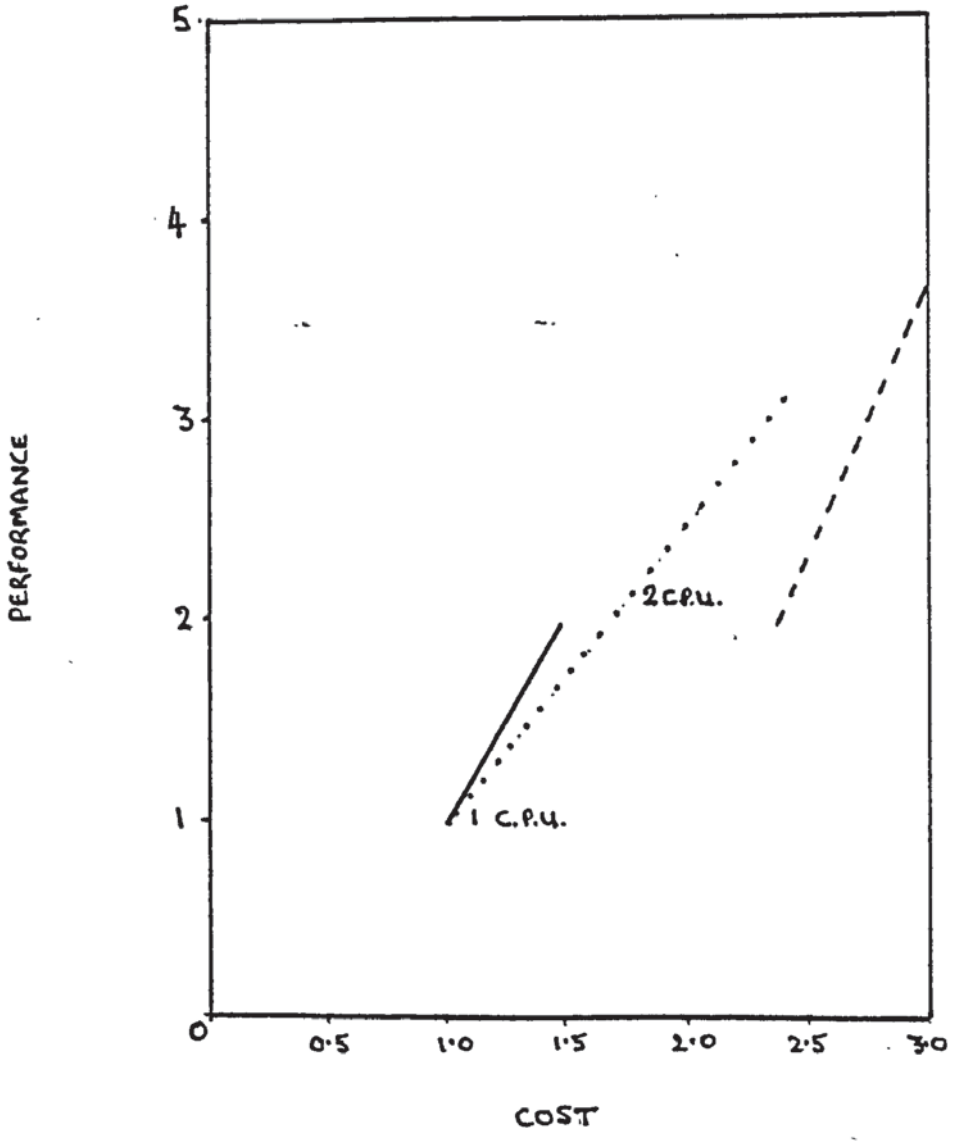
— — — — MODEL 2

..... MULTIPROCESSOR

} COST / PERFORMANCE ADVANTAGE OF A MULTIPROCESSOR

Figure 3:

COST/PERFORMANCE COMPARISON



MODEL 1

MODEL 2

MULTIPROCESSOR

2.3. Design considerations for software

2.3.1. Introduction

To the user the basic capabilities of an operating system for a third generation uniprocessor and for a multiprocessor should be the same. The differences should be internal. It has often been the development of the operating system which has set the pace for the development and performance of the multiprocessor organisation (E7). This leads to the first primary difference between the uniprocessor and the multiprocessor which is the need for an extensive operating system by the latter. An experienced programmer can use a uniprocessor without any system software (although the effective utilization would be low) whereas an operating system is usually necessary to even start a multiprocessor.

In fact, it was the development of the early "multiprocessors" such as the NBS Pilot (L8), Burroughs D-825 (A12) and the Ramo Wooldridge TRW 400 (P11) that first brought into focus many of the basic problems encountered in operating system organisation and operation.

Due to the great similarity between multiprogramming and multiprocessing operating systems, it is assumed that the reader is familiar with general operating system theory, (C17) (C18) (B42) (M5) (G11) (T9) (S13) (C25) (K6) as this section only serves to point out the differences caused by multiprocessing.

It is also important to note that multiprogramming and multiprocessing are not two distinct disciplines but that multiprogramming is almost an essential capability of a multiprocessor operating system and might be considered as a subset thereof.

Although the hardware of a multiprocessor is more complex it does not necessarily follow that the software is also. For, in fact, the redundancy of some functional units can act as an aid to solving some of the problems which a uniprocessor encounters, such as:

1. Adaptability to changing demands both in quantity and in the nature of the workload.

2. Availability of at least minimal operation for the completion of critical functions, i.e. the system degrades gracefully rather than collapses.
3. Expansion to a larger system with no effect to presently operational programs.
4. Response time.

However, the additional units do complicate the software functions in ways not encountered on a uniprocessor such as:

1. Utilization of all resources to the maximum extent possible.
2. Error recovery to save work in progress.
3. Synchronization.
4. Reentrancy.

To summarize, the multiprocessor operating system has to have all the capabilities of a uniprocessor operating system and also has to have the ability to perform several other important tasks that result from the nature of the hardware. These functional tasks are described below.

2.3.2. The functional tasks of a multiprocessor operating system

A list of the main functional requirements of a multiprocessor operating system shows that the capabilities are similar to those of a uniprocessor:-

1. Resource allocation and management.
2. Table and data set protection.
3. Interrupt management.
4. Abnormal termination.
5. I/o load balancing.
6. Reconfiguration.
7. Exploitation of parallelism.

However, many of the problems encountered in providing the common capabilities may be more difficult to solve because of the additional processors present in the system. Further, the effective use of these additional resources makes it even more important that efficient solutions be found; otherwise, poor performance will negate any other advantages that a multiprocessor might have. In fact, the efficiency of the operating system is far more important in a multiprocessor system than in a uniprocessor system.

2.3.2.1. Resource allocation and management

The major resource allocation and management schemes can be categorized in the same manner as those of a uniprocessor:

1.	High level scheduling:	The selection of jobs from the input queue to form the active workload mix.
2.	Low level scheduling/dispatching:	The assignment of a processor unit to the execution of a task.
3.	Memory management:	The allocation, reallocation and control of the main memory required for both user programs and system software.

High-level scheduling

The selection of which jobs to run on a computer can be an important factor in the performance of that computer. Different workloads have different effects on the system and its resources. As a simple example, a workload of mainly i/o bound jobs will probably have the effect of overloading the i/o resources where queues will form, thus leaving the processors idle.

The selection of jobs to run can be made in different ways, depending upon the objectives of the system manager, which might be to give priority to certain users, to run all the small (or large) jobs first, to create a "good mix" of jobs, etc. or combinations of these.

The simplest method is to use a first come first served queue. In this situation, there is no control over the workload.

Another method is to assign priorities to users, and to form the queue on the basis of these priorities.

A more complex method, which gives more control than the previous two, is to calculate an index for each job depending upon the objectives of the centre and the characteristics of the jobs. The queue is formed on the basis of the values of the index of each job.

The above methods are all common to uniprocessor systems. The differences would lie in the differing objectives which would arise due to the differences in hardware resources and system performance.

Low-level scheduling/dispatching

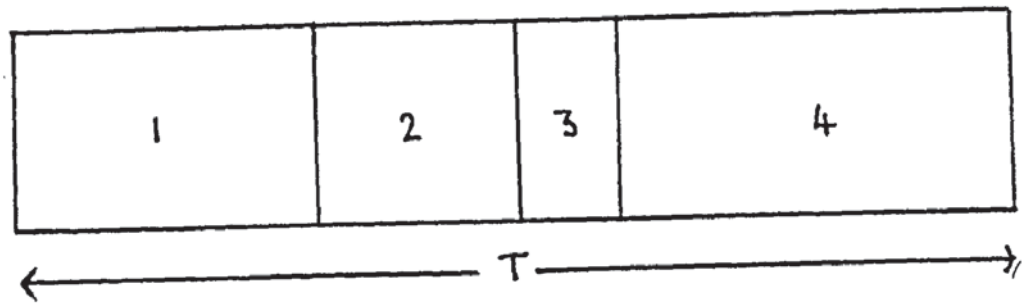
It is possible that the scheduling and dispatching algorithms will not exploit the full processing resources of a multiprocessing system. If the algorithms are badly organised, it may result in the system's failure to utilize several idle processing units. As an example of this kind of situation consider the case in which the scheduler assigns a queue of jobs to a particular processor and from that time considers that processor only to be responsible for executing those jobs. Under such circumstances, the situation could easily arise in which one processor has a queue of jobs ready to run, and several others are idle due to their having completed the jobs assigned to them.

In order to overcome problems of this kind, multiprocessors must be considered as multi-server queue systems, where the queue of tasks ready to run is common to all processors and any processor may execute any subpart of any job. (This will require that the scheduler and dispatcher are classed as critical regions (see next section 2.3.2.2.).

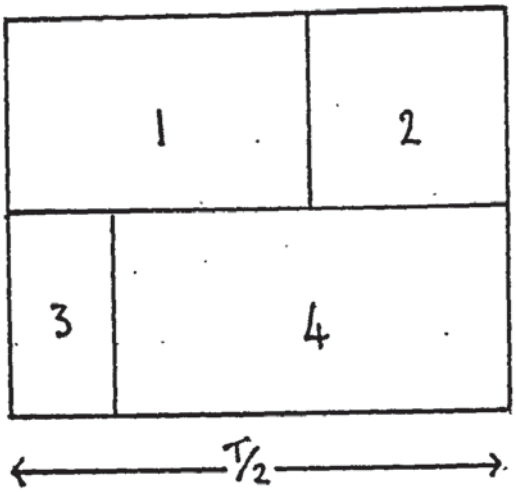
Another aspect which will complicate the scheduler and dispatcher is the exploitation of the parallelism within jobs. If the system is to exploit this, the scheduler must specify the relationships between the various processes which constitute a complete job and the dispatcher must handle the order of execution of independent tasks as well as processes which are dependent upon one another (D13) (D12). The identification of those taks which can be executed in parallel has been the subject of a large amount of research (see section 2.3.2.7.).

Another problem, which is specific to multiprocessor systems is that commonly referred to as "dispatching anamolies". This is well illustrated by an example from Jordan (J6).

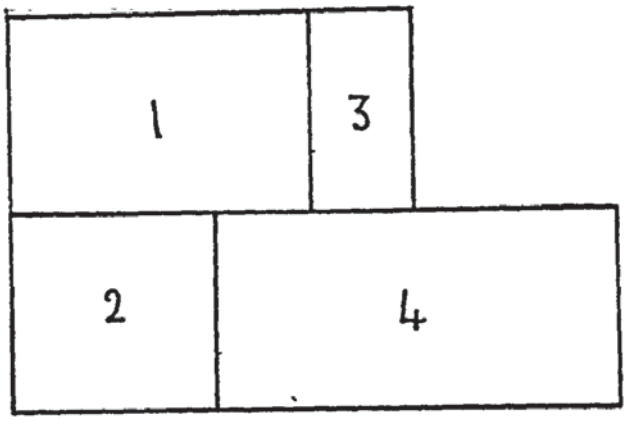
a)


1
2
3
4
PROCESSOR 1
T

b)


1
2
3
4
PROCESSOR 1
PROCESSOR 2
T/2

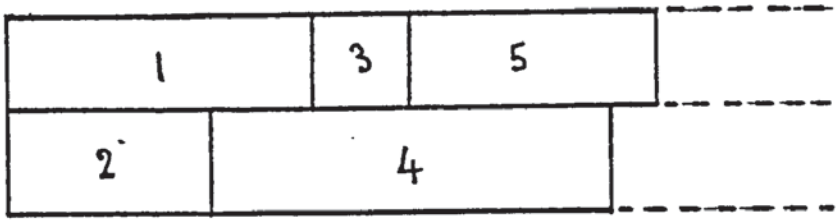
c)


1
3
2
4
PROCESSOR 1
PROCESSOR 2

The figure a.

"represents a multiprocessor with one processing unit. If a second were added to an 'ideal' multiprocessor, the task list would be processed in half the time. However, for an actual multiprocessor, b. and c. clearly show that the time required is a function of the order in which the tasks are processed". (J6).

However, it is questionable as to whether the dispatching algorithms need to be adapted to take this into consideration, for in an actual multiprocessor, there would be jobs following job 4 which could be dealt with by processor 1.



The above figure shows that in such a case the "dispatching anomalies" become neglible. Also, even if an optimal strategy could be devised, the work of Manacher in 1967 and that of Graham (G18) would question the usefulness of this, since they found that optimal scheduling lists fail to remain optimal for 80% of the time, if the execution times of the jobs are slightly perturbed in a random manner as would normally be the case.

Low-level scheduling strategies for multiprocessors are very similar to those of multiprogramming uniprocessors if one disregards the problem of dispatching anomalies, since multiprogramming can be seen as a subset of multiprocessing. The jobs are time-sliced i.e. a process is given control of a C.P.U. for a certain amount of time (usually 100-300 millisecs). The processes are not run to suspension as in the early batch computers because a large job can consume too much processor time and because of the possibility of an infinite loop within a program. If the ready queue is viewed as a multi-server queue, the algorithms used to form the queue are those used in multiprogramming systems,

for example:-

1. Round Robin (FIFO)

When a process finishes with the C.P.U. it is put at the back of the queue. There is usually no priority. The process must then wait for all the other processes in front to be serviced by a processor, before it is its turn again.

2. Priority scheduling

The process with the highest priority is put to the top of the queue. Priorities can be assigned for the "life" of the process, and will thus remain constant, or can be varied

3. Inverse remainder of time slice

If a process used its entire quantum of time last time, it is placed at the end of the queue. If it used only part of it due to an i/o request it is placed into the middle of the queue. This algorithm was devised to keep i/o devices busy and is based on the assumption that a job which requests i/o once is likely to request i/o again.

Memory Management

As there is only the one common memory shared by the processors the memory management schemes are no different from those of a uniprocessor, for example, the techniques of partitioning, segmenting and paging are used. Certain complications, however, not present in a uniprocessor system, can arise. For instance, there is the problem of whether a processor can access the particular part of memory it requires, and a check may be needed to discover if another processor is already utilizing that part of memory. Also, unless the processors have common hardware level synchronisation, there is the problem of access conflicts.

The primary memory management schemes are:

1. Partitioned allocation
2. Relocatable partitioned allocation
3. Segmented allocation
4. Paged allocation

Partitioned Allocation

The partitioned allocation scheme fills the gaps in memory with jobs; each job being partitioned from another by hardware and the partitions are set up as required. However, there are often gaps in the memory which are too small to be useable. This problem is called fragmentation.

Relocatable Partitioned Allocation

The relocatable partitioned allocation scheme is similar to the above scheme but attempts to overcome the problem of fragmentation by keeping a check of the gaps left unused and, if necessary, by relocating the jobs in such a way as to join all the previously unusable gaps.

It is important to note that in the above schemes, as the size of jobs will vary, so the size of the modules formed by the partitions will vary. The remaining schemes assume the size of the modules to remain constant and are fixed accordingly.

Segmented Allocation

The segmented allocation scheme divides the memory into modules (or segments) of a set size and the hardware partitions are fixed accordingly. Into each of the segments is placed a job. In consequence, there may be a great deal of space wasted within each module, since the modules must be large enough to accommodate the largest job expected, and yet may only be accommodating a small job.

Paged Allocation

The paged allocation scheme is similar to the segmented allocation scheme, but the modules (or pages) are small enough to accept only parts of jobs. In this scheme, the jobs have to be divided into segments of the same size as the hardware pages, which is a constant size, and are copied into and out of the pages as required. This scheme is an attempt to overcome the problem of the memory space wasted by the segmented allocation scheme, but is far more ccmplex.

The hardware organisation of the multiple units can place constraints upon which of the above schemes can be used. For example, certain hardware organisations require that the memory be split into modules of a set size (see section 2.4.). In this case one of the latter two schemes will have to be used.

2.3.2.2. Table and data set protection

Although the protection of tables and data sets from illegal access is a problem encountered in uniprocessors, the problem becomes more complicated within a multiprocessor. More than one processor may be executing the same or related parts of the supervisor routine, and in this case, an illegal access to a table can occur on the next memory cycle after a legal one. It is illegal because it is essential that only one processor accesses that table at a time. The multiprocessor system, therefore requires some hardware and/or software capabilities to prevent these unorderly changes in a shared data base. This problem has been referred to as the "lockout" or "mutual exclusion" problem, and the portion of code that accesses the shared data base is called a "critical region". (D12).

One method of providing the necessary protection is by having instructions of the form TEST AND WAIT AND SET (L6) or the equivalent pair LOCK W/UNLOCK W (D9), where W is a bit switch. The effect of these instructions is that when one process enters the critical region W is locked. Any subsequent processes must loop and keep checking the switch until the former process leaves the critical region, when it will unlock W.

The main draw back of these approaches to the problem, is that processes are busy setting flags or waiting to enter, thereby slowing the system. Saltzer (S2) and Lampson (L1) attempt to overcome the looping problem by defining WAKE UP (P) and BLOCK (P) operations.

Dijkstra's solution to this problem has been widely accepted as it does not require processes to be cognizant of the others waiting to enter the critical region (B1). Dijkstra defines a new type of variable called a semaphore which can take only non-negative integer values, and two operations which can be performed on them, P and V (D12), which are defined as follows: (Algol 60 notation).

```
V (S) -   S:= S + 1;
P (S) -   L:if S = 0
            then go to L else S:=S - 1;
```

These statements are indivisible and may not be executed simultaneously. Wirth (W2) has since proposed that the P and V operations be carried out by hardware.

However, the indivisibility aspect of the P and V operations has been criticised, and Bredt (B32), using a model similar to that used in sequential circuits theory, gives procedures to design a hardware lock-out mechanism free of any assumption of indivisibility.

2.3.2.3. Interrupt management

When a job has terminated or requests i/o the processor executing it sets up an i/o channel to carry out the task. When the channel has completed that task, it must then interrupt a processor to inform the system of that fact. The problem then arises as to which processor should be interrupted. The wrong scheme for making this choice may adversely affect the system's performance.

The possible schemes which could be adopted are:

1. The same processor all the time
2. The processor which initiated the i/o operation
3. The processor whose turn is next, basing the choice on a round robin scheme
4. An idle processor. (If there is no idle processor, revert to options 1 - 3),

Another method, however, may prove more efficient, if it can be implemented as a system. Instead of the channel interrupting a processor, which involves changing a bit in the interrupt register, it could enter the memory and change a bit in the queue of jobs, altering the status of the job from blocked to ready. The scheduling strategy would have to take into account the fact that both blocked and ready jobs would be in the same queue.

2.3.2.4. Abnormal termination

Because of the nature of the multiprocessing system organisation, if adequate processor communication is not available, it is very difficult to terminate properly a multitask job, if the tasks are being executed on separate processors. Again, rather than being a unique requirement of a

multiprocessor system, it is more a matter of degree of complexity above the requirements of a uniprocessor.

2.3.2.5. Input/output load balancing

This capability may be difficult to achieve considering the configurations and i/o paths possible in a multiprocessor system. If certain i/o devices are untransferably attached to a particular processor, as in some multibus configurations, the i/o scheduling can be very difficult and must be handled via interprocessor communications. If, however, the i/o devices form an anonymous pool of resources, shareable by all, the control of the i/o load is similar to that of a uniprocessor, as only one i/o queue is required.

2.3.2.6. Reconfiguration

Although some uniprocessors do have limited capability to recognize changes in the equipment available for use, the ability to do this automatically and to adjust all the necessary tables and routines is essential for a good multiprocessor operating system.

2.3.2.7. Exploitation of parallelism

Parallel processing is said to occur when more than one processing unit executes parts of the same program simultaneously.

The literature on this subject, seems to discuss parallel processing at three different levels.

The first level is the processing of independent tasks governed by the same operator; for example, when ultiplying an array by another array, each multiplication can be processed independently. These kinds of operations can be carried out by array processors such as the ILLIAC IV (B12) or the SOLOMON I (B8) (G21) which have been designed specifically for this purpose.

The second level is that of processing independent tasks within an expression (R13) (K23) (R12) (R3) (B2). This involves instruction look-ahead, in which the following parts of the expression are checked to see if any can be executed in parallel. This type of parallel processing is carried out by pipeline systems, specially designed for such capabilities, such as the CDC STAR-100 (C20).

Neither of the above two cases, however, involve true multiprocessors. The third level of parallel processing, which deals with independent blocks of code, is the level which concerns true multiprocessors. At this level, the onus is either on the programmer to code his programs in such a way that the independent blocks of code are apparent to the system or the onus is on a system routine working alongside the compiler to translate a users program into independent parallel tasks.

It must be noted that with either method, it must be ensured that the tasks to be executed in parallel are large enough, such that any efficiency gains counteract the overhead associated with allocating an extra processor to the tasks and then later de-allocating it. Critchlow (C25) points out that the executive must control interrupts, save registers, check pointers and assign facilities for each segment started. These can easily require 10 or more memory cycles and hence he estimates that segments less than 100 memory cycles will prove inefficient.

There are also certain other requirements to be adhered to, for parallel processing to take place (E7).

1. The start and end points of the segments to be executed in parallel must be specified
2. Any special conditions which govern or restrict parallel execution must be specified
3. The conditions which govern the proper synchronization of the parallel tasks with the other parts of the program must be given
4. There must be capabilities within the supervisor to create, schedule, dispatch and recombine the parallel tasks spawned by a single program.

Conway (C21) and Baer(B1) discuss the programming constructs required if the programmer is to be responsible for specifying the parallel segments of his job. These include the FORK-JOIN constructs (C21), the DO TOGETHER-HOLD constructs (O3), the FORK-JOIN-TERMINATE constructs (A13) and the PARBEGIN-PAREND constructs (D12) (W19).

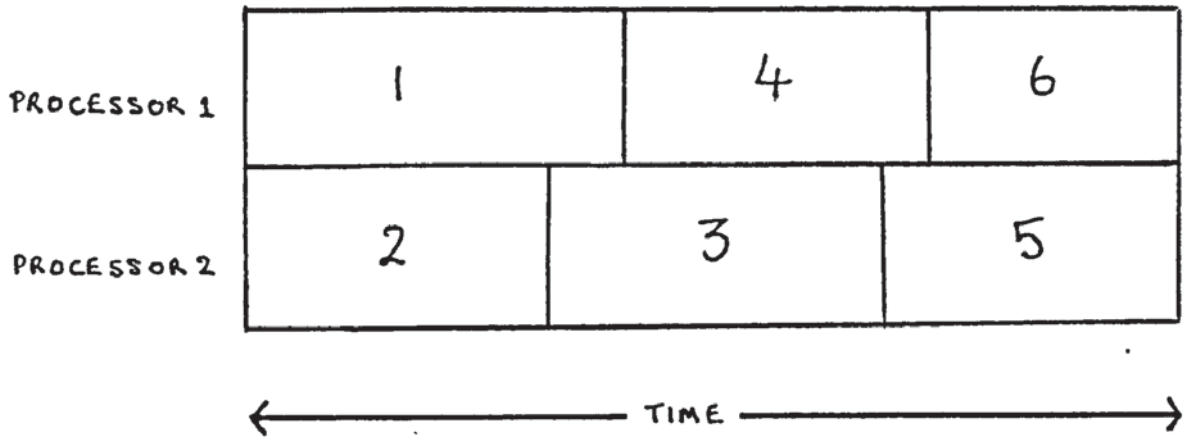
The research of Ramamoorthy and Gonzales (R2) (R3) (G9) concerns the alternative method, i.e. when the onus is placed upon a system routine to detect and exploit parallelism. This routine would work alongside the compiler and should require no assistance from the user.

Baer (B1) sees the methodology of Ramamoorthy and Gonzales being used at the second level of parallel processing, that is, as a form of instruction look-ahead. In this case more than one processor would have to be associated with the job being executed throughout its life, since allocation and de-allocation overheads would reduce efficiency a great deal at a level of such small tasks. However, during the strictly sequential parts of the job the extra processor(s) would be idle. This methodology might be better employed at the third level, if the extra capability, that the routine checked that the size of the segments were large enough, could be included.

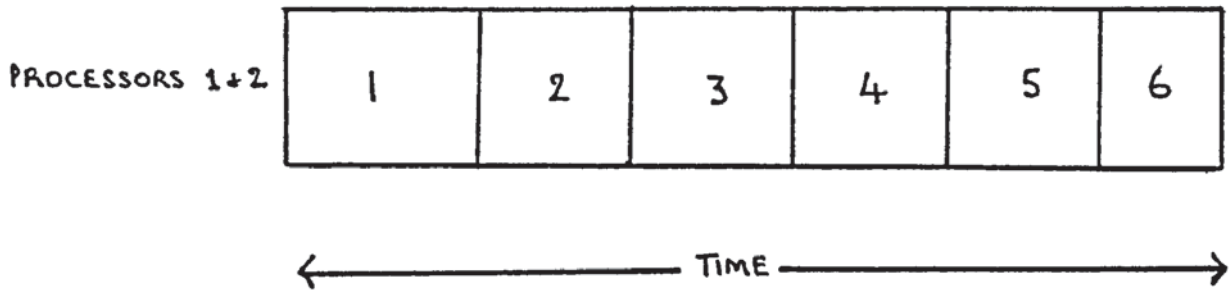
Thus far, it has been assumed that the exploitation of parallelism is a worthwhile, if not necessary, aim of multiprocessing systems. This assumption is made by many authors dealing with this subject. However, if the advantages and disadvantages are inspected, this assumption becomes questionable.

The obvious advantage is that each job is executed faster. In realtime systems this facility is important; but outside realtime applications this advantage becomes relatively unimportant since it does not necessarily follow that each job will be on the computer less time than if parallelism was not exploited by the multiprocessor. This can be illustrated by the following diagrams, which assume that a multiprocessor consisting of 2 processors will cut the job execution time by half, when exploiting the parallelism within the jobs.

Without parallel processing:-



With parallel processing:-



As can be seen from the diagrams, there will be no gain over several jobs. In fact, if the inefficiencies caused by the allocation of processors, the extra complexity of the scheduler and dispatcher, and the extra registers which need to be saved are considered, the assumption that the processing time will be cut by half will not hold.

There are also other factors which seem to point to parallel processing being unfeasible.

The first is the fact that time must be spent either by the programmer or by the system routine in identifying and specifying the parallel segments.

Secondly, there is the presence of critical regions (see section 2.3.2.2.) which implicitly impose serialism into certain parts of the operating system routines. These will have a degrading effect on the efficiency of a system exploiting parallelism.

Finally, there is the problem posed by abnormal termination. If a job is split into several parallel segments, all being executed on separate processors and an error occurs in one of those segments, the problem arises as to how to terminate the whole job efficiently.

If all these factors, taken together with the complexities of the operating system required for parallel processing, are considered, it would seem that there would be no gain, but a loss in efficiency when exploiting parallelism. The only applications which might benefit from an investment is parallel processing are realtime situations, where the speed of results is imperative.

2.4. Design considerations for hardware

Using the definition of a multiprocessor given in section 2.1.1. it is possible to enumerate the hardware requirements of a multiprocessor system. There must be more than one processor and all processors must have access both to shared memory and to the i/o devices.

From this it can be seen that the primary hardware effect of multiprocessing is the interconnection of all these units. These interconnections should provide the full requirement for total resource sharing, that is:

1. Any processor can control and transfer data to and from any location in memory
2. Any processor can pass control commands to any i/o channel controller
3. Any i/o channel can pass data to and from any location in memory
4. Any i/o channel can control and transfer data between the main memory and any appropriate i/o devices.

For true multiprocessors, where several processors share common memory, four basic types of interconnection system can be defined; (B1), (C25) (E7) (I9)

1. Time-Shared bus
2. Multiple bus
3. Crossbar matrix switch
4. Multiport

Time-shared bus (see Figure 4)

In this system organisation there are no continuous connections between the functional units. The control of transfers between memory modules and other units is accomplished by using time-sharing or multiplexing techniques.

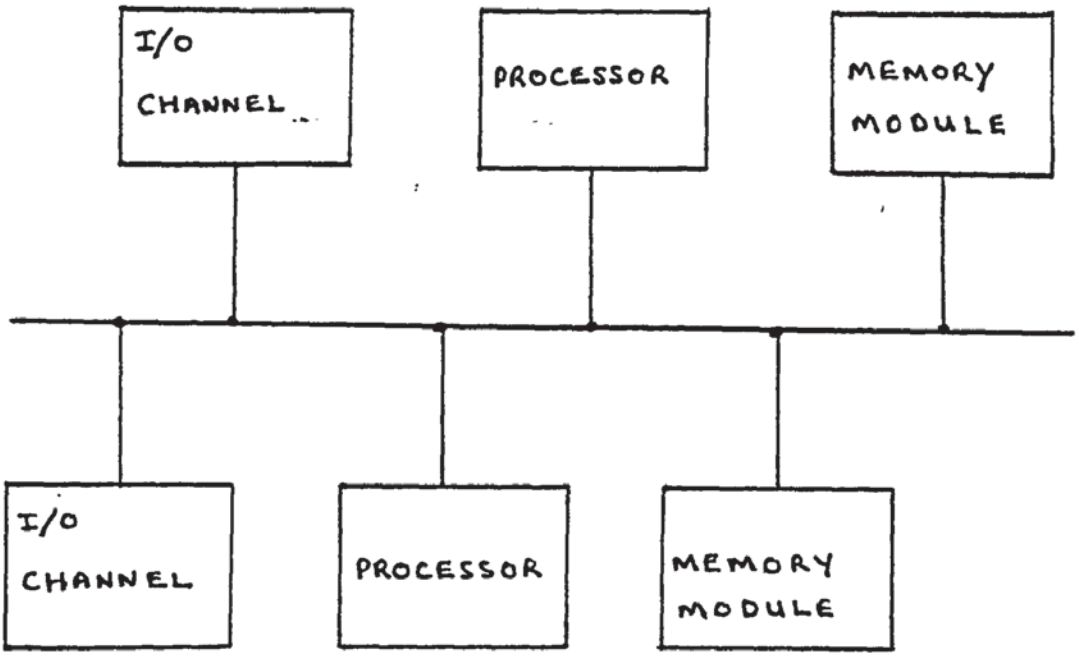
This is the least expensive switching system. It takes advantage of the memory registers in each processor and each memory module to allow the bus to be time-shared. Instead of a processor being connected to a memory module permanently, they are connected only for the time required to transfer the necessary data. This technique can prove especially useful if memory accesses can be pre-planned, such as in sequential instruction fetches and data fetches.

This type of organisation can be found on the Univac Larc (E1), IBM STRETCH (B49) and is also implemented in the communications between the peripheral processors of the CDC 6600 and its main memory (T5).

The advantages of this type of system are its conceptual simplicity and its flexibility for growth.

Its disadvantages are that it is only able to transfer one operation at a time which can result in unacceptably long waiting times, as the system grows and traffic becomes heavier. Thereis also the danger of catastrophic failure if the bus contains active components or if there is a single bus control unit.

Figure 4: TIME SHARED BUS SYSTEM

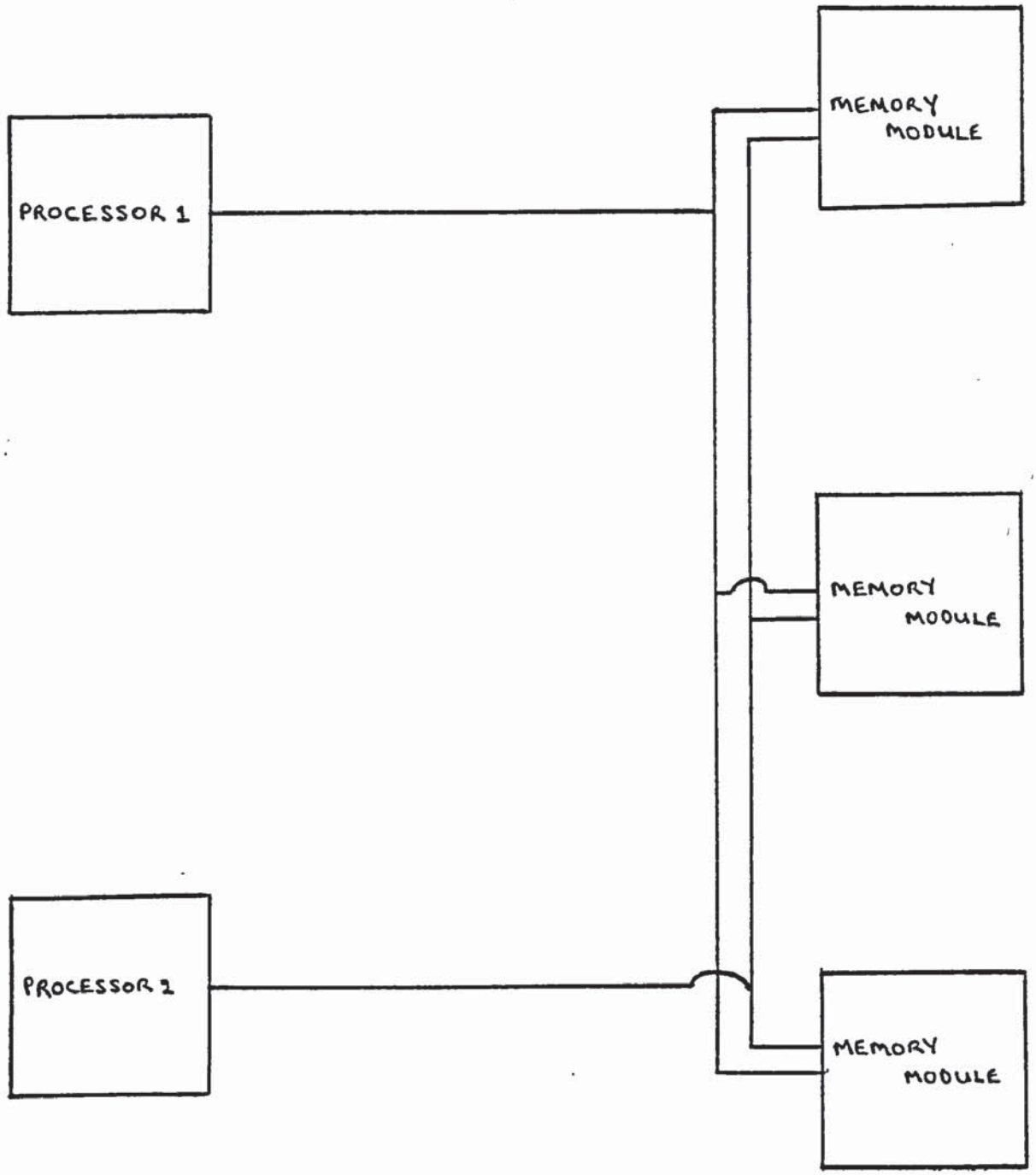


Multiple bus (see Figure 5)

This is the most commonly used method of memory access control. The multiple bus system is an extension of the time shared bus, and contains more than one simultaneous transfer path in order to overcome the time-sharing and hence the delays which occur in a single bus system. From Figure 5, it can be seen that each processor has access to any of the memory modules via its own buses. The control in this system must resolve queueing problems since two processors might attempt to enter one module at the same time. This problem is often overcome by having a priority physically associated with the connecting port. This type of system has been implemented on the CDC 3600 (C25), the Univac 1108 (S26), the Philco 213 (B35), Multics (C23), IBM 360 (B23) (G3) and the IBM 9020 (B24).

The main disadvantage of this system is that the number of buses is restricted by the number of ports available on memory, and this in turn restricts the flexibility of the configuration and its ability to grow.

Figure 5: MULTIPLE BUS SYSTEM



Crossbar Matrix switch system (see Figure 6)

In this system any memory module can be connected to any of the processors or any i/o unit. A full time connection is established between the units for the duration of the transfer.

"Tremendous flexibility is obtained in a crosspoint switch since any processor can connect to any memory in a fraction of a microsecond. Also, there are numerous ways to provide a function in case of failure in any part of the system" (C21).

This type of system has been implemented on the RW-400 (P11), the Burroughs D825 (A12), the Prime (E7) and the CDC 6600 between the peripheral processors and the i/o channels (T5) and on the Burroughs multi-interpreter system (D3).

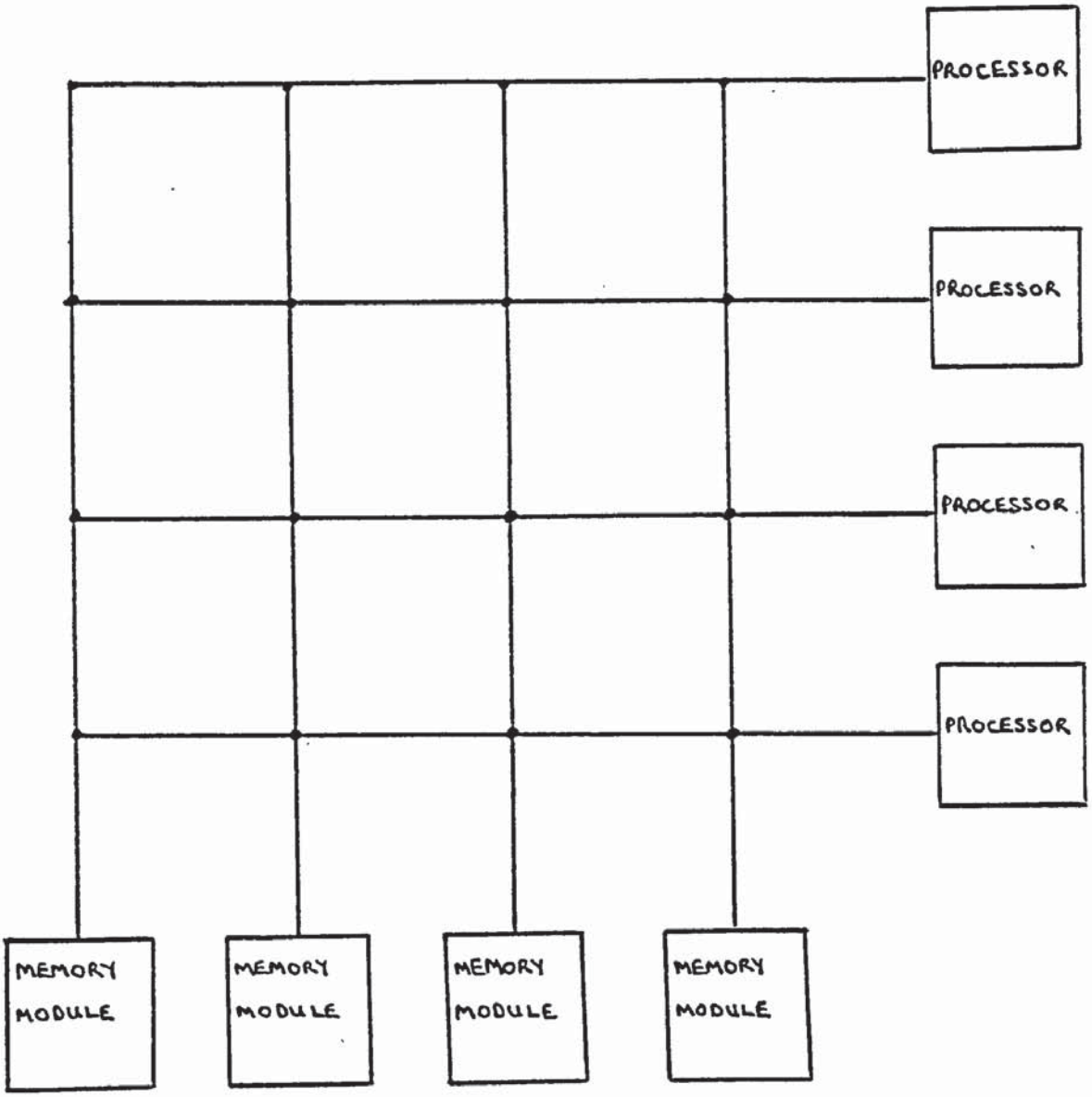
The advantages of this system are that:

1. it is relatively easy to add modules without much re-programming, if the switch matrix is large enough
2. individual unit interfaces are quite simple since they have to perform neither conflict resolution nor recognition of which data is directed to them, since these functions are controlled by the logic of the switch matrix.

The disadvantages of this system are that:

1. the larger the number of processors and memory modules, the larger the amount of circuitry is required and this can become too costly
2. full reliability for the system requires a duplication of the matrix

Figure 6: CROSSBAR MATRIX SWITCH SYSTEM



Multiport system

If a memory is kept whole with only one data register and address register, it is only possible for one processor at a time to access any part of it. This would prove very inefficient on a multiprocessor system and thus usually the memory is segmented into modules each with its own data and address register. This method may also prove inadequate for certain purposes as only one processor can access each module at a time.

The multiport system approaches the problem from a different angle. In this memory access scheme, the memory is not segmented but is equipped with more ports, i.e. more sets of data and address registers, in such a way that more than one processor can access memory at a time. The logical circuitry required to accomplish this is, however, very complex. For two ports the amount of circuitry needed will be more than double that normally needed to a word of memory with one port. There is also the need to include more circuitry to ensure that two processors do not access the same word during the same memory cycle.

The advantage of this type of organisation is that there is no need to check whether a processor is already in the part of memory required by another processor. Thus, there is less restriction to the operating system.

The disadvantage is the excessive cost of the hardware if several processors are required to operate in such a configuration.

2.5. Questions raised concerning the effects of multiprocessor systems

Having made a survey of the effects of multiprocessors, outlined in sections 2.3. and 2.4., questions arise concerning the merits of the various proposals, which are often conflicting. These questions concern the relative performance of the various strategies within a multiprocessor system and the performance of multiprocessor systems in general.

The type of questions, which require answers in order to assess the value of multiprocessor systems, are outlined below:

1. What should the ratio of memory to the number of processors be? Would this ratio vary according to the memory access scheme being used on the system?
2. Which scheduling strategy will perform most effectively in a multiprocessor system? Is the choice of scheduling stretegy as critical in multiprocessor systems?
3. How does the system overhead of the various memory management and access schemes affect performance?
4. Which processor should be interrupted? Which scheme for interrupting processors balances out the workload most effectively?
5. How many jobs are required to be on the system to give almost full utilization to the number of processors?
6. Which memory management scheme will prove to give the best utilization of available memory?
7. By how much would the addition of an extra i/o channel speed up system throughput?
8. What should the ratio of channels to processors be?
9. Does the continual addition of processors to a system eventually become ineffective? If so, at what point?
10. By how much does the addition of an extra processor speed up system throughput?
11. Which of the various strategies are most compatible and give best results?
12. In which areas of the system do the possible bottlenecks lie? How can these be improved?
13. If a system manager wishes to process an N% increase in workload how can this be most easily and economically accommodated?
14. How does a change in workload affect the performance of a multiprocessor system?
15. What effect on performance has the failure of one processing unit?

Chapter 3 EVALUATION TOOLS

3.1. Introduction

In the previous chapter, many questions were posed concerning multiprocessor systems. In order to answer such questions, it is necessary to evaluate the different algorithms and configurations together with their effects upon the total system performance. It is questions such as these which concern designers and system managers, and the need for performance evaluation, specified in section 1.1.1., must be satisfied by the evaluation tool chosen for the purpose. That is, the evaluation tool must be applicable to the three areas of:

1. design
2. comparison
3. extension

The tool must have the capability to give a general system profile including such things as percentage of total time the C.P.U.'s are busy, waiting on in supervisory state, channel overlap, memory usage etc. That is, it must be able to specify multiple resource systems and simultaneous events.

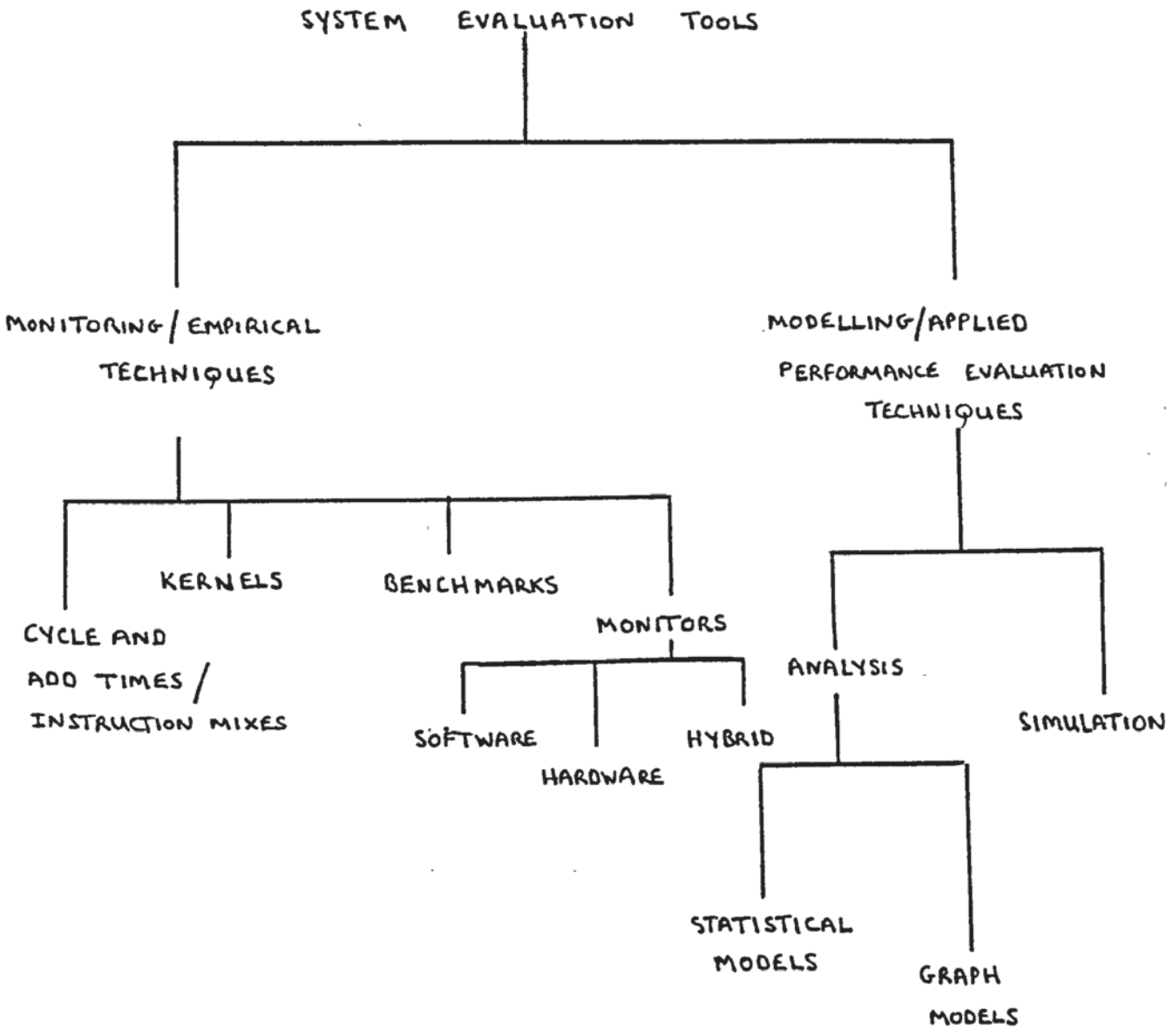
There are several evaluation tools which have been used in performance measurement and these can be categorised under two headings:

1. Empirical techniques
2. Applied performance evaluation techniques

A useful bibliography based upon the above classification is the work of Agajanian (A7). Gotlieb and MacEwan (G15) prefer to class the tools under the headings of monitoring techniques and modelling techniques respectively, but, unfortunately, do not consider all the tools available. An adaptation of their breakdown of the classes of tools which includes all the tools is given in Figure 7.

Four excellent overviews of evaluation techniques are (L13), (C1), (E9), (G22).

Figure 7: SYSTEM EVALUATION TOOLS



3.2. Monitoring/Empirical techniques

3.2.1. Cycle and add times/instruction mixes

Early attempts at computer performance measurement involved simply measures of the clock speeds of the computer (R1). Typical examples are the C.P.U. speed, add rates, memory speeds. Sharp (S12) exhibits formulae based on such parameters. Unfortunately, such formulae take no account of parallelism or the asynchrony of modern systems but rely upon the assumption that the higher the speed of the computer, the faster the workload will be processed.

The first serious attempt to provide a formal basis for comparing computers was a list of several such characteristics of every computer; this was published by Adams Associates (J5). This list was termed the Adams Chart. Its limitations lie in its basic assumption described above. Johnson points out that (J5),

"it has been well established that the use of the raw speed parameters of clock speed, arithmetic speed, memory speed, word size, or i/o rate can be misleading in predicting comparative performance between different systems."

This is due to the fact that such measures pay no attention to the effectiveness of the system software, architecture or to the factors such as concurrent or simultaneous operation for evaluating multiprogramming or multiprocessing.

The Auerbach report was a slight improvement upon this study, being an instruction mix approach. It listed the performance of computer systems on a set of small standard problems. The limitation of this research is the assumption that the performance based on these standard problems reflect the actual performance on a large workload.

A different approach to instruction mix measurement was proposed by Gibson (G4), in which the time taken to process a particular instruction

was multiplied by a weighting factor representing the frequency at which that type of instruction was expected to occur. This approach, although simple, again equates performance to the speed of the C.P.U.

To summarize, with a quotation from Johnson (J5)

"none of the instruction mixes yet devised provide a reliable indication of full load behaviour of computer systems. This is an especially important shortcoming when attempting to predict the performance of multiprogramming and multiprocessing systems".

This view is also supported by Osterberg (O6).

The following references contain examples of instruction mixes plus the situations in which they are applicable (M24) (R1).

3.2.2. Kernels

The use of kernels as a means of performance measurement is a more sophisticated version of the instruction mix technique.

A kernel is a complete nucleus problem, and is devised according to the typical applications which can be carried out on a computer system. The execution time of a set of kernels is assumed to represent the execution time of a typical set of applications.

The job usually consists of a short cyclic procedure, a typical machine-independent nucleus problem, which is executed a number of times and is referred to as kernel.

Some typical kernel tasks are matrix inversion, polynomial evaluation and square root approximations. The work of Buchholz (B48) gives an example of a situation in which kernels have been used. Calingaert (C1) points out that the kernel has a significant advantage over the instruction mix technique, in that it allows all available features of the C.P.U. to focus on a particular problem. On the other hand, the kernel problem must be coded by a programmer experienced on his own system and hence this approach is difficult to validate since it depends upon different programmers' skills. Also, kernels are generally isolated tasks and are not drawn from the workload of the system. It is this fact which leads Calingaert to say:

"There is strong evidence, especially in scientific computation that presently identified kernels are all atypical" (C1)

This technique is capable of measuring more parameters than those previously described, because it utilizes a computers instruction set more fully, and only those characteristics to which the kernel program is sensitive can be evaluated.

3.2.3. Benchmarks

Benchmarks differ from kernel programs in that they are actual programs and not artificial ones. They are existing programs taken from the application programs submitted to systems and are highly typical of the computer's workload. A set of such benchmarks is run on different computers with a view to comparing their performance. They can be used to evaluate such factors as hardware, software, speed of compilation and execution, and the overall ability of the system to perform with such a workload.

"Benchmarks have been widely used - - - - and a number of large organisations use this technique as a standard procedure." (06)

The benchmark approach is used primarily as a measure for marketing purposes. Its limitation is that it takes a large effort to program representative benchmarks and to prepare realistic data (H26). There must also be enough benchmarks to test the system at capacity level. Any problems not tested by the benchmark are often extrapolated and passed as performance predictions but this type of short cut can be misleading. McLain (M17) holds the same theoretical objection to extrapolation from benchmark experiments:

"the criteria is arbitrarily tied to a particular job mix; if the mix is an existing one then it arose on an existing configuration, and was to some extent adapted to fit that configuration".

Brown and Saunders (B45) and Wichmann (W10) (W12) have used this type of technique for comparison of systems and their compilers. Smith (S22)

gives a good review of benchmarks (and instruction mixes) and the type of situations in which they are applicable, and Adams et al (A6) explain the use of benchmarks for evaluating time-sharing systems.

3.2.4. Monitors

3.2.4.1. Introduction

A number of monitoring tools, useful for system and program measurement, have been available for some time. They include such rudimentary instrumentation as system and wait lights; system loop and accounting routines; traces, snapshots and other debuggingfacilities. Such techniques can yield a considerable amount of information concerning the behaviour of a system or of an individual program. More sophisticated, and more expensive, are the special purpose monitors that have become commercially available over the last few years. (B44) (W17) (H27) (C22) (A17) (B28) (H4) (S1). It is the latter to which the term "monitors" is usually applied.

The performance data gained from monitors can be analysed to locate the more crucial system areas which are limiting performance and causing bottlenecks. The information obtained can give a good indication of the degree of improvement which could be expected by making changes to the software or reorganising the hardware. Although this search for bottlenecks tends to be the principal reason for monitoring, these techniques can also be used to obtain a system profile. Such profile data if often necessary as a basis for both analytical and simulation models.

Monitor data can be obtained either by hardware or software techniques or by a hybrid combination of both. However, monitoring techniques often generate extensive quantities of data which require lengthy post analysis. If the analysis or some form of data reduction is carried out within the monitoring a significant overhead can arise, which can distort the performance of the system under observation.

When system improvements are undertaken on the basis of monitor information it is important to perform the same measurements after the introduction of the changes, in order to assess the effectiveness of the improvements attempted. The successful removal of one bottleneck may have created another elsewhere in the system.

The advantage of monitors over the previous techniques is that they measure the performance of a system whilst it is processing its actual workload.

One problem of monitoring is that of collecting representative data concerning the performance of the system, such that it is typical of the full range of demands on the system. This implies a constant monitoring of the system, which leads to the problem of assimilating and interpreting the large amount of data which would be generated in doing this. Monitors should therefore be used in experiments with some specific purpose, such that only the data required for the experiment need be collected.

For further details refer to the following surveys concerning monitors - (R8), (E2), (J2), (K2) (L12).

3.2.4.2. Hardware monitors

Hardware monitors consist of a number of electronic probes, which are attached to the appropriate back pins and test sockets of the measured hardware, and some sophisticated logic for interpreting the signals detected.

The simple monitor may only be a set of digital counters with some form of output meter and, in fact, some manufacturers are now building these into their equipment. For detailed performance measurement, though, their use is minimal.

More advanced forms of hardware monitors are available which place the data they have accumulated onto magnetic tape for later analysis, and often have combinatorial logic patching facilities so that more complex hardware conditions can be recognised. Such logic capabilities allow the

monitor to record simultaneous events such as C.P.U. idle/channel 1 busy (G6).

Thus, there are basically two types of hardware monitor (B1C)

1. Summary type devices
2. Dynamic type devices

The summary type devices record the time or occurrence of an event within the system and accumulate the total time or total number of occurrences of that event during the period of observation. The output is therefore how long or how often that event occurred without regard to when that event occurred. The dynamic type of monitor records information each time an event occurs and outputs the subsequent information to magnetic tape for later analysis.

Some various uses and developments of hardware monitors can be found in the following references (S18), (S6), (D17), (G23), (C11), (B28), (K12), (M27), (A15), (E10), (H5), (H27), (A17), (H4).

Hart (H4) describes several hardware monitors now commercially available; examples of these are the Computer Performance Monitor II (CPM II) produced by Allied Computer Technology, and the System Utilization Monitor (SUM) produced by Computer Synetics. Shrimpton (S18) used a hardware monitor to measure the George III operating system and UCLA developed a hardware device known as the SNUPER Computer (E10).

A history of hardware monitors is given by Warner (W1) and in another paper he presents a case study for hardware monitors (W2).

The advantage of the hardware monitor is that it is independent of the system it is to measure. Hence, it does not interfere with the performance of the system.

However, the nature of hardware monitors restricts them to providing data concerning only the basic operations of the system, particularly to input/output operations. Software operations can only be measured when they result in some detectable hardware effect. Hence, such

data as the length of operating system queues falls outside the range of hardware monitors. It is also difficult to feed the data gained from hardware monitors into models (R8). That is, the system profile they give is inadequate. The main disadvantage of hardware monitors is that they cannot relate the hardware effect to the software programs or tables, thus "the main drawback is that it tells you what is happening but not why" (G6).

3.2.4.3. Software monitors

Software monitors are extra pieces of code which are run on the system which is to be measured, with the purpose of collecting data concerning the performance of that system.

Software monitors generally fall into two main categories: they are either external sampling monitors driven by timer - interupts or internal embedded monitors activated by hooks which generally operate on a continuous basis (I10). The external monitor uses the timer-interupt to get control and then records data concerning the programs running on the system, the queue sizes etc. It is therefore very dependent upon the operating system. The internal monitor consists of a number of data collecting routines which are called from hooks embedded in the operating system or a user program. The hooks are associated with specific pieces of code, for example the disk access routine, and thus the data recorded is specific to those events.

A general overview of the nature and characteristics of software measurement tools can be found in the work of Kolence (K21) and various uses and types of software monitors can be found in the following papers (B53) (C16) (C4) (C2) (D4) (D10) (H5) (C22) (B44) (W17) (H20) (P8).

Some Commercial software packages are described by Hart (H4). The most commonly mentioned package is SMS, which is a family of software measurement products designed to operate in the IBM 360/370 environment and which was developed by Boole and Babbage Incorporated. This consists of two main programs, one of which is the P.P.E. (Problem Program Efficiency) which deals specifically with program efficiency and the other is C.U.E. (Computer Utilization Efficiency) which deals specifically with system efficiency. Hughes (H23) gives a detailed account of this software monitoring package, which is an example of the external type of software monitor.

IBM have developed a software package called S.I.P.E. (System Internal Performance Evaluation Program). This is a software monitor program integrated into the IBM 360/67. Time Sharing System resident supervisor. S.I.P.E. functions as an extension to the supervisor and records events as they occur. A number of hooks are placed at selected points in the supervisor code and when control reaches one of the hooks, S.I.P.E. is entered to record in a buffer the occurrence of the event, together with any other useful information, such as the time of the event. Degradation to system performance due to the interference caused by S.I.P.E. varies from 1% to 30% depending upon the events traced. S.I.P.E. is an example of the internal type of software monitor.

The advantage of the software monitor is that it can provide a higher level of information than the hardware monitor.

However, one disadvantage of the software monitor is that it interferes with the system it is measuring; it adds extra processing time and extra storage space. The degradation in performance due to the S.I.P.E. monitor is a good example of this. The problem which therefore arises is, how can this type of tool give the performance data of a system in normal use? Another disadvantage of this evaluation tool is

that it is particular to a specific manufacturer or operating system. It is also limited to observing variables that are visible to the operating system and therefore cannot record many aspects of machine operation, and hence cannot measure overlap directly.

Thus, the software monitor gives nearly all the information needed concerning system software, but there are occasions when hardware monitoring can be of assistance. In fact, the advantages and disadvantages of software and hardware monitors are complementary and this seems to point to the combination of the two techniques.

3.2.4.4. Hybrid monitors

A number of people have successfully combined hardware and software techniques in an attempt to combine the advantages of each. (S28) (R11) (S9) (B29). Such monitors are called hybrid monitors.

The main advantages gained are that the recording of data causes less interference to the system being measured, but, by using software, the recording of data can be made more intelligent and data relating to named jobs or named data sets is provided.

Suggestions for further reading concerning hybrid monitors are: (H22) (M3) (M26) (S1).

The use of a mini computer as a hybrid monitor, with software control instead of hardwired logic, is proposed by Aschenbrenner et al (A19). This is an attempt at obtaining greater flexibility in that the parameters being monitored can be changed dynamically during the monitoring process. Another system provides a graphics package so that the evaluator can actually observe system performance on line (S1) (G23).

Ideally, the hybrid monitor should consist of a software component running on the measured system which communicates with an external hardware component. Two such systems have been implemented (S9) (E10).

3.2.5. Suitability of monitoring/emphirical techniques

The early monitoring/empirical techniques, that is, cycle and add times, instruction mixes and kernels, are too simplistic for the measurement of multiprocessor systems and have other associated disadvantages as described earlier. Benchmarks, although more sophisticated than these above, are still restricted in the type of results which can be collected. Software and hardware monitors also have disadvantages associated with each, but do provide a clear picture of the performance of a system processing its own workload, and can record nearly all the data required to evaluate a system.

However, there is one common problem associated with all the monitoring/empirical techniques. This is the fact that they all require that the systems they are to measure already exist. A competent evaluation tool must be of assistance at the three stages indicated earlier: design, comparison and extension (see section 1.1.1.). The techniques thus far described, because of their requirement that the system they are to measure must exist, can possibly be of assistance only in the latter two cases. If the fact that they also include no predictive capabilities is taken into account, it would seem that they are also not suitable in making decisions concerning the extension of a system, since the extrapolation of data from a system, to predict what will be the case if the system is altered, can be dangerously misleading.

Therefore, monitoring/empirical techniques are not suitable as general tools of evaluation. It is essential that the tools include the capacity for prediction and make no assumptions concerning the existence or non-existence of systems.

3.3. Modelling/Applied Performance Evaluation Techniques

3.3.1. Introduction

Modelling techniques include the two capabilities which were lacking in monitoring techniques. The models, once developed, can be altered to represent changes in the actual system in order to discover the effects of those changes. This facility provides them with predictive capabilities. Also, the model is totally independent of the system it is to measure and hence whether a system exists or not is irrelevant to the development of the model.

Modelling techniques have been used successfully in many areas of application, such as factory procedures, transport decisions, airline predictions etc.. Besides assisting with decisions in design, comparison and extension in these various applications, they also help to provide a clearer picture of the functioning of the system they model. Hence, it is not surprising that these techniques have been widely used to evaluate computer systems.

There are two main techniques employed in the modelling of computer systems: analysis and simulation. A good review of computer system models, particularly in relation to other evaluation techniques is that of Kimbleton (K8).

3.3.2. Analytic models

The analysis method of evaluation of performance provides probabilistic models of systems and processes and also discrete graph models of programs. Graph models of systems are not normally suitable for use in evaluating performance, but are used primarily as a form of description. Such graph models often take the form of Petri nets (P6), whose main objective is to represent concurrent conditions which can exist in a system. An example of this descriptive use is the work of Noe and Nutt (N8) in which they describe the IBM system 360 and the CDC 6400 using graph models.

The graph models of programs have been used in an attempt to study behaviour within a system and hence the systems throughput. Throughput analyses are made by decomposing the sets of programs to be run concurrently (or in parallel) on a computer into those segments which first can be processed sequentially and then secondly arranging those segments with other segments which can be processed concurrently (or in parallel). This basic type of model is called a directed graph. Martin and Estrin (M8) give a summary of graph theory and its application to throughput analysis. Weilgosz (W13) gives the history and background of graphs models of computation. Suggestions for further reading are; (B31) (C6) (D8) (H1) (H17) (H15) (K1) (M8) (G14) (B1) (V1).

To summarise, the purpose of graph models is to analyse and/or describe systems and program behaviour; they are not designed for evaluation or measurement. However, graph models are often used as a form of input into an evaluation model (W13).

The type of analytic model used for evaluation purposes is the probabalistic or statistical model. In this type of model, the system is broken down into a set of queues for various resources and through the use of queuing theory or Markovian processes the probabilities of certain events occurring are calculated. Hence, the model is a mathematical representation of a computer system or part thereof. Such models can be useful in the study of certain limited areas where performance bottlenecks may occur or as aids to general design decisions.

Four general papers on statistical modelling are Berners-Lee (B19), Hansmann F et al (H3), Freiberger (F21) and McKinney (M16). Good presentations of some useful methods of queueing theory can be found in the work of Martin (M11) (M12) and Denning (D7). Mathematical models have been widely used, but the complexity of computer systems, with many interacting system parameters, tends to prevent the construction of

complete models that are mathematically tractable, and most models are limited to subsystems or specific functions. Examples of these are processor scheduling (M16), (M17) multiprogramming affects (L10), and program behaviour (D5), models of memory systems and core space allocation (W14), and secondary storage devices such as drums (A2) and disks (A1) (F19). Models of batch processing systems are considered by Berners-Lee (B19), and Bard (B11) used statistical methods to evaluate the CP-67 operating system. Knight (K18) developed a mathematical model to evaluate the effect of failure in certain units of hardware or general software failures. Kasper and Miller placed their emphasis on modelling the job stream (K3).

Further reading concerning the use of statistical models is; (F22), (F17), (F13), (S4), (S5), (K13), (K14), (K15), (L10), (R5), (B18), (G1), (B22), (C13).

As mentioned earlier the goals of these analyses are both insight and quantitative results to influence the design of systems and resource allocation. While most would argue that these goals are inherently worthwhile and must be pursued, there is widespread dissatisfaction with the current state of the field of statistical analysis. Basically there are five major areas of dissatisfaction with such analysis methods, which are summarised by Thomas (T4).

"Usually only a limited part of a computer system is modelled theoretically and even then many simplifying assumptions need to be made in order to permit a mathematical analysis, which often relies on queueing theory. The workload or input to the model must also be described in terms suitable for analysis ..... It is difficult to assess the validity of the results obtained from such models ..... Often the results are dubious in view of the many assumptions which must be made."

The first limitation of statistical analysis is the fact that the workloads might not be the exponential functions assumed to facilitate analysis, and that it is not possible to predict the service times that a computer will require to execute aribitrarily given tasks.

The second failing is that analytical results are not often validated by measurement or simulation. Moreover, in cases where system evaluation studies are carried out, the existing models do not seem powerful enough to provide a uniform basis for measurements. The extent of this failing is shown by the fact that Agrawala and Larsen (A8) admit that analytic results are not very accurate and that they are only useful for determining the general behaviour of systems.

The third major criticism is that most of the literature on analytic modelling is a collection of analysis of specialized models. This points up the lack of general, powerful results which would allow analysis to become an engineering tool. As the situation is now, each new application almost always requires a separate analysis by an expert. Hence, there is great difficulty when attempting to compare various parts of systems with other systems (or parts thereof). Osterberg (06) points out that its use in design work is limited.

The fourth failing of statistical analysis is concerned with the fact that the systems to be evaluated are made to fit the models and not vica versa as should be the case. The use of queueing theory is limited to rather simple situations which means that approximations are often used in order to make the problem fit inside the confines of queueing theory. The basic assumption underlying other analyses is that the model is Markovian. Real computer processes are not. Consequently, it is not easy to use this kind of model to determine transaction delay, the effect of interrupts, channel interference, queue build-up and dynamic memory utilization. Transient behaviour cannot be modelled at all by this method.

The final and most criticised aspect of statistical analysis is that the models are generally oversimplified in order to make them mathematically tractable. This obviously restricts the extent of the use of statistical methods and makes the results gained questionable.

"All but the simplest queue systems defy analytic solution, although the mathematical theory of queues is a powerful tool for discerning the kind of phenomenon that can arise". Tocher (T7).

Much analytic work, therefore, has dealt with single resource models. The reason for this is clearly that most of the analytic tools which are available apply to single resource environments. Computer systems though are multiple resource systems. The view that analytical models are inadequate for comparing and testing operating systems is supported by Enslow (E7).

There are very few examples of statistical techniques being used to evaluate a multiprocessor system, which implies that they are not suitable techniques for such complex systems. The three papers which have been found on this subject are (I8), (B22), (F12).

Hence, while the models can be very sophisticated in the mathematical sense, it may be such an oversimplied representation of the real system as to limit the value of the results it produces. Estrin and Muntz (E9) are of the same opinion:

"Much of the analytic work has been concerned with exact mathematical solutions to models which are themselves gross approximations".

In conclusion, then, it would seem that analytic modelling is not a suitable technique for evaluating such a complex, multiple resource system as a multiprocessor. It would seem that simulation is a more powerful modelling tool than analysis.

"Analytical modelling and simulation have often been regarded as opposite and incomparable techniques, and analytic modelling has to a

certain extent suffered in reputation as a result of the comparison. Certainly, the analytical approach does not normally yield results as accurate as those of simulation, and the analytical modelling of complex systems can become inordinately cumbersome". Infotech report (I10),

3.3.3. Simulation

3.3.3.1. Introduction

"Eventually the complexity of the mathematics becomes so great, the techniques so obtruse, that it is a rare individual who can master both the technology of the system being analysed and the mathematics required to analyse it.... At that point an experimental approach becomes more effective than an analytical approach. The primary experimental approach is simulation." Beizer (I10).

Because the analytic technique seems unsuitable for evaluating such a complex system as that of a multiprocessor configuration and the operating system which accompanies it, simulation would seem to be the tool most capable of achieving a solution to the problems which have been outlined in the previous chapters.

There are two broad categories of simulation models, the continuous simulation model and the discrete event models. The continuous models simulate gradual changes in compositions or states, for instance, fluctuations in the concentration of liquids passing through a chemical mixing plant. The discrete event models simulate events taking place within a system and their effect upon other events. Since the existence or non-existence of an event depends upon the effects of other events it is not possible to predict the outcome in any detail, except by means of simulation. All computer system simulations are discrete event models.

Within this category, two main sub-divisions have been distinguished, according to the nature of the input. In one case inputs are generated by random sampling from assumed probability distributions - stochastic simulation -, whereas, in the other, the input is presented

as a sequence of work steps, the nature of each step being explicitly defined - deterministic simulation -.

There are various levels of detail at which simulation can be applied. These range from micro-level simulation to macro-level simulation. In micro level simulation the effects of each machine language instruction are simulated. The contents of core storage, the details of channel operation, etc. are included. In order to run a simulation at micro-level, one unit of real time may take ten units of simulated time and hence it is expensive to run. It also requires a very detailed knowledge of the hardware and software of the system. However, such fine detail is often unnecessary, since macro-level simulations can give suitably accurate results without the difficulty of development or the large run-time (P9).

Simulation is a tool which can provide quantitative information in a wide variety of areas. In some cases this information might be obtained by other means but it is easier and/or cheaper and/or more accurate to use simulation. In other cases simulation is the only suitable method of obtaining the information.

A simulation study does not give absolute results on which decisions can be made; rather it provides comparisons which can be used to give a quantified understanding of the performance of the system under various conditions.

Lucas (L13) discusses and compares all the evaluation techniques and comes to the conclusion that only simulation is a satisfactory technique for the three major areas in which performance evaluation is required. (See section 1.1.1.). The view that simulation is the best method of modelling systems is supported by Hooley (H18), Drummond (D21) and Critchlow (C25).

Suggestions for further reading concerning the simulation of computer systems are Huesmann (H30), Hutchinson (H29), (H28), Legman (L7), Scherr (S5), Ziegler (Z1), Nielson (N6), Naylor (N2), Tocher (T7), Martin (M10), Smith (S21), Emshoff and Sisson (E5), Teichrow and Lubin (T1), Krasnow and Kerikallio (K22), Oren (O4), Gorden (G12), Tryggestad (T8) and a useful bibliography is given in (B21) and (R9).

The reasoning behind the choice of simulation above the other evaluation techniques has several aspects.

The first is that simulation is a management science which is especially well suited to providing the decision makers of the computer industry with information concerning the performance of computer hardware and software. It is both an anlytical and predictive tool which can be applied throughout the evaluation of a computer system from earliest feasibility studies through design phases to eventual implementation. It is an extremely flexible tool which allows the user to examine not only the current state of his design but to also evaluate possible alternatives to discover an optimum solution to his problem.

The second is that simulation has been successfully used in many areas of performance evaluation with respect to computer systems. These are outlined by Von Almen (V2).

1. Feasibility studies
2. Conversion planning
3. Hardware enhancement
4. Hardware/software selection
5. Implementation
6. Growth planning
7. Systems design
8. System tuning
9. Workload scheduling.

The above breakdown includes successful applications of simulation in each of the three general situations given in section 1.1.1. in which the evaluation tool must be able to be successfully applied.

The third aspect is the advantages of simulation:

"Simulation can be applied to any part of a system depending on need". Critchlow (C25).

"Simulation models can emulate a computer system to any degree of detail required, the limiting factors usually being the man power and computer time required to develop and run the model .. . . Unlike analytical models they may more readily incorporate the operating system and other software elements as well as the random effects of interrupts and multiprocessing etc". Thomas (T4).

Hence, simulation is a powerful enough tool to deal with both single and multiprocessor systems, including their associated software.

"Simulation is one of the most thorough and effective of all the performance assessment techniques". (Osterberg (06).

There are also other advantages which simulation provides, these are:

1. an improved understanding of the problem
2. the ability to consider alternative solutions
3. no capital expenditure on the real system
4. no effect upon the performance of the real system
5. an effective training tool
6. the ability to include a large amount of detail about the the system being simulated
7. the ability to compress a long period of real time into an appreciably smaller amount of simulated time.

There are also disadvantages concerned with simulation models:

1. They are costly to develop and run
2. They use scarce and expensive human resources
3. They take a long time to develop
4. They require a large and fast computer.

However, the theory behind the simulator, developed as part of this project, is intended to overcome, or at least alleviate, all of the above disadvantages (see chapter 4).

3.3.3.2. Related work in the field of simulation

The work thus far carried out in the field of the simulation of computer systems can generally be categorised into three sections:

1. specific areas of system performance
2. the performance of specific systems
3. the performance of a large range of systems.

Most uses of simulation are concerned with the first two sections, and there is a great deal of literature concerned with these two areas. However, much of this literature refers only to single processor system simulation. This is probably due to the fact that multiporocessor systems have only recently made an impact on the science.

Specific areas of system performance

Many simulations are restricted to a detailed analysis of some particular strategy in a computer system. For instance, Schedler and Yang (S14), Nielson (N4), (N5), Pooch (P10) simulated the effect of different paging systems and their effect on memory utilization. Terman (T3) modelled interleaved memory systems for the IBM 360 and 370 architecture and Sherman and Brice (S17), (B34) modelled virtual memory and its effect on i/o buffering. Burris (B50) deals with security problems, Fine and McIsaac (F15) the SDC Timesharing System to test different scheduling algorithms and Levy and Cann (L9) reliability. Finally, there is the work of Raynor (R6) which deals with processor scheduling and different interrupt algorithms.

Such studies outlined above can provide valuable information concerning the performance of the various alternatives, and many of the specific areas of system functions have been evaluated via simulation. The information they provide though is totally independent of the effects these alternatives may have on the system into which they are to be included. That is, they are devoid of context. As has been mentioned earlier (see section 1.1.2.) it is not enough to use the performance of subsystems, since the important factor in todays complex systems is the interaction between these various techniques in a larger environment. System-tuning studies employing this approach may alleviate a bottleneck in one area of the system, only to find that another has been caused elsewhere, which the analysts are not aware of. Hence, when designers begin work on a system, the knowledge provided by simulations of restricted

parts of the system is valuable but it is not sufficient. To define an "efficient computer system" is impossible since it depends upon the goals of that system. The goals see the system as a whole and relate to how the resources interact to produce, for instance, a good throughput time, a good utilization of processor time, a bottleneck-free system etc. Only when systems are compared as a whole can the designer be certain that his decisions, concerning which alternatives to implement, are correct. What may prove a better strategy in one context may not in another.

The performance of specific systems

Many of the projects in this category have been developed by manufacturers in order to acquire data for marketing their systems, or by computer system managers wishing to discover a means of improving the performance of the system at particular installations.

Examples of the work on specific systems are Merikallio and Holland (M20) which was restricted to an air traffic control system (multiprocessor system), Hutchinson and MacGuire (H29) who simulated the Univac 1107 system to determine the effectiveness of various peripheral devices and buffering techniques, Spiegel (S23) whose work concerned a data retrieval system, Lehman and Rosenfeld (L7) which was restricted to the IBM O.S. 360 (MVT), Burroughs Corp. who developed a simulator for their B5500 machine (B52) and Winograd (W18). Katz (K4) simulated the IBM 7040-7090 DCS multiprocessor system, and although his model is restricted to this system, it contains some useful pointers relating to the type of data that should be provided by a multiprocessor simulation.

Many manufacturers have developed simulation models for their systems; an example of this is the Univac 1108 model. The disadvantage of manufacturers models is that only one range of equipment can be evaluated.

A good example of the goals of this type of simulation study is the work of Cantrell and Ellison (C4), who simulated the GECOS II operating system. They stated their goals as:

1. find out where the performance bugs are
2. use this knowledge to avoid such bugs when the system is extended.

These goals are typical of this type of simulation and are too restrictive. They take no account of whether the extended system will create new performance bugs, whether a totally different configuration might prove more efficient, and do not extend simulation as an engineering tool, since the model developed is restricted to the system at that particular installation. The main disadvantages of simulation are related to the fact that a simulation model takes so much effort to develop. Bearing this in mind, it would seem wasteful to develop a simulation for a particular installation, since, once that system has been evaluated, the simulation is no longer applicable. Another restriction implicit in this type of simulation is that it often evaluates systems which already exist and thus overlook the use of simulation as a design aid.

The performance of a large range of systems

There are very few examples of this type of simulation study, although it overcomes the disadvantages of the previous models in that various techniques are evaluated within context and the effort to develop the model is not wasted on a short-term application, since the simulation is applicable to various systems.

An early example of this type of simulation is the model developed by MacDougall (M1) which applies to single processor disk-based multiprogramming systems.

Later attempts at this type of study take multiprocessors into account. Cohen (C15) describes a simulator, S3, which can be applied to various systems. In order to use the simulator, a great deal of

preparation is required. Peripheral devices, controllers, channels, memories and C.P.U.'s all must be described in detail. The system software also needs to be formulated. For this purpose S3 provides a formal programming language in which the flow chart logic of the system software can be expressed. It also provides a job control language which is employed by the user to specify file structures, application programs, data, re-entrancy, and the frequency of a jobs appearance on the system. This simulator is therefore very much deterministic (see section 3.3.3.1.), with everything, including workload, expressed in detail for input to the simulator.

A similar approach, again requiring detailed specification as described above is used in the CMS - Configuration Modelling System (C27), IBM's CSS - Computer System Simulator, SCERT and CASE.

Although the theory behind these software packages is essentially correct, that is that the model is general and can be applied to various systems and that both hardware and software can be evaluated together, certain objections to them can be raised:

1. The simulators require a great deal of effort in preparing input. This has to be formulated using specially designed specification languages or structures which will probably be unknown to the user, hence the preparation might prove very complicated.

2. The level of detail required as input will probably not be available at the earlier design stages. In such a case, the simulator cannot be used as a design aid, which is one of its important functions.

3. The user is restricted when he uses the simulator since all his options are pre-defined; the simulator can be seen as being static. Hence, any new strategy or piece of equipment may not be able to be dealt with by such a simulator.

4. The simulator is hidden from the user. Therefore he cannot assure himself that the system being simulated is that which he intended. Nor can the user adapt the simulator in any way.

"However, they (software packages) are obviously not ideally suited to every situation. One's own problem is never quite like any other; and there is always some reason why one can never use somebody else's model," O'Brien, Infotech report (I10).

This study accepts the basic theory of these simulators and attempts to suggest a methodology for overcoming the problems outlined. The methodology suggested attempts to develop a simulator which is easier to use and which requires less preparation; which allows various levels of detail to be included in the simulator; which is dynamic in that it can be adjusted by the user to fit his specific needs or to take account of new developments and in this way make the simulator visible to the user.

Chapter 4 THEORY OF S.C.O.P.E.

4.1. Introduction

S.C.O.P.E. (Simulation of Computer Organisations for Performance Evaluation) was designed as an example of the methodology, described in this chapter, which should take into account all the aspects and factors of performance evaluation which have been found lacking in this field of computer science.

As computer architectures and operating systems have grown in complexity, the need for sound methodologies of performance evaluation has become more pronounced. Many of the innovations that have produced major breakthroughs in system performance, such as independent input/output channels, multiprogramming and multiprocessing have also been largely responsible for major difficulties in the modelling and measurement of system performance.

Normally a system measurement exercise is undertaken in response to specific causes: the system has become overloaded, an equipment lease is pp for renewal and the future procurement policy is under review, a new capacity is envisaged. Although Ad hoc investigations of performance under such conditions are better than none, a more sustained approach is necessary if the performance of a system is to be managed systematically. A sustained approach should also make the investigations simpler to carry out.

The performance predictability requirements (outlined in section 1.1.1.) specify that the measurement tool must be applicable to three basic areas; design, comparison and extension. Most of the research into performance evaluation concentrates only on the latter two areas. This study, however, has found that, if the simulator is developed for use in design, reflecting current design trends, it becomes equally suitable for use in the other two areas and, in fact, achieves a

simple approach to those areas. Hence comparison and extension are seen as secondary in respect to design. This view is supported by Fox and Kessler (F18) who say that the postponement of measuring the performance of a system until it is checked out and fully implemented can be catastrophic. Too many irrevocable software design decisions will have already been made. In view of these considerations, they point out the obvious requirement for some technique which can predict the performance of various design alternatives.

4.2. Current trends in the design of computer systems

The current trends in the design of computer systems can be dividied into two main schools of thought, that is, the modular technique and the hierarchical technique. Further reading concerning these techniques can be found in (W25) which deals with the design of the Hydra multiprocessor operating system. This design was based on a methodology which is a hybrid of Dijkstra-style structured programming (D14) (D15) and Parnas-style modularisation (P2); (D11) which deals with the design of the T.H.E. system and is an example of the hierarchical technique; other references include (B9) (D13) (G10) (R10) (B33) (L11) (B38) (B43) (P4) (B13) (S3).

In a hierarchical design the hierarchical layers represent certain specific functions within the operating system and each layer relies implicitly upon the layer below it in the hierarchy. This has been criticised as being inflexible and thus the modularisation technique performs a similar division of the operating system into functions but dispenses with the hierachically ordered structure.

Thus, the two techniques both include the same purpose within the design, that is, some form of decomposition of the system into separate functions.

The reasoning behind this functional decomposition is that large systems programming is dominated by the integration and debugging problem. There is no doubt that programmers are fallible and always will be, and so it was necessary to devise some method of programming large systems which would:

1. prevent most logic errors

2. detect those errors remaining more easily than before,

and it was found that these capabilities could be achieved by using modularisation techniques. (M22) (M23) (S32) (B5) (I11).

The methodology is to write a specification of the system based upon the design, which serves as a skeleton for the whole program, using segment names where appropriate to refer to code that will be written later. In fact, by inserting dummy members into a library with those segment names, one can compile or assemble, and even possibly execute, this skeleton program, whilst the remaining coding is continued. Now the segments at the next level can be written in the same way, referring, as appropriate, to segments to be written later (again setting up dummy segments as they are named in the library). As each of the dummy segments become filled with its code in the library, the recompilation of the segment that included it will automatically produce an updated, expanded version of the developing program. Hence, once the initial skeleton program is formulated, each programmer or group of programmers, can be responsible for a separate segment and can work independently within the structure of the overall program design.

" - - - since the system will undoubtedly be a team effort, it is extremely important that clearly defined functional modules be established as the basic building blocks for the software system" (E7)

This building block approach has the advantages that:

1. separate groups can work largely independently on separate modules
2. changes can be made to an individual module without changing other modules
3. modules can be studied independently and thus a better understanding is reached. Better understanding leads to less errors and a better design
4. the communication problem within the design group is eased.

4.3. The methodology for using S.C.O.P.E. as a design aid

Abernathy et al (A3) carried out a literature search on the subject of design goals for operating systems. The first entry on the list of design goals they compiled was - maximize system efficiency. In order to do this, the tendency of designers to

1. incorporate design features from previous systems without assessment of their suitability to the new system,

and to 2. assume that optimization of individual system components will automatically lead to an optimization of the whole system,

must be counteracted.

Banks (B10) says that in order to overcome problems of this type the designer must have at his disposal performance information at each stage of the design. He must also have a pre-implementation evaluation so that he can meet the objectives of the system being designed.

" ... it will probably be a relatively simple task to specify what the operating systems should do, and it will not be too difficult to design the logic to accomplish these goals. The real problem is in verifying that the logic actually does conform to the desired results, and that it performs its work in an efficient manner" (E7).

In order to supply such a design aid the methodology of S.C.O.P.E. must reflect the methodology of system design. That is, it must reflect the fact that the system is composed of logical segments, each of which is designed to operate sequentially but whose interaction may be sequential or parallel, synchronous or asynchronous, consequential or logically independent.

When Many simulation models are built, the tendency is to incorporate too much detail. All the operations are described in minute detail and hence, when these are to be implemented into a single level concept, complex problems arise. This is the same failing which designers had and the reason why the design methodology, outlined above, was devised.

S.C.O.P.E. was built using a top-down approach, similar to that of system design, in which the system is specified at a grosser level, but the model is so constructed that, as the need for increased detail arises in certain areas, the model can be expanded to add that detail.

As in the case of system design, a skeleton structure of the system to be evaluated is provided, which is composed of segments or segment names and a main body which co-ordinates the execution of the segments. The representation of a segment is in terms of an algorithm, which, given certain input values produces the appropriate effect or output values, and an estimated time lapse for the activity. This permits a preliminary evaluation of the contribution of that segment to a total system performance. The segment thus specifies what the component does. The details concerning how this is done can be supplied at a later stage with a call from within the segment to other segment names, or by incorporating these details into the segment concerned.

With such a procedurised simulation model, the designer is able to have at his disposal an evaluation of the proposed system at any point in its design; the performance information produced being consistent with the degree of detail achieved in the design specification. The highest level will contain merely a skeleton of the system functions and their effects, whilst the lowest level will contain the detailed workings of each segment.

The other advantage of this approach is that modifications to the model can be made easily, since is is merely a matter of altering a segment, and thus changes in design can be compared and investigated before the level of detail at which they occur in the actual implementation.

A simulator, using the same methodology as that employed in S.C.O.P.E., could then be included into a Chief Programmer Team organisation (M23) (B5) (B6), whose project concerns operating system design and development. Chief Programmer Teams are becoming more and more common since they have proved successful after their introduction by IBM, for example BIS Applied Systems are undertaking a similar approach (Datalink 29.5.78, p.12).

The Chief Programmer Team is deliberately small, consisting of a chief programmer, three to five programmers, and a librarian. Other specialists may be included for certain functional capabilities.

The chief programmer is responsible for the development of the system and in this capacity will draft the initial skeleton structure as referred to earlier. He also produces all the critical segments in full and integrates all the separate program modules. He is supported by a back-up programmer, who is capable of standing-in in a technical and managerial sense. The final role is that of librarian, whose responsibilities include maintaining the records of the project in the development support library in both machine readable and human readable form. The records show the current status and the previous history of the project plus the result of the latest tests.

The simulator could be included in the library of such an organisation where the results of the various stages of evaluation would be maintained. It would be the responsibility of the back-up programmer to use the simulator and to supply his findings to the rest of the team via the library. In this way the project may be speeded up, since design decisions will require less deliberation and the effects of the design decisions taken can be predicted in advance of implementation. Hence, not only will the project be completed on time, which is the main aim of present Chief Programmer Team organisations, but the system designed should also be more efficient and should also represent the objectives of the design more accurately.

4.4. The use of S.C.O.P.E. in comparison and extension situations

As has been previously mentioned, the development of S.C.O.P.E., with design as its primary influence, does not hinder its use in other situations, but assists it.

For instance, if the system has been designed using S.C.O.P.E. then it will be a very simple matter to extend the system or compare various other hardware and software alternatives.

If this is not the case, the procedurised design helps to formulate an accurate picture of the system which requires evaluation. In the case of both comparison and extension situations, the different alternatives are investigated on the same basic model (there are not two wholly independent models of the alternatives to be tested). This allows more confidence in the results of the comparison.

Another important aspect is that the system(s) can be evaluated at various levels of detail. This is not usually the case with many simulation models.

4.5. Other factors influencing the development of S.C.O.P.E.'s methodology

4.5.1. Introduction

Some of the problems encountered in using general-purpose simulators have been referred to in Chapter 3. That is, that the user is not familiar with the package and has no means of assuring himself that the simulator is doing what he intends it to do. The user is also restricted in the functions which he can simulate (and the degree of detail to which those functions can be simulated), due to the fact that packages are static and their functions pre-specified. The other problem facing the user is the difficulty of preparing to use the simulator, which might be increased due to his lack of understanding of it.

"In addition, very few models are general enough and have application techniques and caveats well enough specified that a novice can use them successfully unless he has aid from the models developer" (B26).

Boehm and Bell in the above quote point to these problems which can be classified under the headings of the type of systems which can be evaluated i.e. the models must be general-purpose, and the ease with which the model can be used. It is these two other factors which have influenced the development of S.C.O.P.E.

4.5.2. The type of systems which can be evaluated

Due to the considerations outlined in section 1.3., the type of multiprocessor system which S.C.O.P.E. was designed to evaluate was that of symmetric processors or "true" multiprocessors, using a shared memory and shared input/output facilities. Since the most basic organisation of this type of system is one processor having total use of the "shared" memory, and since these systems included multi-programming, S.C.O.P.E. is also able to evaluate single processor multiprogrammed systems.

The range of algorithms and configurations within these types of systems which can be evaluated using S.C.O.P.E. is large since it is designed to be able to include all the proposals put forward in sections 2.3. and 2.4., which must be the case if it is to provide answers to the types of questions raised in section 2.5.

As was argued in section 1.1.2. S.C.O.P.E. simulates a system in total, since it is well known that some system parameters can often be improved only at the expense of each other, for example CPU usage and input/output channel usage. Without some global criterion, optimization may merely amount to pushing a bottleneck around the system, from one component to another. Boehm and Bell (B26) advocate total system simulation in their summary of the ACM/NBS conference on computer performance evaluation. Hence, S.C.O.P.E. not only takes account of the various hardware configurations but also various software algorithms.

"performance questions are less likely to be answerable without taking software considerations and hardware-software interactions into account" (N6).

Indeed, within S.C.O.P.E. it is the software which takes precedence over the hardware considerations, since in today's systems, the hardware is controlled by the software; the effectiveness of the hardware is thus dependent upon how the software applies it. This does not imply a lack of concern for the hardware performance, rather, it is a realization of the fact that the efficiency with which the software utilizes the potential of the hardware configuration is the major factor in determining total system performance.

4.5.3. Factors aiding the use of S.C.O.P.E.

Library of routines

Since the simulator is designed to be modular, the matter of adjusting some routine to investigate the effects of a different algorithm is not difficult. The segment which requires adjusting can either be replaced by another or merely altered if the adjustment is a matter of detail.

However, if the user is to make many changes it would not seem worthwhile to use such a simulator. For this reason a library of routines must be supplied as part of the package, which takes into account any important changes which the user may wish to make. Equipped with such a library of routines the user need only make minor adjustments in order to reflect the idiosyncrasies of his particular system. Each segment in the library should be well documented and the documentation should explain the differences between two segments of the same name, utilizing different algorithms and whether they are dependent upon other changes in the system specifically.

This technique also reflects some new design techniques since Wecker (W7) and Davis, Zucker and Campbell (D3) advocate such a procedure in the design of multiprocessor systems.

Davis, Zucker and Campbell describe the design of the Burroughs Multi-interpreter system. The software system is decomposed into modules of operating system functions and these are used as building blocks. The system has the ability to automatically select the appropriate units from a prestored library of such units. This technique permits using only those units necessary to perform all the desired system functions for a particular set of tasks.

Wecker takes a more general approach, and puts forward an interesting proposal. Since current operating systems reveal basic common functional elements, the categorisation and definition of these could create operating systems where the functional elements become independent modules of execution and the "operating system" merely a mechanism for communication and synchronisation of these modules. His paper then focuses on an "operating system base" which would be suitable for building various different multi function systems from these building block functional modules.

The main point of interest with respect to this study is that such a general method of design will allow the flexibility of being able to replace functional blocks with "equivalent" blocks that are better suited to the type of operating system function desired.

"By creating a library or inventory of specific process modules, the system designer can create systems by choosing appropriate modules and combining them in a 'building block' fashion" (W7).

The relationship between Wecker's library of modules and his operating system base to S.C.O.P.E.'s library and skeleton structure is easily recognizable. Should this approach to general design theory be adopted, and current trends do seem to point in that direction, then simulation studies using the methodology of S.C.O.P.E. (or S.C.O.P.E. itself) would be of particular assistance in gaining performance analyses.

Documentation

Documentation becomes an important aspect of a simulation

package such as S.C.O.P.E. since the structure of the model should be visible to the user.

S.C.O.P.E. is documented at three levels of detail according to the aspects of the model, with which the user wishes to acquaint himself. All the documentation is in natural language since diagramatic documentation, such as flowcharts, is gradually becoming extinct and since natural language is universally understood and easier to assimilate.

The first level of documentation is in the form of comments included in the program. These explain the bare essentials of each segment and its function. This level is geared towards an understanding of how the model works and its basic structure, and allows the user to gain knowledge which will assist him in deciding whether the basic structure contains enough detail for his interests or whether the model needs to be expanded in certain areas to suit his purposes.

The second level is included in the library and takes the form of a more detailed comment for each segment together with any relevant notes concerning the use of that segment. This level is geared towards assisting the user to select the modules he requires. Hence, if the user wishes to relocate core depending upon certain conditions governing the degree of fragmentation he will be able to select the segment which includes relocation rather than the one which does not.

The third level, also included as part of the library, is the most detailed level, which takes the form of a step-by-step account of the functions of each particular segment, explaining what each part of the code is doing. This level is provided for the user who wishes to make minor alterations to the segments or who wishes to reassure himself that the segment is a true reflection of part of his system.

Such levels of documentation are a necessary part of this methodology, if the user is to make full and easy use of the simulation package.

Input to S.C.O.P.E.

It has already been mentioned in Chapter 3, that one of the major disadvantages of simulation packages is the great deal of effort required to prepare a simulation run. The user often has to use specifically designed formal languages or graph representations to describe the system, the jobs which are to run on the system, the hardware configuration etc. Such an operation can take a great deal of time and hence, if a simulation package is to be easy to use, this preparation time must be reduced. The preparation time required to formally specify each job that comprises the simulated workload is of primary importance, since it is now recognized that a system's performance is depdent upon its workload. Therefore, in order to fully investigate a system, the simulation must be run using different workloads; it is in this situation that the preparation time becomes critical and a great burden to the user.

In order to overcome this problem, a stochastic method of input should be used. In this case, random number generators decide the workload which is to be run on the simulated system, within limits defined by the user. Hence, the chance of an input/output instruction occurring or the end of a job due to error or completion is defined by the user and the random number generator choses the next event based upon this information. The same principle applies to the types and sizes of jobs to be run on the system. In this way, the user's task is greatly simplified. In order to simulate a different job load he merely sets the "seed" of the random number generator to a different value so that a different sequence of events is followed. In order to simulate various unusual conditions, such as a large number of input/output bound jobs or very large jobs, he merely resets his limits to depict this. Using this method a great deal of effort and time can be saved where it is of major importance in defining different workloads.

The other information which must be supplied by the user is the hardware configuration he wishes to simulate. This consists of varying parameters within the model which represent the number of processors, size of memory etc., and is a relatively simple task.

Dynamic model

One of the objections to current simulation packages is that the package itself is not "available" to the user. He cannot manipulate it in any way and hence he is restricted in his options, to those already defined in the package.

The theory behind S.C.O.P.E. is to allow the user to adjust the model; in this way he will understand the model better and be reassured. Hence, the model is not static but is dynamic. If the user changes the model by adjusting/replacing/adding a segment and if he then includes it, together with its documentation, in the library, that user is promoting the ease with which the model can be used since another user may require a similar alteration.

It is also hoped that this facility will extend the life of the model, since future uses and alternatives, not yet envisaged, may be incorporated into the model's library.

This facility is also important in S.C.O.P.E.'s use as a design aid, since it must be able to include alternatives yet to be designed. If the model is to be of use in this situation it must be adjusted by a person who has a detailed knowledge of the system which is to be evaluated, i.e. the designer himself, or, as in the case of the Chief Programmer Team, the designers right-hand man, the back-up programmer.

"In many cases .... solutions ..... are not at all obvious or intuitive, so that it is necessary to try many different ideas. Hence, the importance of the flexibility and ready adjustability which a simulation model may be able to offer" (N6).

Output from the model

The ease of using a simulation package not only concerns developing the model and running it, but also concerns the ease with which the results can be understood or the type of output adjusted.

If the user can manipulate the model himself then obviously he can output certain factors which interest him. This is an extra gained from the dynamic nature of the model. However, the package must also include some standard output routines. S.C.O.P.E. includes output statements built into the skeleton structure, which, on completion of a simulation run give a tabular summary of various performance factors. This must be included in all simulation studies. However, the major point to be stressed, is that the output must be clear and easily read.

For completeness, the simulation model should also include a means of output for intermediate results obtained during the simulation run, in order that the user can see any patterns emerging.

4.6. Conclusion

"The current state of the art does not enable analysts to quickly interface an appropriate model to whatever data happens to be available".

This problem, outlined above by Boehm and Bell (B26), is one of the many, which, if the methodology described in this Chapter is employed in building simulation packages, should be overcome.

Morrison (M25) discusses the present trends in computer performance evaluation and says that certain changes are required. Those changes are:

1. Measurement technology needs to become more implementation indepdendent
2. Collection and display of measured data should be made more dynamic if to be effectively used
3. The means of understanding the performance consequences of changes in design, configuration workload, without the expense of implementing them and then measuring them, must be improved.

It would seem that the theories advocated in this study would fulfill all those necessary changes and would go further in establishing an improved state-of-the-art in the field of performance evaluation, especially if design trends continue in the same direction.

"There is a serious need, not only to develop the necessary models to cope with current system performance problems, but also to advance the current state-of-the-art for these techniques" (N6).

Chapter 5 IMPLEMENTATION

5.1. The stages of implementation

5.1.1. Introduction

In the previous chapter, it was specified that the simulation package should consist of a skeleton structure and a library of segments which could be added to or interchanged with segments which form part of the skeleton structure.

The skeleton structure must be implemented first, with the premise that it must be modular in order to reflect design methods and to facilitate the inclusion of segments from the library, once that has been developed.

Four steps are recommended in developing the skeleton structure:

1. Define the segments and their functions
2. Define the standard interfaces between the segments
3. Determine the logic of each segment
4. Prepare code for each segment

5.1.2. Define the segments and their functions

In defining the various segments within the skeleton structure, the developer must bear in mind that these modules should encompass all the basic operations which are found on a system, for instance, assigning jobs to processors, dealing with interrupts, assigning memory space to jobs etc.

Besides these modules representing the operating system functions, it is also necessary to include an initializing routine in the skeleton structure which will initialize variables, set timers to zero, and, in general, set up the state from which a simulation run will begin.

Another requirement is the inclusion of a master segment or the main body of the simulator, which is responsible for controlling the functioning and time-keeping of the simulator and co-ordinating the various segments representing operating system functions.

Also, since the simulation model is to be of a stochastic nature, such that the simulator itself determines the types of jobs and the various events within those jobs (within pre-defined limits), the simulator must include two segments to carry out these operations: one to deal with the types and sizes of the jobs and one to determine the events within those jobs.

Hence the simulation package has basically four components:

1. An initializing segment
2. Two segments to deal with the generation of the workload
3. Segments which reflect the basic operations of an operating system
4. A main body to co-ordinate all the segments named above.

The initializing segment and the segments generating the workload have self-defined functions which have been stated earlier.

The segments which compose the third component have yet to be explicitly defined. A study of operating systems shows that they can be divided into various categories of operations. These are:

1. input-output operations
2. processing management
3. interrupt management
4. memory management

In order to promote the ease of understanding and programming operating systems, these categories can be further subdivided into the basic operations which can be found in each category. This procedure should be adopted in the development of a simulation of this kind since it will promote readability and the ease with which the package can be used. (See Appendix 1 for the breakdown of these four categories adopted in S.C.O.P.E.). Each operation within the subdivision of the categories should have associated with it a segment in the skeleton structure, and it is in this way that the segments within the skeleton structure are defined.

The main body of the simulator should also be split into various parts according to the functions it is to perform. (See Appendix 2 for the breakdown of the main body of S.C.O.P.E.).

Once these operations have been carried out, stage 1 of the implementation is completed, since all of the components of the skeleton structure have been defined together with their functional specifications.

5.1.3. Define the standard interfaces between the segments

Dividing the basic operations of the skeleton structure into the four categories of input-output operations, processing management, interrupt management and memory management, and then subdividing these, implies that the segments within each category are closely related. It is upon these relationships that the first emphasis should be placed. The remaining standard interfaces between the segments have the effect of joining the categories and the other segments, (i.e. the main body, the job and event generating segments and the initializing segment) together to form a cohesive system, (See Appendix 3 for a description of the standard interfaces adopted in S.C.O.P.E.).

Once all procedure calls into and out of the various segments of the skeleton have been decided upon, stage 2 of the implementation process is completed. The implementer should now have a coherent system of mutually dependent system segments, which should, at this stage, reflect the basic structure of an operating system.

5.1.4. Determine the logic of each segment

The theory in the previous chapter, stated that the initial logic of each segment reflecting the operating system functions should be composed of an algorithm (or algorithms) which bring about the effect of that function, together with a time delay for that function. The task of the implementer then is to develop the logic of such an algorithm for each segment.

The best approach to this problem is to utilize the information already acquired during the implementation process. The segments have been defined and in order to determine their standard interfaces, an outline of their functions has been formulated. These functions must now be expanded into more detail.

A use of the techniques of stepwise refinement is indicated, since it is necessary to break down the function of a segment into its component parts until such a level of detail is reached that the coding of that segment can be carried out without difficult.

An example of such a breakdown with respect to a segment of S.C.O.P.E. may assist at this point and, for this purpose, consider the segment INSERT.

It's function is defined as: organise the queue of jobs ready to be processed: and it is only called in order to place an entry for a job on the queue, according to some scheduling strategy.

If it is assumed that the strategy to be implemented is the Round Robin Scheduling Strategy, the development of the segment INSERT should follow a similar course to that outlined below:

A. 1. Organise the queue of jobs ready to be processed.

B. 1. Add on delay associated with this action.

2. Place an entry on the ready queue, dependent upon the time when it will be ready to run. (Those ready to run first will be at head of list).

C. 1. Add delay on to the time of the processor, and on to a variable which will indicate the time spent in system routines (as opposed to time spent executing jobs).

2. If the head of the list is empty, put the entry at the head of the list.

3. If there are entries already on the list, check down the list until an entry is reached whose time is greater than the entry which is to be inserted, and insert before this one.

4. If no such entry is on the list, place entry at end of queue.

The specification of the logic now reached in this segment is at an acceptable level of detail and can be used to generate code for the segment.

One advantage of this approach is that it is particularly suitable for generating the three levels of documentation which were considered necessary for a simulation package of this type (see section 4.5.3.). The implementer, at this stage, has before him progressive levels of detail ranging from a functional definition to a logic specification. From these various levels of stepwise refinement, he can easily generate comments for the program or detailed specifications for the library.

A further advantage of this approach is that the logic of the segments is, as yet, implementation independent. That is, the implementer has no need to decide upon the language which will be used to code it, since it is not based upon any assumptions concerning this decision. The advantage is thus that a good specification is universally acceptable at this level.

5.1.5. Prepare code for each segment

Due to the approach taken in the earlier stages of implementation, the task of preparing code for each segment becomes less difficult. The logic has already been decided upon, and the task of the programmer thus consists of determining the language structures which will be necessary for coding this. For instance, in the example concerning the segment INSERT, (see previous section), the programmer will discern that he requires a list structure containing the job entries, and that each entry will include the name of the job and the time at which it will be ready to run.

Another aspect of this process of implementation becomes apparent at this stage. By studying the logic specification of the skeleton structure and thereby determining the language structures needed,

the programmer is assisted in his choice of programming language. Thus, if a language which he is considering does not contain, or has difficulties associated with, a certain type of data structure which he requires, that language should be classed as unsuitable for the project.

In preparing the code for a simulation package of this type, it is suggested that all variables be global. If this is the case, confusion as to the meaning of variables can be reduced, since the meanings of variables must, by their global nature, be standardised. If local variables are used there exists the possibility that two variables of the same name might occur, each having a different role in the program. The difference between them is not then distinguishable by name but by content only and it is from this type of situation that confusion arises. Since one of the primary aims of this type of package is that it should be easily understood, a standardisation by means of global variables is advocated. (See Appendix 4 for the programming language used in S.C.O.P.E.; Appendix 5, evaluates this language's suitability for simulation projects).

5.1.6. Conclusion

If the skeleton structure is implemented in the manner described above, it assists in various aspects of the simulation package, for example, documentation and modularity. The segments which will eventually form the library, i.e. the alternative operating system strategies, can be developed in a similar manner. The functional specifications of the alternative segments will be the same as those of their counterparts in the original skeleton; however, the stretegy and thus the logic and its stepwise refinement will be different. Hence, their development might proceed from step 3 in the implementation process. Additional segments, with no counterparts yet included in the skeleton structure, must be developed from step 1, by defining their functions.

Chapter 6 S.C.O.P.E. USER'S GUIDE

6.1. Introduction

The guidelines given in this chapter should assist any subsequent users of S.C.O.P.E. in both implementing his model and retaining the dynamic nature of the package. The package is to a large extent self-explanatory due to the documentation included in it (see Appendices 6, 7, 8); these guidelines can be seen as part of that documentation. As one of the main aims of this study is to provide a visible model and ease of use to a user, this user's guide is designed to include all the relevant information required to develop a model, yet, at the same time, is designed to be easily intelligible. A problem with many existing models or packages is that their manuals are often too complex to be easily understood by a prospective user, who is not familiar with them, for example, the GPSS III User's Manual (I1).

There are two basic problems which a user must overcome before the model can be run effectively:

1. How to develop a model to reflect the system it is to evaluate
2. How to run the model.

These problems are dealt with in the next two sections respectively.

6.2. How to develop a model

6.2.1. Skeleton selection

The first level of documentation, i.e. the comments within the listing of the skeleton will assist the user in understanding the structure of the skeleton. Appendices 2, 3 will also be of use in this process.

The skeleton structure of S.C.O.P.E. contains the segments of the model which are required for minimal operation, i.e. these segments must be included in the model in some form or the model will fail to run. The skeleton structure, therefore, should contain all the categories of segments given in Chapter 5. These segments, however, need not be the same as they are listed in the original skeleton given in Appendix 7. They can be altered or replaced should a user's requirements necessitate this. As an example of this, consider the segment INIT, which is responsible for initialising the state from which the simulation run begins. The user may wish to begin his simulation with a fragmented memory, or certain jobs blocked etc., and he can alter INIT to reflect this. (See section 6.2.5.). The only requirement is that the segement is included in some form or other.

One major factor which has been found to affect the form of the skeleton structure is the type of memory access/management scheme the user may wish to include. The question of whether the memory in the system is divided into segments or not affects the skeleton segments INIT, GAPLIST, SPACEINCORE and NEWJOB. The original skeleton structure does not provide the facility for a divided memory and since it is necessary to alter all the above segments in order to do this, both skeleton structures have been included in order to assist the user. These are given in Appendices 7 &12 respectively. The documentation for the skeleton structure reflecting a segmented memory (Appendix 12) is the same as the original except for the segments INIT, GAPLIST, SPACEINCORE, and NEWJOB which are included in the library.

6.2.2. Altering the skeleton structure

There are two ways in which the user can alter the details of the skeleton. The first method is to replace segments in the skeleton with alternatives from the library of segments; the second is to develop a segment for himself or alter the details of an already existing segment. (In either of the latter two cases the user will be adding a new version of the segment to the library).

In order to assist the user in the first method the library of segments is listed together with documentation for each segment in Appendix 8. The documentation index (see Appendix 6) will be important to the user in determining whether the library contains a suitable replacement segment and its position in the library. The documentation index constitutes the second level of documentation in the package. If a suitable segment is found in the index the user simply swaps that segment with the one of the same name in the skeleton structure. (Notes concerning the use of the replacement segment are included in the detailed documentation where necessary). Thus, in this case the user's task is a relatively simple one.

However, if a suitable replacement segment is not found in the library or one is found which, although similar, requires slight alteration, the task of the user becomes more difficult. The onus is upon him to develop an alternative segment which can replace the original segment. Also if the dynamic nature of the package is to be retained, the user must also include his alternative into the library together with its detailed documentation, and a full comment for the documentation index. In order to assist the user in altering a segment, or merely discerning if the segment does what he wishes it to do, a list of process variables has been compiled (see Appendix 10) which explains the meanings of those variables simulating system processes. Any new variables which he declares must also be placed onto the appropriate list i.e. they must be classed

under the heading of specification, process or measuring variables, and explained. (See Appendices 9,10,11). In this way the library will be augmented and the probability that a future user will have to develop his own segment will be decreased.

An example may indicate the user's role in this process. The example assumes that the user wishes to allocate jobs to processors on a priority basis (rather than the round-robin strategy assumed in the original skeleton), using the type of the job as a priority value.

From an inspection of the original skeleton structure he sees that the segment INSERT is responsible for the ordering of the entries in the ready queue. His next step is to check the documentation index to determine whether an alternative segment INSERT has already been developed which orders the queue in this way. Assuming that such a replacement segment is not found, the user returns to the original segment and its documentation, where he discovers that the queue is ordered by means of the TIME field in the JOBENTRY structure. The user then realises that a new field is required in the structure to hold the jobs priority value. The priority value is to be the job type value, which is recorded in array JOB, and thus no difficulties should arise in transferring that value to the new field which the structure is to have. The new field is then declared as part of the JOBENTRY structure and is placed in the process variable list explaining its use. The algorithm for the new segment can now be developed:

A. Place an entry onto the ready queue noting the time when it will be ready to run and add on a delay for such a task.

B. 1. Add on the delay.

2. Check down the list and place an entry for the job on the list. The entry contains the job's name, the time when it will be ready to run, and the job's priority. The placement of the entry on the list is determined by priority.

C. 1. Add on delay to SYSTIME, OSTIME and processor time.

2. If head of the list is null, the new entry is placed at the head of the list.

3. Otherwise search down the list until the priority of the new entry is less than the priority of an element in the list and place entry on the list before this element.

4. If no element on the list has a priority greater than the new entry, place the new entry at the end of the list.

5. Each entry on the list contains job name (PR [I]) job time (PRTIME [I]) and the job priority (JOB [PR [I], 4]), which is, in effect, the job type).

The coding for this segment can now be developed and when the segment works correctly a listing of it must be added to the library together with detailed documentation. An explanatory comment must also be supplied to the documentation index.

This alternative segment INSERT has, in fact, been developed in this manner and now forms part of the library. (See INSERT (2)). From this example, the process of enlargement of the library can be discerned.

6.2.3. Addition of detail

The addition of detail to the model can be done by either making the skeleton segments more detailed, in which case the process becomes the same as that for altering the skeleton, described in the last section, or by developing new segments which are included into the model and are called by a segment from the skeleton structure thus far determined. In this case the user must still follow the process of enlargement to the library and the variable lists, in the interests of subsequent users. It is envisaged that these extensions to the skeleton structure will form a more detailed simulation of the basic operations already covered in some respect by the skeleton. For example, instead of using IODEL i.e. the average time for a channel to input/output a piece of information from/to secondary storage, the user may wish to replace it with a segment simulating disk seek times etc., such that each input/output request has its own individual delay.

Some additional segments such as this have been developed and are documented in the library. A good example is the segment RELOC which was not incorporated into the skeleton. Since users may wish to simulate a relocatable partitioned memory management scheme, RELOC was developed to carry out the relocation operation; it is called from NEWJOB (3) in its attempts to bring in a job if fragmentation is at too high a level.

The filestore associated with S.C.O.P.E. contains a file for the original skeleton structure, SCOPSKEL, a file for the segmented memory skeleton structure, SCOPSEG, and a number of files, named after the code names given in the library, which each contain one segment from the library. For instance, file NEWJOB 2 contains the second version of the segment NEWJOB. (The library code name being NEWJOB (2)):

Whenever the user wishes to incorporate an alternative segment from the library into the skeleton structure, he must do so by editing the file holding the skeleton structure. The operation is a simple one which consists of copying the file until the beginning of the segment he wishes to swap with the alternative, and then merging it with the file holding the new segment. He must then delete the original segment but copy the rest of the skeleton. The same operation should be carried out with new segments. The user must first create a file which contains his new segment with a suitable library code name and then merge that into the skeleton structure at the appropriate place. The only difference between the two operations is that, in the latter, there is no corresponding segment to delete.

In carrying out these operations, the user must:

1. not alter the original skeleton and hence it is suggested that the user copies it into a new file before beginning any alterations.
2. retain any new segments he develops in suitably named files for the benefit of subsequent users.
3. carry out a final check that the variables used in his segments are declared at the head of the skeleton program.

6.2.4. Skeleton/Model Specification

Once the structure of the model has been decided upon, it is necessary for the user to specify the configuration of his system: for instance, the number of processors, size of memory, timing delays for software operations etc. To facilitate this task a list of the relevant specification variables in S.C.O.P.E. and their meanings has been compiled and is given in Appendix 9. They can be categorised into two sets; the first dealing with configuration specification, the second with timing specifications. The values assigned to them must be of the type and in the units outlined in Appendix 9. (See section 6.3. for further details).

It is also necessary to alter the value distributions provided to segments JOBSIZGEN and INSTRGEN such that the workload characteristics of the system are depicted accurately.

JOBSIZGEN and INSTRGEN are related in that the type of job generated in JOBSIZGEN has an influence on the events generated for that job in INSTRGEN.

The first step for the user is to define how many types of jobs he requires and the percentages of each type occurring in the workload. For example, his "typical" workload may consist of small jobs 50% of which are processor bound and 50% input/output bound. The mill times used by his jobs may vary from 10 seconds to 70 seconds. If he takes three average classes of mill times as being 20, 40 and 60 seconds (10-30, 31-50, 51-70 respectively) each class having both types of job described above, he will require six job types (3 x 2). The user must then define the percentage of each of these types occurring in his workload. The input/output bound-processor bound properties are simulated by INSTRGEN which utilizes the job type value associated with each job.

A similar procedure is carried out in the second part of JOBSIZGEN, which is responsible for generating the size of the jobs. The user defines classes of sizes and the percentages of those sizes of jobs occurring.

INSTRGEN is in the form of an "if" statement each entry being equivalent to a job type, and containing the probability of certain events which might occur within the execution of a job. In the skeleton structure, there are three basic events - time-slice expired, input/output request, termination of the job (due to error or completion). Further events can be placed into each entry if required. A simple formula can be used to calculate the values which represent the probabilities of a time slice expiring, an i/o request occurring or a job terminating:-

$$\text{prob} \text{ —— } \frac{x}{y}$$

where prob = either SLICEPROB, IOPROB, TERMPROB.

x = the whole range of the random numbers generated (should be large enough to give positive integer values in all cases)

y = the time between the occurrences of the event in question (in millisecs).

Same examples will illustrate the use of the formula. Assuming the range of random numbers to be 0-100,000:-

The value for a job terminating for jobs requiring 30 seconds mill time will be

$$\text{TERMPROB} = \frac{100{,}000}{30 * 1000}$$

(A termination will occur every 30 seconds on that type of job).

$$= 3$$

The value for an i/o request occurring if input/output facilities are required 20 times per second on the system, will be:

$$\text{IOPROB} = \frac{100{,}000}{1/20 * 1000}$$

(every 1/20 second input/output facilities are required by that type of job).

$$= 2{,}000$$

The value for a timeslice expiry if time-slices are 100 millisecs, will be:

$$\text{SLICEPROB} = \frac{100{,}000}{100}$$

$$= 1{,}000$$

Although the events will not occur in these exact proportions due to the effect of the random numbers, the sane principle applies to the actual system where many fluctuations in demand occur. Also, since the sequences of random numbeps generated can be altered by placing differing values in SEED i and SEED 2. (see appendix 9), differing workloads (with the same characteristics) can be simulated. Hence the same system can be evaluated on differing workloads or differing systems can be compared by their performance on the same workload.

6.2.5. The Initial state

One difficulty which can be found in simulation projects is that all the runs of the model begin from a set initial state. This factor can influence the results gained and thus, various initial states should be possible in a model. This facility is provided in S.C.O.P.E. by the segment INIT, which can be manipulated by the user so that he can specify the state from which his run is to begin. In this segment, jobs are placed in memory, on the ready queue or can be blocked. Interrups can be set to occur, channels can be set to be free or busy until a certain time. From these individual factors, the user is allowed a multiplicity of initial states. He can begin his run with a fragmented memory, only one channel free, several jobs blocked awaiting input/output etc.

6.2.6. Output from a model

Included in the skeleton are standard output statements which tabulate the information collected throughout the duration of the simulation. This information should satisfy many user's requirements. The user can, however, build his own measuring devices or variables into the

model, if he requires any additional information. To assist the user in formulating more measuring capabilities, a list of those variables concerned with this function is given in Appendix 11. This list should first be checked, if the user does require more measurements, since not all the information collected in the skeleton is output, and the variables he requires may already exist. The output statements which form the standard output can be found at the end of the MAIN BODY.

There is also another mechanism built into the loop of the MAIN BODY, in which the master clock is checked against a sample variable, SAMPLEVAR. If they are the same, the mechanism collects various pieces of information for use in the standard output. A further time delay is then added to SAMPLEVAR. In this way the user can gain sample data every x seconds/millisecs.

Another similar, but optional, mechanism relies on a similar variable SAMPLEPRINT. This is used not only to gather information periodically but also to output it. In this way the state of the system can be viewed throughout the run. Although this facility is not included in the skeleton it has been used in a study of the George 3 operating system. (See Chapter 7). Two other segments have been developed which can be called from within the SAMPLEPRINT mechanism. They are JQSIZEPROBE and IOQPROBE, which count and output the sizes of the critical region queue and input/output queue respectively.

6.3. How to run a model

The S.C.O.P.E. Package is at present in use on the ICL 1904S running under the George 3 Operating System at the University of Aston, and uses the standard macros there for running jobs both in the foreground and background queues.

Assuming that the user wishes to run the skeleton structure as it stands or has formulated a model and placed it in a file, he is now faced with the problem of running it. For this purpose S.C.O.P.E. includes an interactive program which asks the user to give the specifications which he has already formulated (see section 6.2.4.). The system specifications he must provide are given in the specification variable list. These are ordered and numbered and the interactive program asks for the serial number of any variables the user wishes to alter. The variables are given the standard values which are given in the list and which are stored in a file called INFO. If the user does not wish to alter any of the standard values, the program copies the values from INFO into a file called OUTPUT, which will form the user's specification file for that particular run. If the user does wish to alter some values he must give the serial number of the first variable in the list whose standard value he does not wish to retain. The program will then ask him to give the value he requires in place of the standard value. Any variables he does not request are automatically assigned the standard values. Thus, in formulating his system specifications the user should write down in the order given in the list his values and note any which differ from the standard values.

The interactive program then asks the user to give his workload specifications, commencing with the number of job types he requires and the percentage of each type on his system. (The maximum number of job types allowed is 20). The user must then give the number of job sizes he requires, and then supply the percentage of each size occurring on his

system. For example 5% of jobs might be 10K core image size. (The maximum number of job sizes is 8). Next the program requests the number of events the user wishes to simulate. (The maximum number is 8). Three are supplied by the skeleton i.e. time-slice expired, i/o request, job termination as these are basic to any system and their probability values which must then be supplied must be given in that order (or, if the user alters INSTRGEN, in the order they are programmed in INSTRGEN). The probability values given can be calculated by the formula explained in section 6.2.2. The user will require a set of probability values for each job type, since each job type will have different characteristics. Hence an array of these values must be supplied by the user, e.g.

	Event 1	Event 2	Event 3
job type 1	1000	4000	4
job type 2	2000	2000	2

The values the user supplies for all his specifications must be of the type required (real or integer) but should not be accompanied by any other signs or symbols, e.g. K, millisecs, %, etc. should not be included. Thus if 5% of jobs are 10K, the user should punch 5 and, when prompted, 10.

- 5
- 10

Once all these values have been input as necessary the users specification file is completed. He is then asked to supply the name of the file which holds his model and the simulation run is carried out automatically henceforth.

In order to initiate the interactive program the user must first ensure that he has 10K core space and then punch in "SCOPE" from the terminal.

Any new specification variables the user has declared must be put onto the end of the specification variable list together with its serial number and its standard value; the standard value must then be

placed into the file INFO in the order indicated by its serial number and a read and print statement placed in INIT (similar to those already existing in INIT). Any specification variables which the user does not use in his model may be left to be automatically assigned the standard value as they will not be referred to in his model.

A listing of the interactive program is included for interested readers (see Appendix 13).

Chapter 7

A SIMULATION OF THE GEORGE 3 OPERATING SYSTEM USING S.C.O.P.E.

7.1. Introduction

In order to discover whether the simulation package S.C.O.P.E., and its associated methodology, is applicable to large systems containing many facilities, it is necessary to carry out a simulation experiment based upon such a system. For this purpose, the George 3 operating system, which is currently in operation at the University of Aston, was chosen. This choice was influenced by the availability of the systems performance statistics and by the fact that it is a large and complex system which would test S.C.O.P.E. in many ways.

7.2. A description of the George 3 operating system

The first stage in simulating an existing system is to formulate a brief description of both the operating system and the hardware configuration, paying particular attention to the systems idiosynchrasies. This is of assistance when developing the model, since it shows the developer what details need to be added to the skeleton, which existing segments are suitable and the type of segments which need to be developed when no counterpart exists in the library.

The hardware configuration on which George 3 operates consists of a single processor, the ICL 1904S, three input/output channels between the processor and secondary storage, an exchangeable disk store and a high-speed drum.

The operating system is highly transient. It is divided into chapters and only those chapters required by jobs running at a particular time are in memory, except for certain chapters, which, because they are required very frequently, reside in memory at all times. This means that if a job requires a different chapter from its previous one, a search must be made to see if the chapter is already resident in memory. If not, it must be brought in from secondary storage, overwriting a chapter not in use.

A formula is provided by the system manual (I6), which calculates, from the number of jobs started (and tentatively started) the minimum amount of memory which will be taken up by George chapters.

The system uses the relocatable partioned memory scheme, relocating memory when the amount of fragmentation exceeds the amount of memory used by core-images.

The scheduling strategy of the system has three levels:

1. The high-level scheduler accepts jobs from the background job queue and determines whether one can be tentatively started, and, if so, which one. The jobs which are tentatively started are passed onto the low-level scheduler. The high level scheduler is called when a job is placed in the background queue, when a job terminates and when a user wishes to run a job from a terminal.

2. The low-level scheduler decides which of the tentatively started jobs are to be fully started i.e. rolled into memory and given processor time. The low-level scheduler is called when a job terminates.

3. The executive scheduler multiprograms the jobs in memory on a round robin basis, deciding which job should be allocated the processor next.

The operating system has two queues of jobs; the background and the foreground queues. Jobs in the background queue must wait to be started at the discretion of the high-level scheduler. However, jobs in the foreground queue, which are initiated from a M.O.P. (Multiple On-line Programming) terminal are immediately placed onto the low-level scheduler list, if the high-level scheduler decides it can be tentatively started.

Another aspect of the system which must be considered is the "red tape headings" associated with the jobs on the system. This is the information required by George 3 in order to process the jobs, i.e. their state, characteristics, the buffers they have opened etc. A formula is given in the system manual (I6) for calculating the amount of memory which red tape headings will require based on the number of jobs started. This factor and the fact that the operating system is itself transient

means that the amount of memory available for core images of jobs is always fluctuating. Also, in order to function properly George 3 requires a free pool of store of 5K and 3K for essential functions.

The above description contains the main factors which typify the George 3 operating system and it is these factors which must be added to the skeleton structure of S.C.O.P.E. as extra details.

7.3. The development of the model

The development of the George 3 operating system model described here should provide a useful example of using the guidelines given in the previous chapter. A separate section is given for each of the various steps given in the User's Guide.

7.3.1. Skeleton selection

The original skeleton structure of S.C.O.P.E. is a suitable starting point for this model since, like George 3, it assumes an undivided memory and can easily be adapted to provide the right hardware configuration. However, due to the complexity of the George 3 system several additions of detail to the software are required. On comparing the skeleton structure and the description of George 3 given in the previous section several differences can be discerned; these are given below:

1. Transient nature of operating system
   - swapping from chapter to chapter
   - bringing in a chapter from secondary storage
2. Relocation
3. The extra level of scheduling - the high-level scheduler
   (executive scheduler = segments INSERT and REALLOC
   low-level scheduler is included in NEWJOB)
4. The foreground queue of jobs and the distinction between fully started and tentatively started jobs
5. The red tape headings, the free pool and the memory required for essential functions

7.3.2. Altering the skeleton structure

The next step in the process of developing a model is to alter the skeleton structure to suit the system which is to be simulated. This might entail either adding details to an existing segment, rewriting a segment or replacing a segment with one from the library.

In the case of this model only three segments of the skeleton structure required alteration, all of which required the addition of details.

The first section which needed more detail was the memory management section, since the possibility of relocation of memory, the transient nature of the operating system and the red tape headings all affected this section. In particular the segment NEWJOB was affected in that it had to include the possibility of relocating memory if a new job could not be fitted into memory. In order to include this the segment must first try to find space for the job, as occurs in the original segment, but if that fails, a check must be made to see whether the fragmentation level is greater than the job load, in which case relocation must take place. Since relocation is not included in the skeleton, a call to a new segment must be made at this point. (The new segments are dealt with in the next section). The segment NEWJOB is also affected by the extra level of scheduling since a job cannot be fully started unless the high level scheduler has already tentatively started a job. Hence, a check for this has to be included also and if a job is fully started the number of tentatively started jobs must be decremented. Another factor of the George 3 system is introduced here, because the number of tentatively and fully started jobs affect the amount of memory required for George chapters and red tape headings, which in turn affects the amount of memory left for core images. Thus, a call to another new routine which simulates this fluctuation in memory allocation is necessary after passing through the segment NEWJOB.

These modifications were made to the original segment NEWJOB and the George 3 version placed in the library (see NEWJOB (3)).

The second segment which required alteration was INSTRGEN, since the three basic events i.e. time-slice expired, input/output request and termination included in the skeleton were not sufficient. George 3 required six events:

1. Time-slice expired
2. Input/output request
3. Termination
4. Chapter change
5. Chapter transfer
6. A job being placed in the background well or initiated from a terminal

These events were added to the original segment and the revised segment now forms part of the library (see INSTRGEN (2)).

The final segment requiring modification was INIT. In order to set up the initial state of the simulation run, INIT attempts to fill the available memory (FREECORE) with core images. However, FREECORE fluctuates according to the number of tentatively and fully started jobs. The additions of details to this segment therefore were to set an initial number of tentatively started jobs and load jobs into memory until the space required for chapters and red tape headings prevents any more being loaded. In order to do this two new segments were needed to calculate the quota of memory required by both. These new segments are named CHAPQUOTA and REDTAPEQUOTA respectively and are both called from INIT for the purpose outlined above. The new segment INIT can be found in the library (see INIT (3)).

Once these alterations had been made to the skeleton structure, the next step was to develop the new segments, which have been referred to, as they provide the idiosyncrasies which typify the George 3 system.

7.3.3. Additional segments

The five new segments required in addition to the model thus far developed are included in the library and so they will not be described in detail here. However, an outline of their functions and their relationships to the rest of the model is provided.

The five segments are: (to give them their library names)

1. RELOC (1)
2. HLS (1)
3. CHAPQUOTA (1)
4. REDTAPEQUOTA (1)
5. FREECOREADJ (1)

N.B. Since all the above are new segments all their library subscripts are (1)

RELOC (1)

The segment RELOC relocates the core images in memory by moving jobs down to the bottom of memory, thereby collecting space at the top. It requires the inclusion of another specification variable RELOCDEL i.e. the delay associated with relocation.

Its relationship to the rest of the model is straightforward in that it is only called from the segment NEWJOB to which control returns when the segment RELOC has been executed.

HLS (1)

The segment HLS simulates the behaviour of the high-level scheduler and thus determines whether or not to tentatively start a job. Its decision is based on an algorithm devised by the system programmers at the University of Aston, which briefly is that a limit of 200K is set as a parameter to the high-level scheduler. All the MAX-SIZE specifications (provided by the users on system job cards) of all the tentatively started jobs are totalled. If this total is less than 200K a job is tentatively started. For the purposes of simplicity in this

model a max-size specification of 40K was assumed, since that is the specification of the majority of jobs on the University of Aston system. The segment HLS requires a new specification variable HLSDEL.

Its relationship to the rest of the model is that it is called only from INSTRGEN when a job terminates or when a job is placed in the background queue or initiated from a M.O.P. terminal.

CHAPQUOTA (1)

The segment CHAPQUOTA calculates the minimum amount of memory required by George chapters. No specification variables are required for this segment.

Its relationship to the rest of the model is two-fold in that it is called by INIT as described earlier and is also used in the new segment FREECOREADJ.

REDTAPEQUOTA (1)

The segment REDTAPEQUOTA calculates the amount of memory which will be occupied by red tape headings and other George facilities i.e. the freepool and the pool for essential functions. No specification variables are required for this segment.

The relationship of REDTAPEQUOTA to the rest of the model is the same as that of CHAPQUOTA.

FREECOREADJ (1)

The segment FREECOREADJ uses the values calculated by CHAPQUOTA and REDTAPEQUOTA and adjusts the amount of memory available for core images accordingly. No specification variables are required for this segment.

FREECOREADJ is called from NEWJOB only as described earlier and calls no other segments itself.

Finally, once all the new segments had been coded and notes made of any specification variables they required, all the new measuring and process variables created by the addition of these segments to the model were placed onto the appropriate lists (see Appendices 10 and 11) and declared at the head of the model.

7.3.4. Skeleton/model specification

Additional segments to the skeleton will often present new specification variables to the model. The extra specifications required by the George 3 system model are given below:

1. Delay for swapping from chapter to chapter
2. Delay for transfering a chapter
3. Relocation delay
4. High-level scheduler delay
5. The free pool of memory required by George
6. The memory required for essential functions

These, therefore, were placed onto the specification variable list, giving their names, types, meanings and standard values. The latter were also placed onto the file INFO, which holds the standard values, in the order given in the specification variable list. Also segment INIT was also augmented with extra read and print statements to take account of these specifications. (N.B. Since the User's Guide is up-to-date at the stage at which this research ended, the above specifications are already included).

The next step was to prepare the performance specification values for the George 3 system, which were required by the model. The values for the specifications were drawn from various sources: the ICL George 3 operations manuals (I6) (I5), discussions with the systems manager and the performance statistics collected at the installation.

The configuration specifications such as memory size, number of processors and channels etc. were easily collected. The values for

FREEPOOL and ESSENTFUNCT were given in the manual (16). The delays for input/output and for rolling in/out jobs were calculated from the average disk seek times, wait times etc. However, the delays for the operating system software functions could only be estimated due to the fact that no information regarding this type of specification was available. The workload specifications were collected from software monitoring files produced by the system, indicating the sizes of jobs, mill times used by jobs, how often chapter changes occurred etc. From these the job load characteristics and event probabilities were formulated.

7.3.5. Initial State

The initial state of the system simulation was kept basically unchanged from that of the skeleton structure, the only change being due to the flexibility of the amount of memory available for core images.

7.3.6. The Output from the Model

Besides the standard output mechanisms of the skeleton, the George 3 model included the SAMPLEPRINT mechanism as described in section 6.2.6., which was executed every 50 seconds simulation time and which provided the percentages of time spent by the processor in operating system routines, in execution, or idle state since the last sample and also the amount of memory currently allocated to core images, system functions, or currently unused. It also provides the number of core images, tentatively started jobs and jobs in the background queue at that particular time. This additional output mechanism created new measuring variables which were placed on the measuring variable list.

7.4. RESULTS AND CONCLUSIONS

Since the operating system specifications had to be estimated due to the unobtainability of any data which could be used in the George 3 system model, the results could be said to be a true reflection of the George 3 system. However, the results obtained are similar to those gained from the software monitors of George 3. A listing of the George 3 system model together with the results gained is given in Appendix 14. (N.B. The times are given in milliseconds).

The successful formulation of this model, did, in fact, prove that a complex system can be evaluated using S.C.O.P.E. methodology which was the main aim of the exercise. However, it also pointed out certain deficiences in the simulation package.

The first of these concerns the formula given in section 6.2.4. for calculating the probability values for the event generating routine INSTRGEN. This formula is based on the assumption that the segment INSTRGEN is entered every time-step, i.e. every millisecond, of simulated time. Although this would seem to occur when running the skeleton structure, it is not always the case. The frequency of entrance to the segment INSTRGEN is determined by the amount of time spent in the operating system segments of the simulator, since the more time the processor spends in operating system routines, the less time it spends executing jobs and hence segment INSTRGEN is entered less frequently. Due to the large amount of time the processor spent in the operating system on the George 3 Model, INSTRGEN was entered less and hence the events occurred consistently less often than the probabilities would have determined. This problem was overcome by adjusting the probabilities, but no formula could be discerned for estimating the probabilities under those circumstances.

Another difficulty with S.C.O.P.E. is that because the simulator runs with average values, the results obtained are average. Hence the diverse fluctuations which occur over several minutes on a complex system do not appear in as great a proportion on the simulated system.

Another result of the George 3 experiment was the time scale factor of the simulation model, which was:

$$\frac{\text{duration of simulation run}}{\text{duration of run simulated}} = \frac{1}{1}$$

This points to another difficulty with S.C.O.P.E. since a simulated model would normally be expected to run faster than the actual system. However, this could be overcome by stepping the model every second rather than every millisecond, although the results would be grosser.

The final difficulty with S.C.O.P.E. which the George 3 experiment showed, was that file handling work such as editing etc. which forms a large part of the work carried out on terminals at the University of Aston installation could not be included in the model due to the absence of these facilities from the package. It may, however, be possible to build such facilities into the package although this was not attempted as part of this study.

Overall, however, excluding the difficulties outlined above, the model is a good representation of the George 3 system and the development of the model was relatively simple due to the methodology inherent in S.C.O.P.E. Another consideration on this point is that

the task of developing such a model at a later stage, when S.C.O.P.E.'s library has been augmented, should prove to be even simpler.

Chapter 8

THE RESULTS OF SOME EXPERIMENTAL WORK USING S.C.O.P.E.

8.1. Introduction

Several questions were raised in section 2.5. of this study concerning the hardware configurations and software proposals for multiprocessor systems. In order to find some general conclusions to some of the proposals put forward, it was necessary to evaluate them using the skeleton structure of S.C.O.P.E., for, if any system details were included in the simulation experiments, the results would apply only to systems containing those iodiosynchrasies. However, since the skeleton structure includes only the minimum of detail, its results cannot be seen as being examples of actual system performance, but merely indications of general conclusions.

8.2. Experiment 1: Ratio of Memory to Processors

8.2.1. Questions:

What should the ratio of memory to processors be?
Does this ratio vary according to the memory access scheme being used on the system?

8.2.2. Method

In order to answer the first of the questions raised as part of this experiment, the original skeleton structure was run with various hardware configurations; namely, 1, 2 and 3 processors and amounts of memory available for core images (i.e. FREECORE) varying from 30K to 120K. A constant number of channels, 4, was assumed for these experiments. The results of the simulation runs were compiled into graphs, the axes being the size of memory available to core images and the (average) percentage of idle time of the processor(s).

In order to answer the second question, the segmented memory access skeleton was used in a similar manner for 1 and 2 processors and the graphs compiled from this experiment were compared with those of the previous experiment.

8.2.3. Results and Conclusions

The results from the experiments outlined above are given in Figures 8a, b, c, d and 9a, b, c. (Figures 8 referring to the first experiment and Figures 9 to the second).

When assessing Figure 8a, in which various simulations with one processor were carried out, it can be seen that the percentage of idle time of the processor does not decrease much above 80K available memory. However, below 80K a definite increase in idle time can be seen, which becomes more pronounced as the amount of memory available for core images decreases. The conclusion to be drawn from this is that between 60K and 80K would seem to be the ideal amount of memory for one processor. (This conclusion is reflected by the George 3 system at the University of Aston, where, as has already been mentioned (Appendix 14), the amount of memory available for coreimages is often set at 80K).

A similar pattern emerges in the graph for two processors (Figure 8b) except that the level of acceptability in this case is approximately 100K and in order to have 95% utilization of processor time 120K is required instead of the 80K in the previous graph.

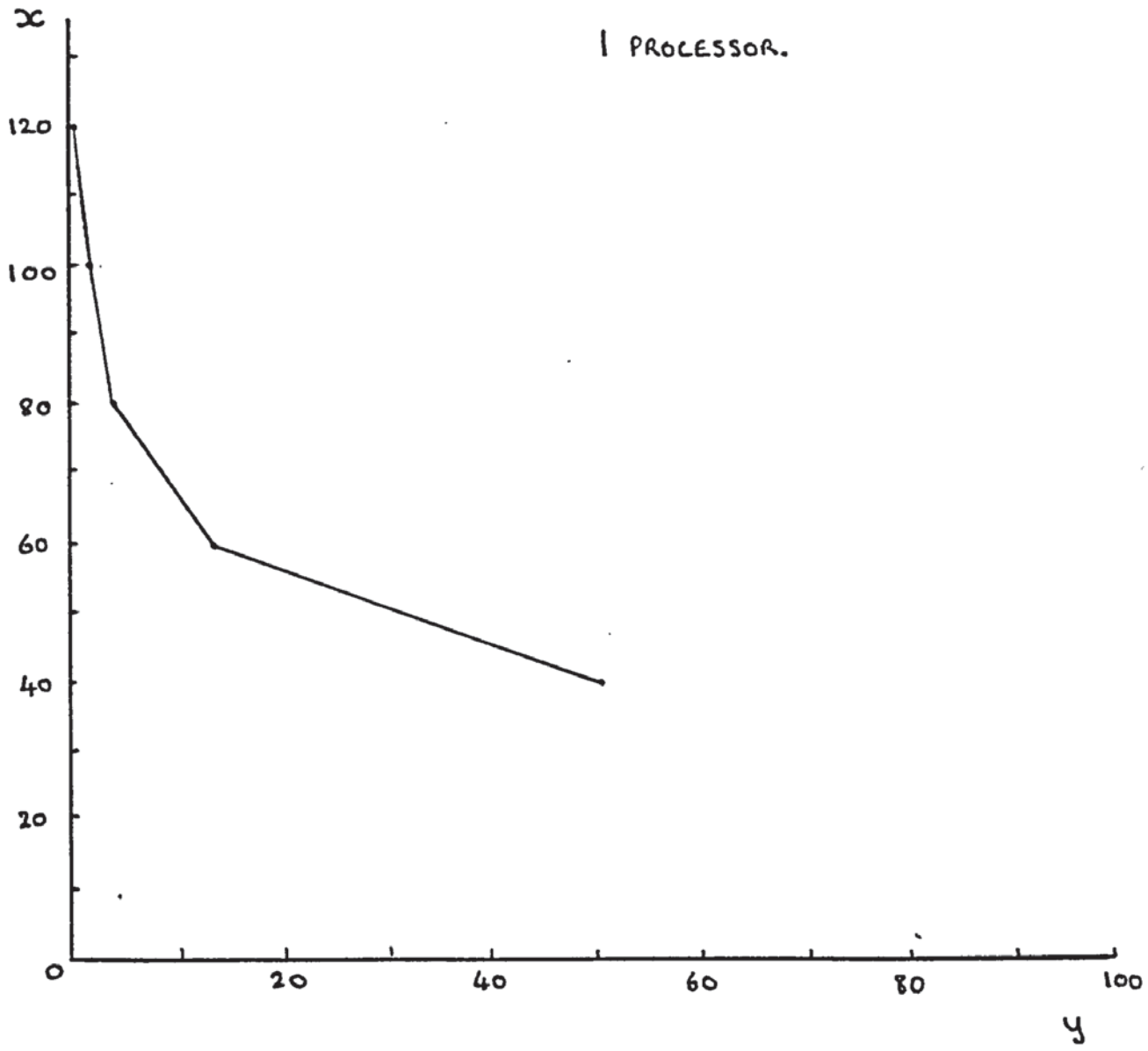
With three processors on the system (Figure 8c) a level of 120K produces almost the same percentage of idle time as 100K. This seems to contradict the trend which seems to be emerging, since from this graph 100K would seem to be the best ratio for three processors as well as for two. Also the graph does not, as yet, indicate an amount of memory which would allow 95% utilization of processing power. In order to probe into this further, more simulation runs were made using up to 220K i.e. almost twice as much memory, but even at this level the processor was idle 13.8% of its time. Because of this, it was assumed that some other factor was causing the restriction in processing power; namely, the number of input/output channels on the system. This was indicated by the fact that the four channels were, on average, busy for

more than 90% of the time with 120K and for 99% of the time with 220K. Hence, the question of the ratio of channels to processors was raised which is dealt with in experiment 3. However, for the purposes of this experiment, a graph for a three processor system using 5 channels was compiled (Figure 8d). From this it can be seen that an acceptable percentage of processor utilization is reached at 120K - 140K and that for 95% utilization 180K is required.

The results of the experiments made with segmented memory access are given in Figures 9a, b, c. If these are compared with Figures 8a, b, c, d, it is apparent that the type of memory access scheme does have an effect on the ratio of memory to processors. In fact, the segmented memory access scheme wasted so much space that at a level of 120K with 2 processors the regular pattern had still not emerged. The regular pattern did emerge if memory was extended as can be seen from Figure 9c.

FIGURE 8a:

RATIO OF MEMORY TO PROCESSORS (UNSEGMENTED MEMORY)

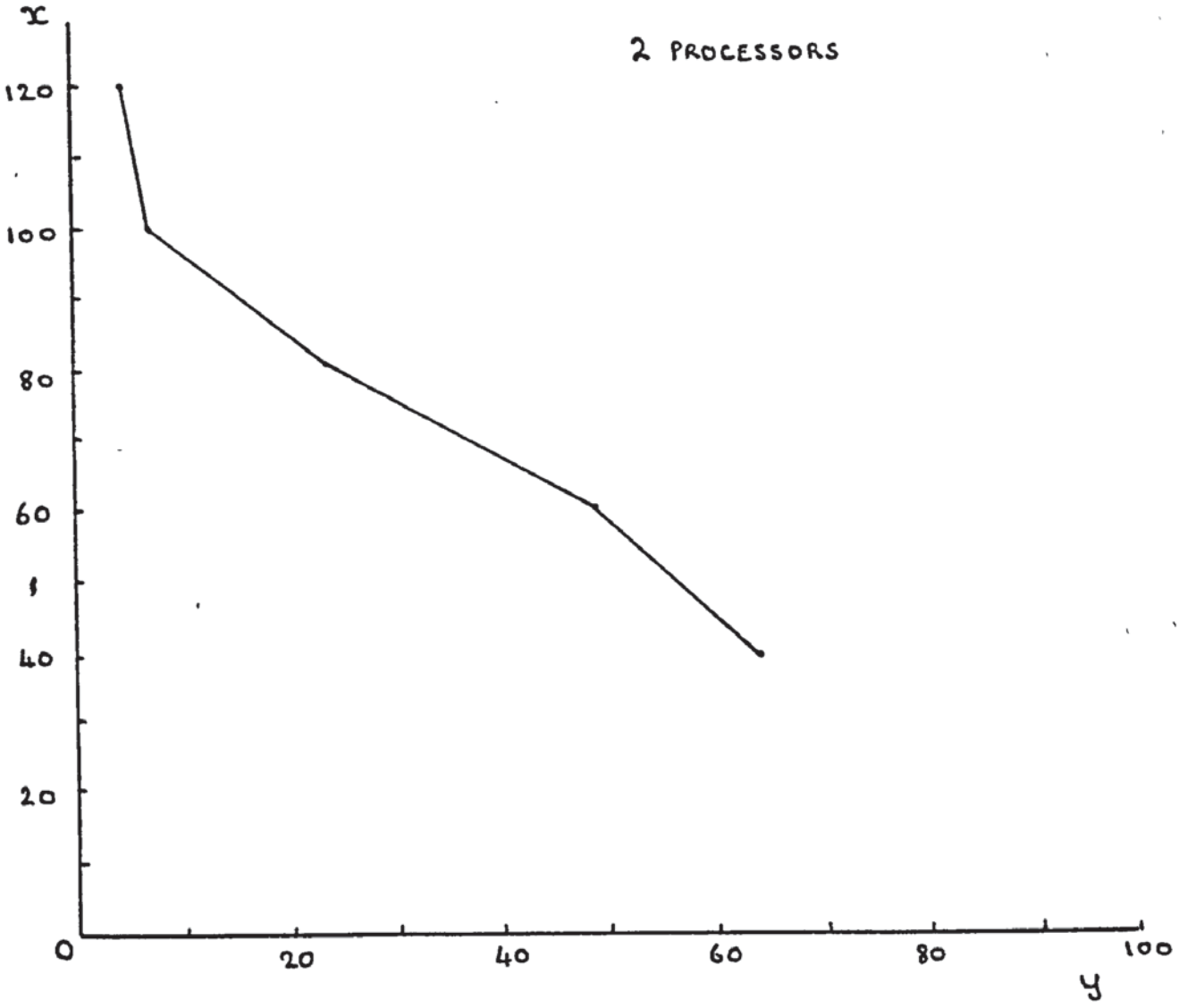


x AXIS – MEMORY SIZE (K WORDS)

y AXIS – PERCENTAGE IDLE TIME

FIGURE 8b:

RATIO OF MEMORY TO PROCESSORS (UNSEGMENTED MEMORY)

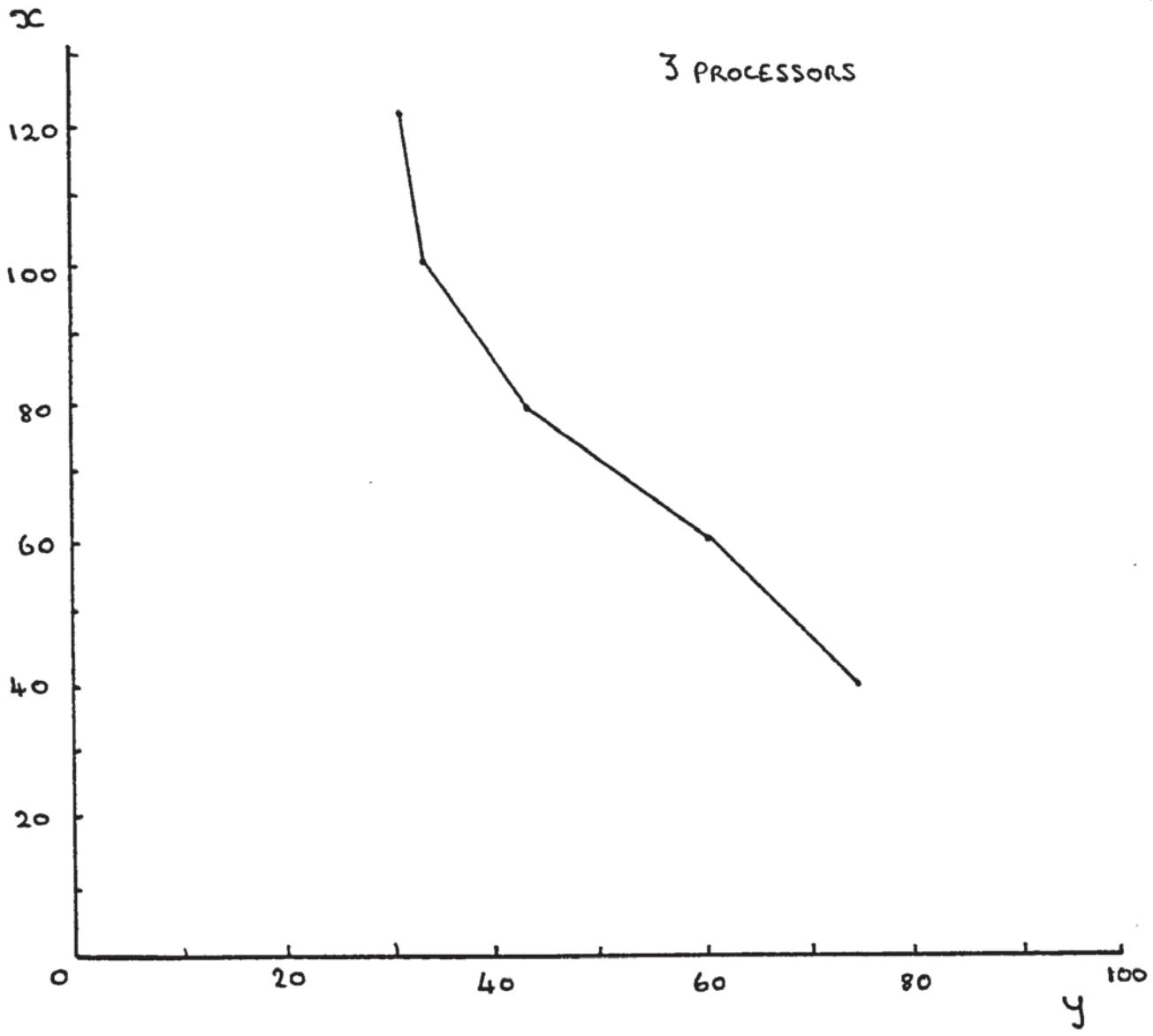


x AXIS — MEMORY SIZE (K WORDS)

y AXIS — PERCENTAGE IDLE TIME

FIGURE 8c:

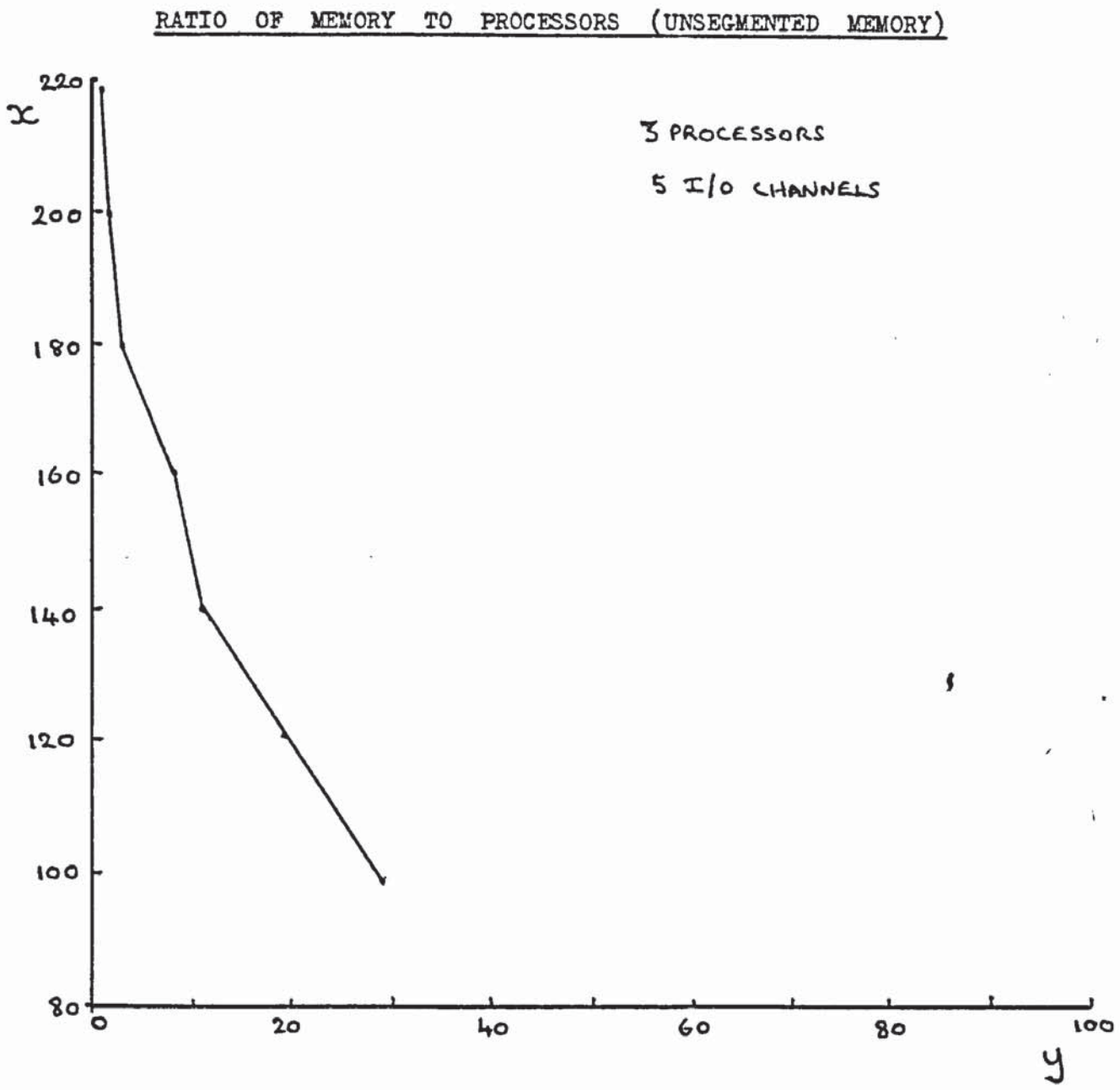
RATIO OF MEMORY TO PROCESSORS (UNSEGMENTED MEMORY)



x AXIS – MEMORY SIZE (K WORDS)

y AXIS – PERCENTAGE IDLE TIME

FIGURE 8d:

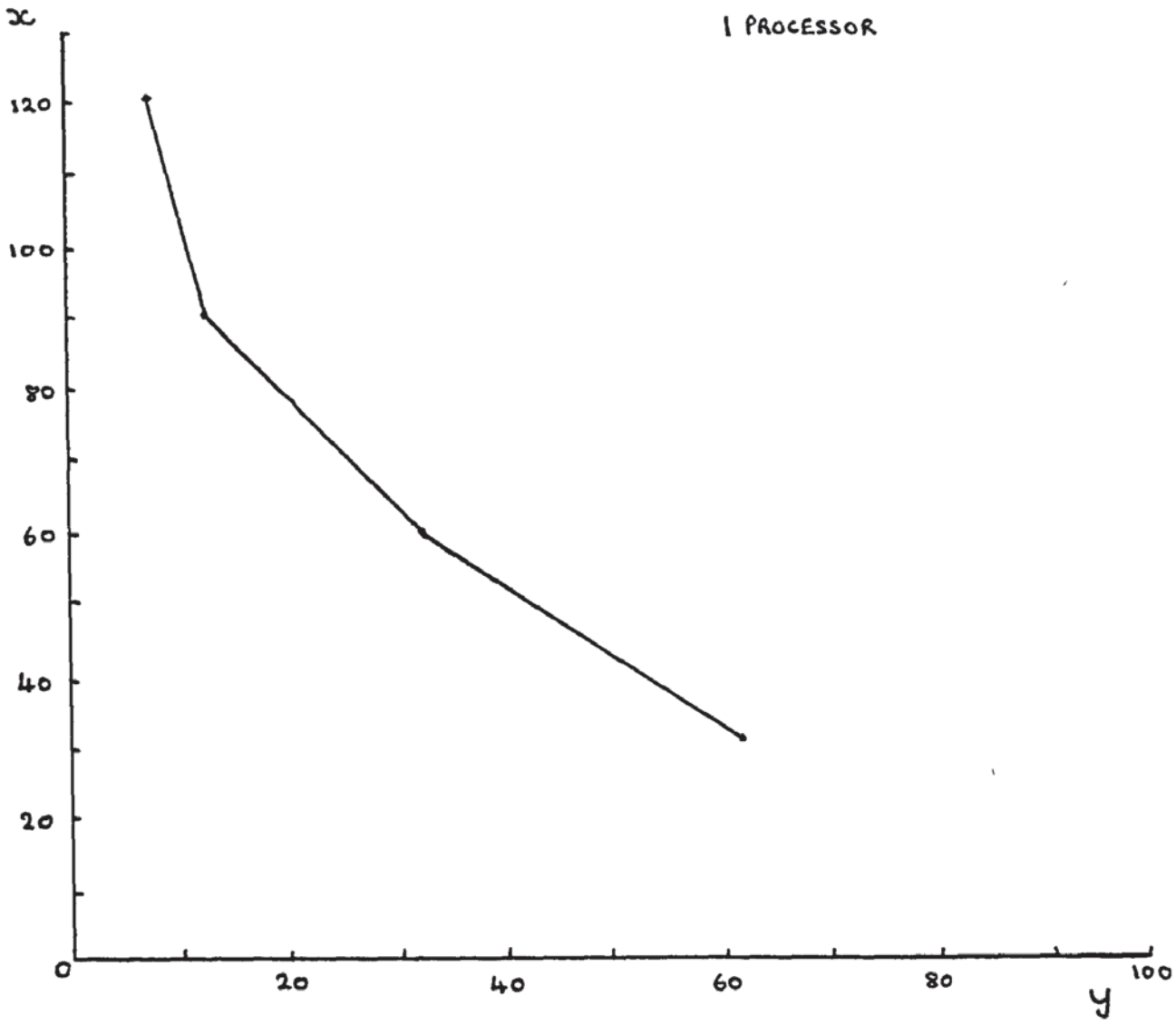


x AXIS – MEMORY SIZE (K WORDS)

y AXIS – PERCENTAGE IDLE TIME.

FIGURE 9a:

RATIO OF MEMORY TO PROCESSORS (SEGMENTED MEMORY)

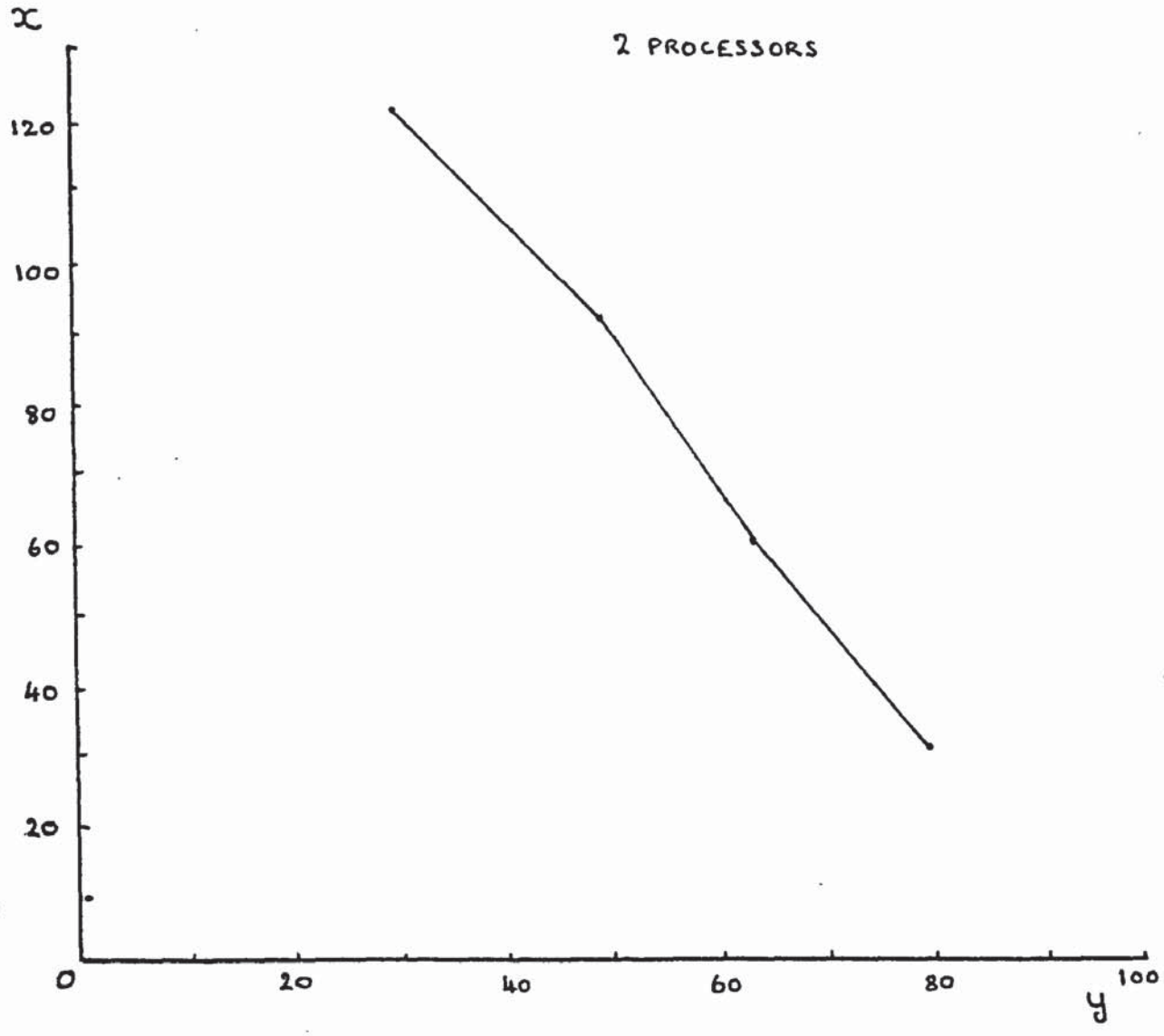


x AXIS – MEMORY SIZE (K WORDS)

y AXIS – PERCENTAGE IDLE TIME.

FIGURE 9b:

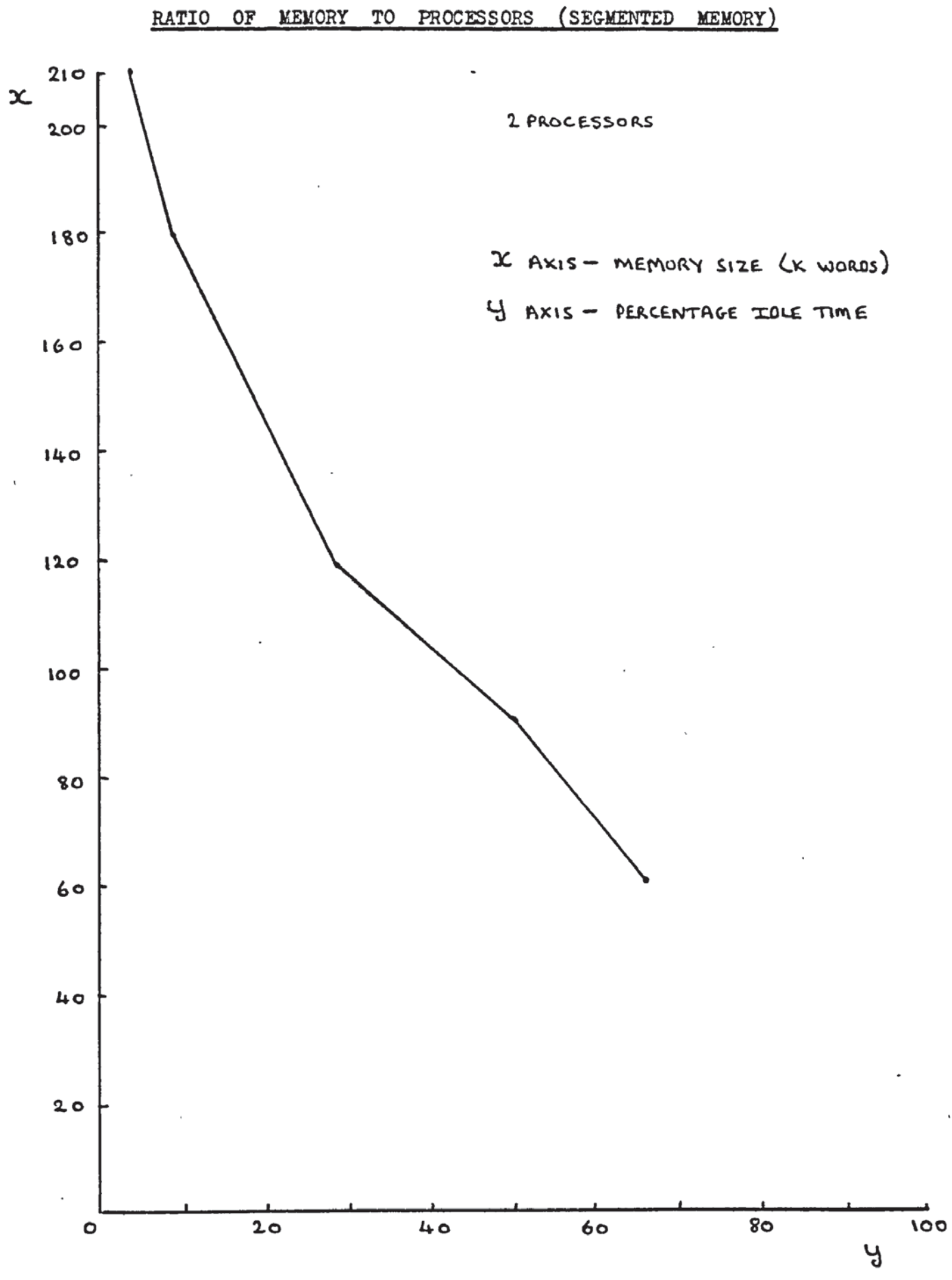
RATIO OF MEMORY TO PROCESSORS (SEGMENTED MEMORY)



x AXIS — MEMORY SIZE (K WORDS)

y AXIS — PERCENTAGE IDLE TIME.

FIGURE 9c:


RATIO OF MEMORY TO PROCESSORS (SEGMENTED MEMORY)
x
210
200
180
160
140
120
100
80
60
40
20
0
20
40
60
80
100
y
2 PROCESSORS
x AXIS – MEMORY SIZE (K WORDS)
y AXIS – PERCENTAGE IDLE TIME

8.3. Experiment 2: Ratio of the number of coreimages to processors

8.3.1. Questions:

How many core images are required on the system to give almost full utilization of processing power?

8.3.2. Method

The last experiment proved that the ratio of memory to processors varies with the memory access scheme used, which makes it difficult to predict the utilization of processing power from such a ratio. A more constant guideline is required. Hence, this experiment is designed to test another ratio which might provide a more constant guideline.

In order to answer the question given above, a utilization of processing power of approximately 95% was assumed as being a good utilization. When this level of utilization was reached using the original skeleton structure, the number of coreimages on the system were noted. In order to test if this gives a more constant measure of processor utilization, the same number of core images were used in simulation experiments using segmented memory access and the results compared.

8.3.3. Results and Conclusions

The following tables give the number of core images on the system at approximately a 95% level of processor utilization on the original skeleton and the utilization of the processing power with the same number of core images on the segmented memory skeleton respectively.

Table 2a: Ratio of coreimages to processors (unsegmented memory)

No. of processors	No. of coreimages	% utilization
1	4 - 5	94.8
2	6 - 7	93.8
3	9 -11	94.4

Table 2b: Ratio of coreimages to processors (segmented memory)

No. of processors	No. of coreimages	% utilization
1	4 - 5	99.7%
2	6 - 7	94.8%
3	9 -10	92.4%

If Table 2a is compared with Table 2b it can be seen that a great similarity in processor utilization exists on each system, if the number of coreimages is the same. Hence, it can be concluded that the ratio of core images to processors can provide a useful guideline to processor utilization. The tables also provide a direct answer to the question, - how many core images are required on the system to give almost full utilization of processing power for various numbers of processors.

8.4. Experiment 3: Ratio of channels to processors

8.4.1. Question: What should the ratio of channels to processors be?

8.4.2. Method

In order to test the assumption made in experiment 1 that the large rise in memory required to give good utilization of processing power for three processors was due to the restraining effect of the number of input/output channels present on the system, it was necessary to repeat those previous simulations with extra channels. Also, if the number of channels can restrict computing power at one end of the scale, it might waste it at the other end, hence experiments were conducted keeping a set system configuration but varying the number of channels, and processors and size of memory (since this proved an important factor in experiment 1).

8.4.3. Results and Conclusions

Table 3 shows a summary of the results gained in this experiment. The table shows the optimum number of channels i.e. the lowest number of channels for that configuration. The general maxim was that if another channel were added to that configuration processor utilization would not increase by more than 5%. The general observation that was noted was that the more memory for core images and/or the more processors there are on the system, the more channels are required.

Another aspect of this experiment was that the addition of an extra processor above an efficient combination can often degrade overall performance rather than increase it.

TABLE 3: Ratio of channels to processors

No. of processors	Size of memory available for coreimages	No. of channels
3	120 K	5
3	100 K	4
3	80 K	3
3	60 K	2
3	40 K	1
2	120 K	4
2	100 K	3
2	80 K	3
2	60 K	2
2	40 K	1
1	120 K	2
1	100 K	2
1	80 K	2
1	60 K	1
1	40 K	1

8.5. Experiment 4 — The effect of an extra input/output channel

8.5.1. Question:

By how much does the addition of an extra input/output channel speed up system throughput?

8.5.2. Method

The method used for this experiment was the same as for the previous experiment in which all the permutations of the number of processors and channels and size of available memory were tested. For this reason, the output from the previous simulation runs were used, with the main emphasis on the effect on throughput rather than processor utilization, and on only those systems in which the number of channels seemed to be restricting throughput.

8.5.3. Results and Conclusions

It was found that the gain in throughput by the addition of an extra channel could not be formulated in any constant form. Gains of up to 50% were recorded but no pattern emerged.

8.6. Experiment 5 Scheduling Strategies

8.6.1. Question:

Which scheduling strategy will perform most effectively in a multiprocessor system?

8.6.2. Method

A constant system configuration was kept except for the fact that four different executive scheduling strategies were implemented. The four strategies used were:

1. round robin
2. priority scheduling based on JOBTYPE
3. priority scheduling giving preference to input/output bound jobs on the system
4. priority scheduling giving preference to processor bound jobs on the system.

8.6.3. Results and Conclusions

Figures 10a, b, c, d show the results of this experiment. The figures show that hardly any difference in processor or channel behaviour ensued due to the scheduling strategy chosen over the extent of the simulation runs. This is probably due to the fact that the jobs already present on the system are scheduled by the executive scheduler, thus they are all executed at some time or another. Hence, although a particular job's turn around time may be faster, the effect of the whole workload on the overall system is almost constant. It would seem, therefore, that the scheduler which determines which type of jobs which are to enter the system determines the effect on the system rather than the executive scheduler.

FIGURE 10a:

THE EFFECTS OF DIFFERENT SCHEDULING STRATEGIES:-

ROUND ROBIN

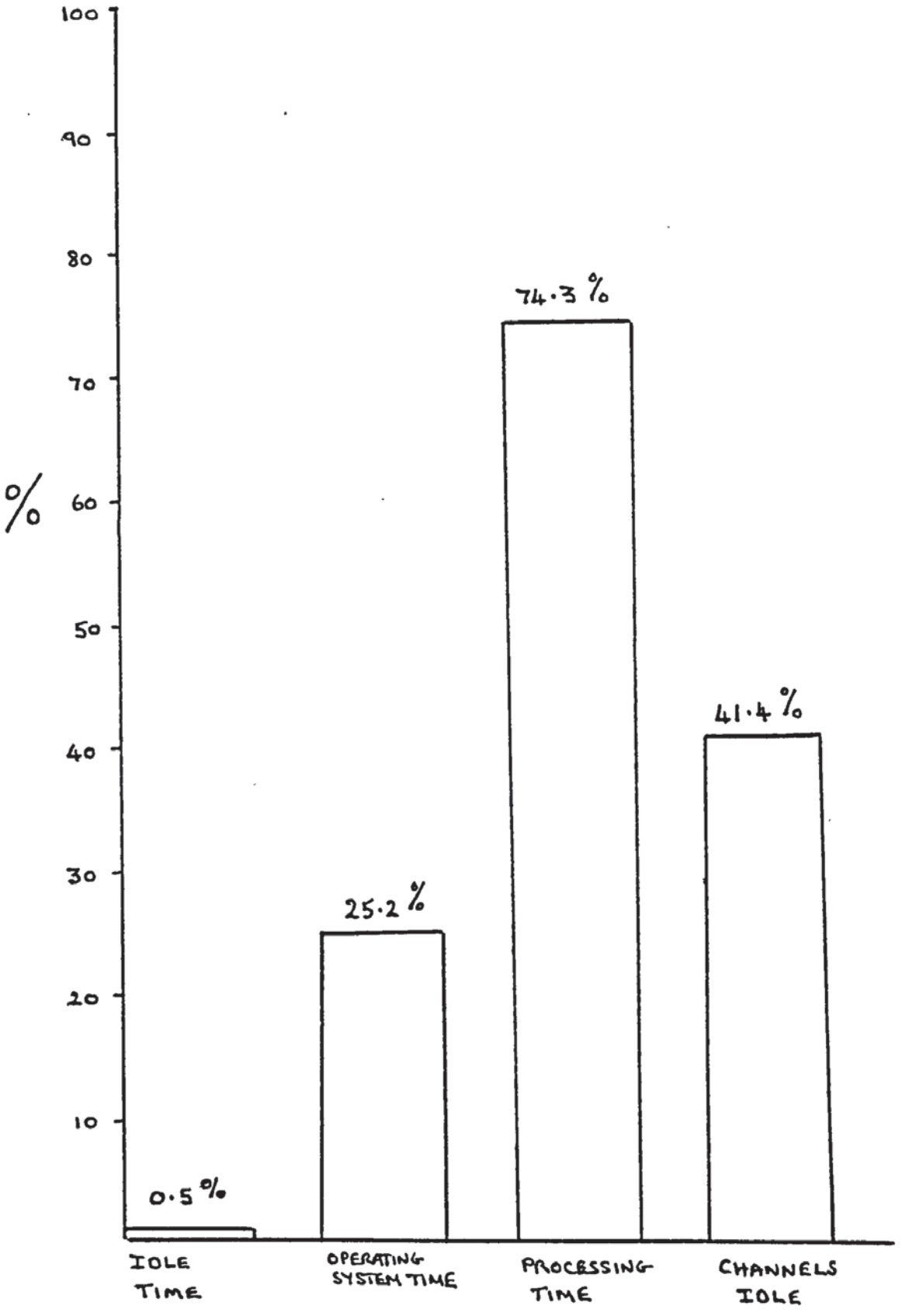


FIGURE 10b:

THE EFFECTS OF DIFFERENT SCHEDULING STRATEGIES:-

PRIORITY ON SHORT JOBS

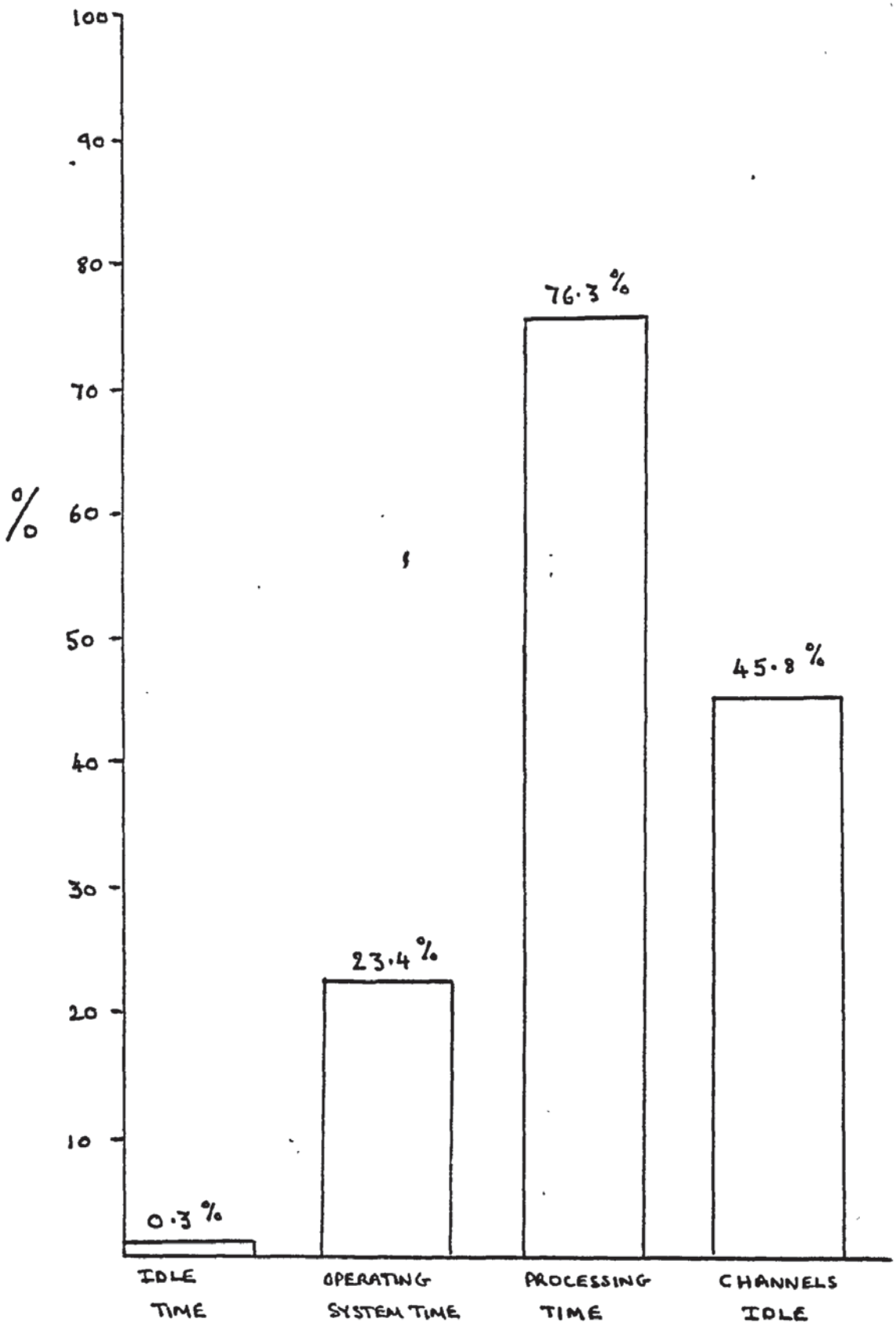


FIGURE 10c:

THE EFFECTS OF DIFFERENT SCHEDULING STRATEGIES:-

PRIORITY ON I/O BOUND JOBS

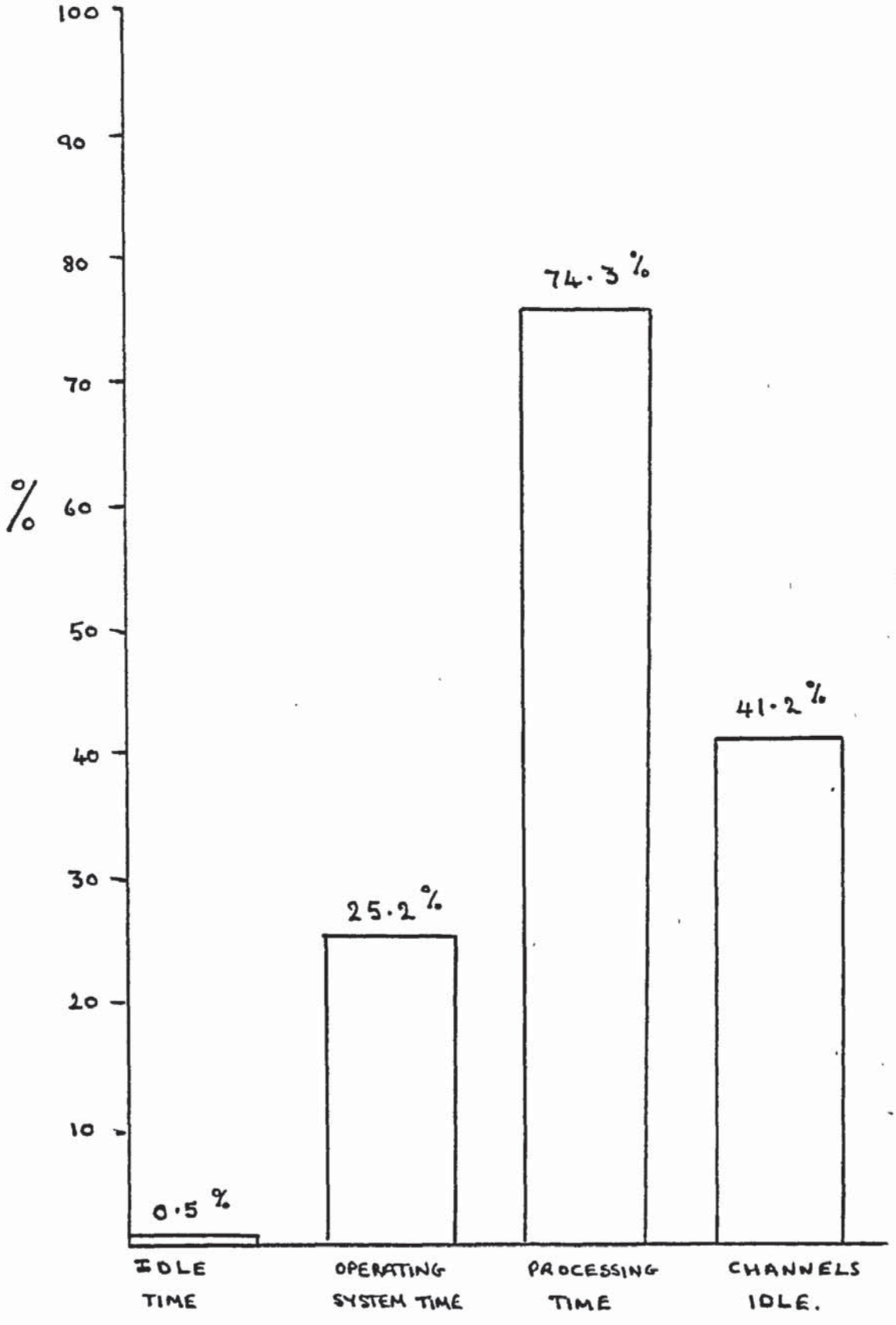
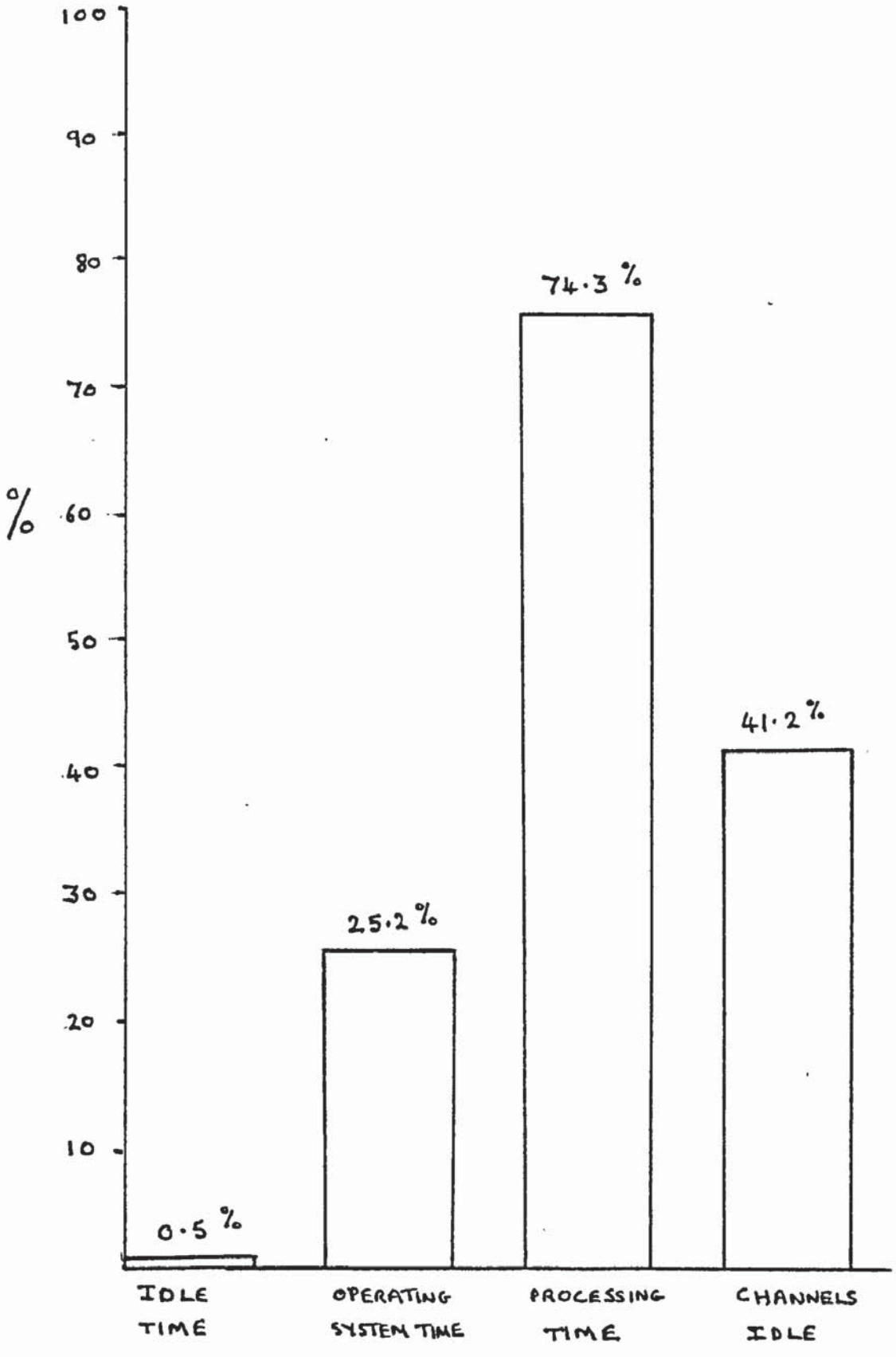


FIGURE 10d:

THE EFFECTS OF DIFFERENT SCHEDULING STRATEGIES:-

PRIORITY ON PROCESSOR BOUND JOBS



8.7. Experiment 6 Interrupt Algorithms

8.7.1. Question: Which processor should be interrupted?

8.7.2. Method

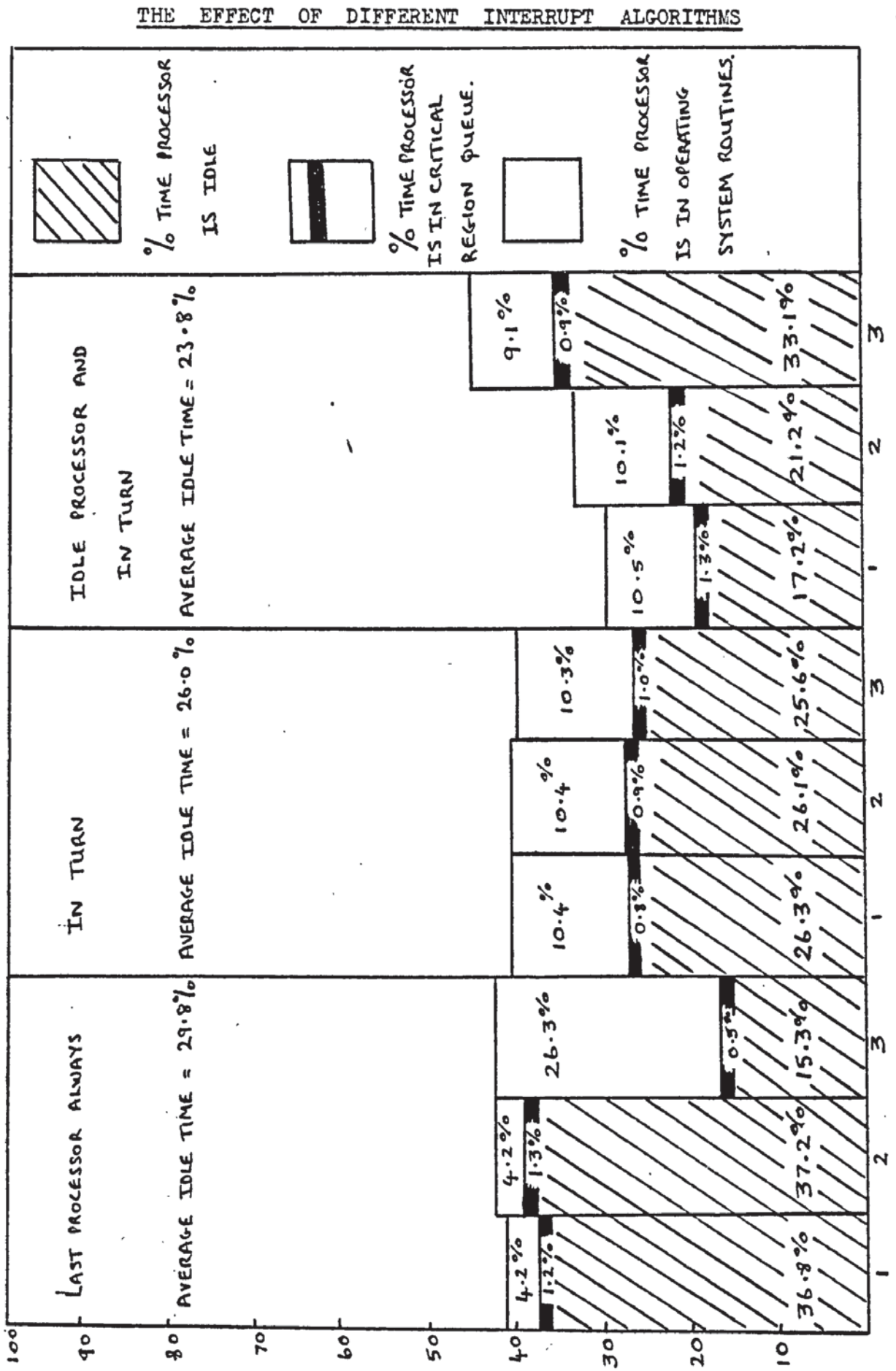
A constant system configuration with three processors was simulated using three different interrupt algorithms, and the results compared. The three algorithms tested were:

1. The same processor is interrupted always
2. Each processor in turn is interrupted
3. An idle processor is interrupted: or, if no idle processor, they are interrupted in turn

8.7.3. Results and Conclusions

The block diagram shown in Figure 11, shows that algorithm 3 is the most effective algorithm, since it produces a better overall processor utilization and provides a good balance of time spent by each processor in system routines. The latter point also applies to algorithm 2. With algorithm 1 the burden of system is placed on one processor. This has the effect of leaving the others idle unnecessarily. It is assumed that this effect would grow worse if more processors were on the system using this algorithm.

FIGURE 11:


THE EFFECT OF DIFFERENT INTERRUPT ALGORITHMS
% TIME PROCESSOR IS IDLE
% TIME PROCESSOR IS IN CRITICAL REGION QUEUE.
% TIME PROCESSOR IS IN OPERATING SYSTEM ROUTINES.
LAST PROCESSOR ALWAYS
AVERAGE IDLE TIME = 29·8%
IN TURN
AVERAGE IDLE TIME = 26·0 %
IDLE PROCESSOR AND IN TURN
AVERAGE IDLE TIME = 23·8%
% TIME
100
90
80
70
60
50
40
30
20
10
36·8%
4·2%
1·2%
37·2%
4·2%
1·3%
26·3%
15·3%
0·5%
26·3%
10·4%
0·3%
26·1%
10·4%
0·9%
25·6%
10·3%
1·0%
17·2%
10·5%
1·3%
21·2%
10·1%
1·2%
33·1%
9·1%
0·9%
1
2
3
NUMBER OF PROCESSORS.

8.8. Experiment 7 — The number of processors on the system

8.8.1. Questions:

Does the continual addition of processors to a system eventually become ineffective?

8.8.2. Method

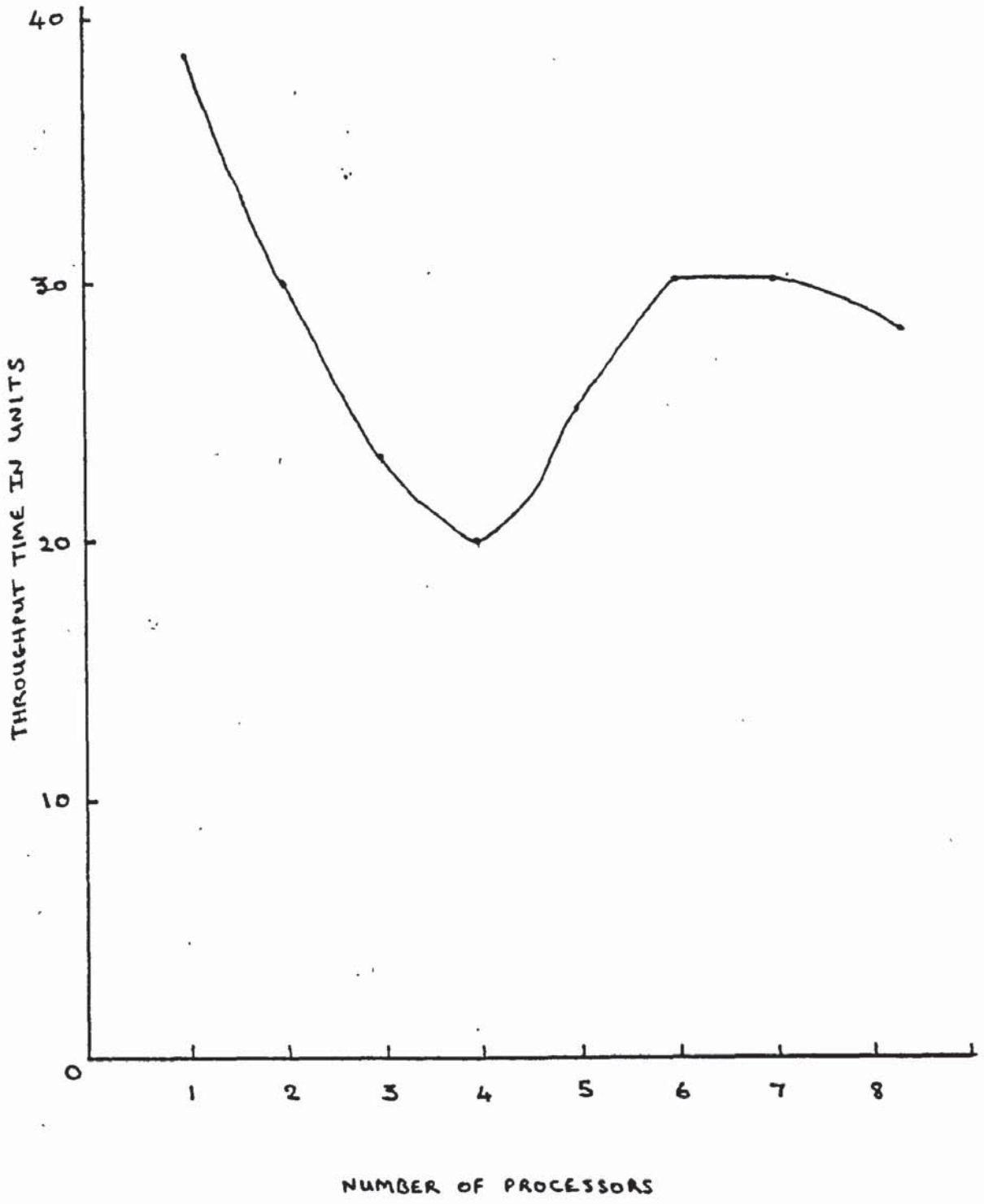
A constant system configuration was simulated with an incremented number of processors on each run, and a graph of the results compiled with axes throughput time and the number of processors.

8.8.3. Results

Figure 12 shows how throughput times were affected by the continual addition of extra processors. It was found that throughput time could be almost halved merely by the addition of processors, but that at a certain point the restraining effect of other factors on the system nullified the effect of more potential processing power. In fact, it can be seen that at this point the throughput time for the system degrades. This is probably due in part to the contention of the processors around the critical region, but a detailed analysis of this effect has not been carried out.

FIGURE 12:

THE RATIO OF PROCESSORS TO THROUGHPUT



Chapter 9 CONCLUSION

9.1. Research Aims Achieved

The main aim of this study was to develop a methodology of computer performance evaluation which would be of practical use in the design, comparison and extension of systems. Such a methodology was developed, and, based upon this, a simulation package was built. The simulation package should be of use in the three areas of performance evaluation given above, since:-

1. its skeleton structure can be made more detailed as the process of design continues and its building block characteristics reflect current design trends.

2. its stochastic method of input allows the user to run a model with different workloads easily or to run two models, which are to be compared, with a workload of the same characteristics.

3. a model which has been developed using this package can easily be extended, due to its building block nature, to include possible new additions or alterations to the existing system.

The methodology described in this study also provides a cost-effective evaluation tool which is easy to use, because, as the package develops and expands, less work is required of the user.

Finally, the simulation package has been designed to cover a large class of systems and can evaluate both single and multiprocessor systems and their appropriate operating system functions.

Hence, all of the research aims of this study have been achieved to some extent.

9.2. Suggestions for further research

One of the main drawbacks in using the simulation package developed in this study is the difficulty of obtaining performance figures for operating system functions. Many manufacturers release their figures concerning their various pieces of hardware, but seemingly no monitoring of the software functions is carried out, or if such monitoring is made, the findings are not published. This may be due to the difficulty in

expressing the performance of a software routine, since any number of different paths exist through the routine, some of which will be complicated and hence take more time, and some of which will deal with simpler situations.

It is surprising to find that this defficiency is not deliberated more in papers concerned with evaluation models and due to the lack of any discussion on this subject, researchers in this field remain unaware of the problem until they are faced with obtaining such performance figures. It is therefore suggested that a study concerned with this problem be undertaken as research. The study suggested could consist of developing some method of measuring software routines and designing some format for expressing the measurements taken. Then a collection of the performance figures for such routines could be prepared for the benefit of researchers in computer system evaluation models.

Another suggestion for further research is concerned directly with the simulation package developed in this study, i.e. the development of new and alternative segments for the library, to encompass the many differing characteristics of systems. Also due to lack of time the experiments carried out (see Chapter 8) give only indications of the answers to the questions posed and leave many theories and proposals to be tested, and these could be used as a basis for further research.

Appendix 1

THE BREAKDOWN OF OPERATING SYSTEM FUNCTIONS ADOPTED IN S.C.O.P.E.

Input/output operations

Segment name	Segment function
Rollin Endjob Inout Iolist	1. Rolling jobs into memory 2. Rolling jobs out of memory when they have completed or been stopped because of errors 3. Deals with input/output requests from jobs 4. Places an entry for any of the above operations onto a queue for a channel
Notes - since the main body is responsible for the timing of the operations, it will determine when a channel is free to take another entry from the queue created by Iolist	

Processing management

Segment name	Segment function
Insert Reallec Jqlist	1. Organise the queue of jobs ready to be processed 2. Allocation of jobs to processors 3. Management of the critical region around the above operations performed on the job queue
Notes - the main body is responsible for organising the flow of processors through the critical region	

Interrupt management

Segment name	Segment function
Intruplist Interrupt Intrupcheck	1. Organising a time-ordered list of interrupts which occur 2. Dealing with various interrupts and their effects 3. Masking interrupts if more than one needs to be carried out at one time
Notes - the main body of the simulator checks if an interrupt is due to occur at a particular time	

Memory management

Segment name	Segment function
Gaplist Spaceincore Newjob	1. Keeping records of gaps left in memory (fragmentation) 2. Checking if there is enough space in core which is free and can be assigned to a job 3. Assigning space in core to a job

Each of the above sections in the categories has associated with it a segment in the skeleton structure. Hence, the segments in the skeleton structure which reflect the basic operations of a system have been defined.

Appendix 2

THE BREAKDOWN OF THE MAIN BODY OF S.C.O.P.E.

The main body segment in S.C.O.P.E. can be seen as being divided into sections according to its various functions.

The first function is to call the initializing segment (INIT) in order to set up the initial state from which the simulation run will commence.

The second part achieves the effect of more than one processor working simultaneously (if necessary) by using the event generating routine to determine the next event for each processor in turn. Only after each processor has dealt with its respective event, is the master clock moved on.

The third part is responsible for the organised flow of processors through the critcal region, created by the processing management category of operations (see Appendix 1).

The fourth part determines when a channel is free and when it will subsequently have completed the next input/output operation on the queue. It uses INTRUPLIST to generate an interrupt accordingly.

The fifth part determines when an interrupt is due to occur.

Also the main body is responsible for the output from the simulator and can provide "snapshots" during the run or end-of-run diagnostics.

Finally, there is a mechanism in the main body which prevents the simulation run from merely looping around the main body whilst nothing is happening on the system. This mechanism, determines the time at which the next event will occur and alters various parameters accordingly. This mechanism creates a new segment - NXTEVENTIME - which should be seen as being part of the main body.

Appendix 3

A DESCRIPTION OF THE INTERFACES BETWEEN THE VARIOUS SEGMENTS OF S.C.O.P.E. (skeleton structure)

In implementing S.C.O.P.E. the first decisions made about the interfaces between segments, concerned the relationships of the segments within each category of operations.

In the input/output operations category, the ROLLIN, ENDJOB and INOUT segments all call the IOLIST segment, because each requires an entry to be placed on the input/output queue.

In the processing management category, all calls to the REALLOC and INSERT segments must pass through IQLIST in order to check if there is contention for the critical region which can be visualised, from the implementors point of view, as being around the segments REALLOC and INSERT.

In the interrupt management category, the segment INTRUPLIST is used by the main body to check if an interrupt is due to occur and, if this is the case, the segment INTERRUPT is called. INTRUPCHECK is called immediately after in order to check if another interrupt requires attention.

In the memory management category, NEWJOB calls SPACEINCORE in order to determine whether there is space in core for a new job to be brought in. SPACEINCORE determines this via the records maintained by the segment GAPLIST.

The remaining standard interfaces between the segments have the effect of joining the categories and the other segments, i.e. the main body, and the job and event generating segments, together to form a cohesive system. (See figure 13 for a diagramatic representation of these interrelationships).

Calls into the input/output operations category come from:

1. NEWJOB, which calls ROLLIN if it is possible to fit another job into memory

2. INSTRUGEN, which calls ENDJOB if the event which occurs on a particular job is the completion of that job or INOUT if the event generated is an input/output request by a job

Calls into the processing management section come from:

1. INSTRGEN, which calls JQLIST if a processor is idle or if it requires to place a job on the ready queue (in the event of its timeslice expiring).

2. INTRUPCHECK, which calls JQLIST if there are no more interrupts which require attention, in which case the processor must be reallocated with a job

3. INTERRUPT, which calls INSERT, because the processor must place the job it is currently executing on the ready queue before dealing with any interrupts

4. JQLIST timing section of the main body, which allows access to INSERT and REALLOC

Calls into the interrupt management section come from:

1. The MAIN BODY; one is prompted by the completed execution of an input/output entry by a channel. A call is then made to INTRUPLIST.
   The other is prompted by an entry on INTRUPLIST reaching its time for attention. A call is then made to INTERRUPT.

Calls into the memory management section come from:

1. INTERRUPT, which calls GAPLIST if a job has been rolled out, in order to record the gap left, and which calls NEWJOB in an attempt to fill this gap with another job.

Calls into the job generation segments come from:

1. INIT, which calls JOBSIZGEN, in order to generate some jobs for preparing the initial state of the simulation run

2. NEWJOB, which calls JOBSIZGEN to generate the next job to be rolled in

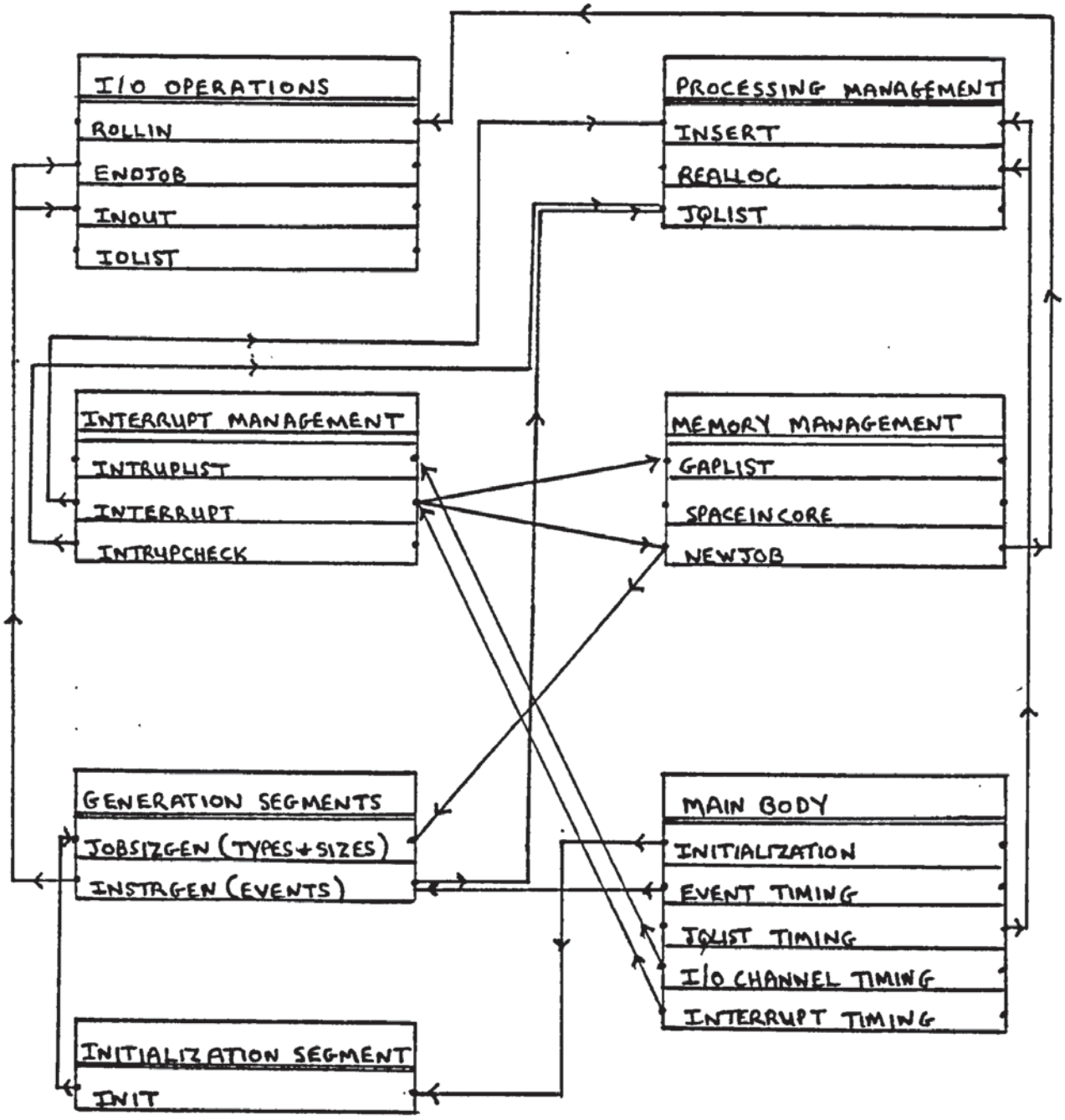
3. Event timing section of MAIN BODY, which calls INSTRGEN, to generate the events of the jobs being processed.

Calls into the initializing segment come from:

1. MAIN BODY, which calls INIT to set-up the state of the system before commencing the run.

Figure 13

DIAGRAMATIC REPRESENTATION OF THE INTER-RELATIONSHIPS BETWEEN THE VARIOUS SEGMENTS OF S.C.O.P.E.



Appendix 4

LANGUAGE USED IN CODING S.C.O.P.E.

As mentioned earlier, the approach advocated in this study can assist in determining the languages which are suitable for the package being developed.

There are various languages which have been specifically designed for use in simulation studies. Examples of these are Simula (O1), SIMSCRIPT II (K90), (M7), SOL (K19), (K20), GPSS III (I1), CSL (T1), GASP (T1), CCP (T1) and ASPOL (M2).

However, there are certain difficulties associated with such special purpose simulation languages. O'Brien (I10) holds the view that they are difficult to manipulate and do not do what the user really wants them to do and that they do not adequately express the types of processes and entities which exist in a computer. He also says that simulation languages are very difficult to learn quickly.

For these and other reasons, it was decided that Algol 68R should be used in this study. The advantages of Algol 68R are primarily that:

1. It is available on the local installation, that is the ICL 1904S installation at the University of Aston operating under the George III operating system.
2. It is a language with which the authour is already familiar.
3. It is a reasonably widespread language, especially in academic establishments, and thus this research might be of interest to works in other establishments.
4. It contains the important concept (to this study) of modularity.

The choice of Algol 68R was also influenced by a desire to determine whether a general-purpose language of this sort is actually suitable for this particular application. Some notes concerning the performance of Algol 68R in the context of this simulation study can be found in Appendix 5.

However, before this choice was confirmed, it was necessary to determine whether Algol 68R contained the features which are desirable in a simulation study. Teichrow and Lubin (T2) carried out a survey of various simulation languages which included SOL, GPSS II, SIMSCRIPT, CSL, GASP and CLP. From this study they summarised the desirable features which a simulation language, and the model coded with it, should possess. It is of interest to note that the results of the study showed that many of the languages did not include all the desirable features.

The desirable features included:

1. Five types of variables, i.e. arrays, lists, records, fields and groups of records. Boolean and complex variables should also be considered.
2. Ability to create and destroy temporary entities.
3. Capability of simulating parallel activity.
4. Timing device or routine.
5. Extensive list processing capability.
6. Recursion - subprograms and procedures and extended uses thereof.
7. Commands to compile statistics easily.
8. Facility for generating variates having specified and arbitrary probability distributions.
9. Flexible report generator - as much information as necessary gained from the simulator plus a standard print out of results. Results should be so presented that they can be used without transcription.
10. Every possible aid to facilitate experimentation. A model should provide multiple runs, changing parameter values, changing data, analysing results, stopping and restarting runs.
11. Language should have aids such as flowcharts, decision tables etc. to help formulate models.

It was found that a model developed as advocated in this study and coded in Algol 68R would contain many of these features. For example Algol 68R can have any type of variable - a type can even be specified by the user; it has extensive list processing capabilities, recursion, random number generators etc. In fact, it was found that the language itself only lacked one of the above properties, the quick compilation of

statistics aids such as flowcharts. However, due to the stochastic nature of the model the statistics to be compiled are minimal and due to the design process outlined in section 5.1., aids such as flowcharts and decision tables become unnecessary, though they could be developed by the user.

However, Algol 68R was found to contain the features which would be necessary within S.C.O.P.E., such as the variable types, list processing, random number generators, structures, procedures - modularity, loops, formatted transput etc. and this is one factor which is basic to the methodology, i.e. the choice of language should be dictated by the structures required in the model and the availability of the language. Hence, Algol 68R satisfied the general requirements of Teichrow and Lubin and the specific requirements of S.C.O.P.E.

Appendix 5

THE SUITABILITY OF ALGOL 68-R AS A SIMULATION LANGUAGE

One of the questions which surrounded the introduction of general-purpose languages such as Algol 68-R, was whether they were adequate in what have become specialised areas of Computer Science. In order to answer this question, it is necessary to use such a language in these areas and to report on its suitability. The use of Algol 68-R as a simulation language is such a specialised area. Therefore, some information regarding the performance of Algol 68-R in this study should be supplied.

Due to the facility in Algol 68-R of declaring the user's own complex structures and treating such a structure as one entity, the list processing capabilities required in this simulation study became a relatively simple operation, and yet still included all the details which were required. For instance, one entity in the list of gaps in memory (maintained by GAPLIST) contains the size, the location of the beginning and the location of the end of the gap. These entities can either be accessed whole or each of these pieces of information can be accessed separately, if necessary. The ability to create and destroy the temporary entities of information on a list is also relatively simple, since creation involves only the assignment of values to each of the fields in the structure and the placing of this on the list, and destruction involves only a reassignment of the pointers in certain entities to disregard the existence of another.

Another feature of Algol 68-R which was found to be relevant to this study was its capabilities for formatted transput. Besides commands such as PRINT (SPACE); PRINT (NEWLINE); PRINT (NEWPAGE); which are standard amongst programming languages, Algol 68-R includes formatted transput procedures which allow the user to control the style of printing of output values in a compact and convenient way.

An introduction to the use of formats is given in Chapters 12 and 13 of the "Algol 68-R User's Guide" (2nd Edition , HMSO July 1974). By the use of these formats the user can specify the pattern of his output values: for instance, the format ⟨3.6⟩ denotes a real value which will be output as a space or minus sign, 3 digits before and 6 digits after the decimal point. Algol 68-R also includes special symbols for spaces, newlines, back spaces etc. which can be included into a format to facilitate the ordering of the output into tables or some other desired format. Literal character strings can also be inserted into the output formats and these enable headings or comments to be output with the tables. There is also the facility for providing graphical output using Algol 68-R, however, this facility was not utilized in this particular study. All these factors enable the user to provide meaningful and understandable output from a simulation model and this is very important to the effectiveness of a model.

The facilities which a model should possess in order to be satisfactory, according to Teichrow and Lubin (T1), i.e. the timing device, parallel activity and the flexible report generator, did not pose any problems when coded in Algol 68-R. The first two facilities are inherently tied in the model, which is due to the difficulty of simulating parallel activity on a sequential machine. Rather than have the parallel operations actually carried out in parallel, each event is dealt with in turn within a loop and it is only at the end of the loop that the simulated clock is moved on. The effects of these events are then carried through in a similar manner. Thus, the simulation of parallel events becomes a loop and a clock increment and this is not difficult to implement due to the properties of the Algol 68-R WHILE statement, which is easily manipulated. The report generators in the model are flexible by nature of the theory of the project, and any changes to be made to them by subsequent users should be simplified by the output formats previously described.

Algol 68-R was also found to be very suitable for coding the stepwise design of each segment. Due to the fact that it contains high-level constructs, the breakdown of the segments into their component functions did not need to be unacceptably detailed. The high-level of constructs contained in Algol 68-R also facilitate the 'top-down' approach implied in the use of stepwise refinement, since jump statements become unnecessary to a large extent.

The concept of modularity is very important to the design of this model and it was found that the degree of modularity required was easily interpreted in Algol 68-R. Algol 68-R is a procedural language and its procedures are not difficult to use. They can return either a numerical or boolean value or, as is mainly the case with the segments of S.C.O.P.E., merely carry out some separate operation without necessarily yielding an actual value.

Due to the stochastic nature of S.C.O.P.E., the random number generating procedures of Algol 68-R had to be reviewed. These procedures generate pseudo random numbers based on the use of "M-sequences". Each call of a procedure produces a number which is, as far as practicable, statistically independent of what it has delivered before, but the whole sequence repeats after 8388607 calls of the same procedure (except for the "gaussion" procedure whose period is twice this number). When a program using one of these procedures is run more than once, different sequences of random numbers will be obtained each time. However, a resetting variable associated with each procedure allows the user to initialize the sequence and hence, if the same initial value is used in different runs, the sequences will be the same. This aspect of the Algol 68-R random number generators was found to be effective in running different systems organisations with the same job specifications and events and thus ensured accurate comparisons. A difficulty arose however, because, if two (or more) sequences are required from a particular

procedure in one program, more than one resetting variable is needed. As the Algol 68-R Library provides only one of these for each procedure, the user has to declare his own. This formed the crux of the difficulty encountered in this study, since the model requires two independent set sequences - one for job generation and one for event generation. These sequences also had to be within a range specifiable by a user of the model. Only one Algol 68-R procedure - "Random" - was found to have this latter facility and so two separate random number generating procedures could not be used. 'Random' delivers a real number between 0.0 and 1.0 and each of the numbers R/8388608 occurs exactly once, where R is an integer in the range 1 to 8388607 inclusive. Hence, an even distribution is gained. The user can easily specify a range using this procedure by multiplying the number generated by the range he requires. Thus, if a range of 0 - 30 is required, the user calls 'Random' and multiplies the number generated by 30. Since two separate procedures could not be used in S.C.O.P.E., it was necessary to devise a scheme to provide two distinct sequences. The scheme was to declare two other variables which could act as resetting variables for each sequence and to provide a procedure within S.C.O.P.E. for each sequence. The procedures developed are given below:

```
'PROC' RAND1 = 'REAL' :
  (NORMRAND:= SEED1 ;
  RV: = RANDOM ;
  SEED1: = NORMRAND ; RV) ;

'PROC' RAND2 = 'REAL' :
  (NORMRAND: = SEED2 ;
  RV: = RANDOM ;
  SEED2: = NORMRAND ; RV) ;
```

NORMRAND - RESEETING VARIABLE in Algol 68-R Library

SEED 1 - Resetting variable for 1st sequence

SEED 2 - Resetting variable for 2nd sequence.

If a number in the first sequence is required, RAND1 is called. The Algol 68-R resetting variable NORMRAND is given a value by SEED1. RANDOM is then called which uses NORMRAND to generate a number. The value in NORMRAND after the generation is now different and becomes the

initializing value for the next call. This value is then assigned to SEED1. Procedure RAND2 operates in the same manner. Therefore, if SEED1 and SEED2 are given different values two different sequences will be produced.

Although it was not a difficult task to devise the scheme to overcome this problem with the Algol 68-R random number procedures, it is still a failing in the language which needs to be pointed out. However, overall, Algol 68-R was found to be a satisfactory language for this study and had many features which assisted its development.

Appendix 6

DOCUMENTATION INDEX

This index constitutes the second level of documentation included in this package (see section 4.5.3.). The various alternatives to segments in the skeleton are numbered from 2 upwards; alternative number 1 being the skeleton segment. Additional segments are numbered from 1. The index contains a comment for each segment explaining the operations simulated by that segment. The detailed documentation for each segment (the third level of documentation (see section 4.5.3.)) is given in the library together with a listing of that segment. The exception to this is the skeleton which is listed in total in Appendix 7 together with its detailed documentation.

SKELETON

CHANNELTIME (1)

Calculates the time of the next input/output operation or, more specifically, the time when a channel will be free to carry out the next operation.

INTRUPLIST (1)

Please the event of an interrupt which is to occur onto a list and makes a note of relevant data such as the type of interrupt, the type of operation to be performed, the number of the job with which it is associated and the time when it will occur. The list is time ordered.

IOLIST (1)

Places an input/output request onto a queue which is served by the channels. Relevant data such as the time when the request would have been placed on the list, the type of interrupt which will be generated after the operation has been completed, the type of input/output operation to be carried out and the name of the job from which the request came are included in each entry. The list is time ordered.

INSERT (1)

Places an entry onto the ready queue noting the time when the jobs will be ready to run. The ready queue is time ordered - i.e. round robin strategy.

INOUT (1)

Initialises the input/output operation by blocking the job in question, by adding on delays for setting up the channel, and by setting variables to denote the operations to be carried out.

ROLLIN (1)

Initiates the rolling in of a job into memory and blocks it until the rolling in operation has been completed. Adds on a delay for setting up an entry on the channel queue.

REALLOC (1)

Manipulates the ready queue, when time ordered by INSERT (1), and assigns the first entry on the queue to a processor requiring a job. Tests if the time of the entry (i.e. the time when it will be ready to run) is less than or equal to the time of the processor requiring a job, since otherwise the entry would not have been in the queue when the processor examined it. Sets JBLISTIME to the time the processor leaves the critical region, which prevents other processors entering before that time.

JOBSIZGEN (1)

Generates via random number generators and probability distributions the type of job and the size of the job.

JQLIST (1)

This segment places entries onto a queue of processors waiting to enter the critical region surrounding the ready queue. The list is time ordered.

INIT (1)

This segment is responsible for setting up the state from which

the simulation run will proceed and for initializing the simulation variables. The memory is partitioned and is filled with as many jobs as possible. No interrupts or input/output requests etc. are generated so the simulation will start from a null state.

GAPLIST (1)

Keeps a check of all gaps in memory and joins any gaps which are adjacent. Maintains a list of these gaps. This segment deals with a partitioned type of memory management scheme.

SPACEINCORE (1)

This routine searches for enough space in memory for a job to be brought in and thus uses the data maintained by GAPLIST (1). It delivers a boolean value to indicate whether a large enough gap has been found or not. If a gap is found its entry is removed from the list of gaps but a pointer remains pointing to it so that this entry can subsequently be used by NEWJOB (1).

NEWJOB (1)

Initiates selection and rolling in of new jobs. This segment deals with paritioned memory management schemes.

INTRUPCHECK (1)

Checks if there are any interrupts which are to occur at the time the processor leaves the INTERRUPT segment after having just dealt with one. If there is, the processor is set to a null state (PR [I] = -1) which stops any events happening to it until it checks if there are any more interrupts. If not, the processor is passed to JQLIST in order to reallocate it with another job. This segment assumes that the same processor deals with all interrupts.

INTERRUPT (1)

Deals with the interrupts. Any work the processor interrupted is executing is placed on the ready queue. The type interrupt is checked

to show if it requires swapping a job from the blocked to the ready queue or if a job has just been rolled out. In the former case the entry for the job is placed on the ready queue in the latter a note is made of the gap left and NEWJOB is called to bring in another job.

ENDJOB (1)

When a job terminates this segment sets up the rolling out of that job and then calls IOLIST which uses the variables set in ENDJOB to put an entry onto the queue for input/output resources, so that the job will be rolled out.

INSTRGEN (1)

Generates the next event to occur on the system, within particular jobs being processed. Probability distributions govern the range of events which are chosen by random numbers. The events included in this segment are:

1. A job's timeslice has been completed
2. A job requests input/output
3. A job terminates.

Processors entering this routine idle (PR [I] = 0) are automatically passed to JQLIST from where an attempt will be made to reallocate them with a job.

NXTEVENTIME (1)

Determines the time the next event will occur on the system and sets the variable TEMP to that time. This segment is implemented so that the simulator does not loop whilst nothing is happening on the simulated system.

MAINBODY (1)

The main body controls the timing of the simulator as it contains the master clock, REALTIME. It runs the simulator for a set number of jobs, printing results both during and after the run. It controls the parallelism within the system by checking and co-ordinating the disjoint queues of events which are due to occur.

LIBRARY

INIT (2)

This segment is responsible for setting up the state from which the simulation run will proceed and for initializing the simulation variables. It deals with segmented memory access and places a job in every segment. No interrupts or input/output request etc. are generated so the simulation will start from a null state.

GAPLIST (2)

This segment keeps a check on all gaps in memory. As it deals with segmented memory access, it notes which memory segments are unused.

SPACINCORE (2)

Checks the list maintained by GAPLIST (2) for a segment which is free, and delivers a boolean value to indicate whether a free segment has been found or not. If a segment is found SPACEINCORE (2) alters the records to show that it is being used and leaves the variable SEG equal to the number of that segment; this information is subsequently used in NEWJOB (2).

NEWJOB (2)

Initiates the rolling in of new jobs into a segmented memory, noting relevant information such as job size, the segment the job will occupy, the amount of space wasted in the segment and the job type in array JOB.

NEWJOB (3)

Selects and initiates the rolling in of new jobs. Same as NEWJOB (1) except that if a job cannot be fitted into memory due to fragmentation the jobs in memory are relocated to gain enough space. Relocation takes place if the fragmentation level is greater than the jobload. (See RELOC (1)). This segment takes into account tentatively started jobs as in the George 3 operating system.

RELOC (1)

Relocates the jobs in memory in order to reduce fragmentation and adds on a delay to all timing mechanisms because the system cannot be used whilst relocation is taking place.

JQSIZEPROBE (1)

Counts and outputs the size of the queue of processors waiting to enter the critical region around the job queues.

IOQPROBE (1)

Records the size of the queue for input/output channel resources.

HLS (1)

This segment decides whether or not to tentatively start a job, based on whether the size specifications (given by the user of the system (assumed to be 40K) of all the tentatively started jobs is less than a system parameter set at 200K.

FREECOREADJ (1)

Whether jobs are started or terminated on the George 3 system, the amount of memory needed for George chapters and red tape headings is altered. This segment alters the variable FREECORE, which is the amount of memory left for core images, according to these fluctuations. This segment is for use in transient operating systems.

CHAPQUOTA (1)

Calculates the amount of memory which should be necessary for George chapters based on the number of jobs started. The formula is given in the George 3 manual. This segment is called from FREECOREADJ (1).

REDTAPEQUOTA (1)

Calculates how much memory will be taken up by red tape headings for jobs based on a formula given in the George 3 operating system manual. The formula is based on the number of jobs started and the number of jobs tentatively started. This segment is called from FREECOREADJ (1).

NEWJOB (4)

Selects and initiates the rolling in of new jobs. Relocates memory if enough space is not found for the job and if the fragmentation level is greater than the jobload.

INSERT (2)

Orders the ready queue on a priority basis according to the job types. Jobs of type 1 taking top priority, jobs of type 2, second priority etc.

INSERT (3)

Orders the ready queue according to certain types of jobs. In this case, all even numbered job types have top priority.

INIT (3)

This segment is responsible for setting up the state from which the simulation run will proceed and for initializing the simulation variables. This segment was developed for a George 3 system model. The memory is partitioned and is filled with jobs until the quota of memory for George 3 chapters and red tape headings prevent this. The simulation then starts from a null state since no interrupts or input/output requests are generated in this segment.

INSTRGEN (2)

Generates the next event to occur on the system for each processor. Probability distributions govern the range of events which

are chosen via random numbers. This segment is part of a George 3 system model and has six events:

1. time-slice expired
2. Input/output request
3. termination
4. chapter change
5. chapter transfer
6. M.O.P. job started or job placed into background queue.

Processors entering this routine idle are automatically passed to segment JQLIST where an attempt will be made to reallocate them with a job.

APPENDIX 7

SKELETON LISTING AND DOCUMENTATION

MOP :SSP8086.MOPPMB

```
#LISTING OF :SSP8086.SCOPSKEL(15/)  PRODUCED ON  5SEP78 AT 15.19.33

#GB.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 1MAY79 AT 13.20.42 USING  U15

DOCUMENT     SCOPSKEL


TIMESIM
'BEGIN'
'BOOL'CONTINUE:='TRUE',OK:='TRUE',GAPFOUND,OK1,BOOL,INTRUPTYPE,LASTJOBIN:='TRUE';
 [1:70 ]'REAL'JBSIZE,JBBLOCKED,JOBSTART,JOBFINISH,JBBLOCKSTART,SYSTIME;
[1:100]'INT'IOQ;
'REAL'RV,SAMPLEVAR:=50.0,PROIDLESUM:=0.0,CHANNELIDLESUM:=0.0,PROJQWAITSUM:=0.0,JBLOCKSUM:=0.0;
'INT' SEED1:=11,SEED2:=7,IOUSIZE:=0,JQSIZE:=0,FREECORE,SAVE,COREIMAGE:=0,TOTAL;
'INT'JOBNEW ,JOBCOUNT:=0,X,NO,I,SIZ,BEG,FIN,FRAGVAR,
       JOBNO,INTRUPPTR1:=1,INTRUPPTR2:=1,JOBLOAD:=0,JOBSIZE,IOSUM:=0,
     FRAGLEVEL:=0,NEXTJOB:=0,UNUSECORE:=0,FRAGCOUNT:=0,PERCENT,AVGUNUSED,
     CHA,NXTTYPE:=0,SLICEPROB,IOPROB,TERMPROB,JOBTYPE,
     SYSCORE:=0, SYSCORECOUNT:=0,USERCORE:=0,USERCORECOUNT:=0,
      USERCOREAVG,SYSCOREAVG;


'REAL'REALTIME:=0.0,IOTIME:=0.0,JBLISTIME:=0.0,TEMP,CHAPCHDEL;
'STRING'TITLE;
'INT'JOBTYPNO:=20,JOBSIZNO:=8,EVENTNO:=8;
[1:JOBTYPNO]'INT'TYPPERCENT;
[1:JOBSIZNO]'INT'JOBSIZES,SIZPERCENT;
[1:JOBTYPNO,1:EVENTNO]'INT'EVENTPROBS;
'INT'ESSENTFUNCT,FREEPOOL,SEGNO,SEGSIZE;

'INT'MEMSIZE,N:=20,CHANO:=20;

'REAL'IODEL,REALLOCDEL,INSERTDEL,ROLLOUTDEL,
       RELOCDEL,CHANNELDEL,GAPCHECKDEL,ROLLINDEL,
       INTRUPDEL,SWAPQDEL,HLSDEL,LLSDEL,ACCOUNTDEL;
[0:70,1:4]'INT'JOB;
[1:N]'REAL'IDLETIME,PROJQWAIT,OSTIME;
[1:CHANO]'REAL'CHANNEL,CHANNELIDLE;
[1:70]'BOOL'NOTROLLEDOUT,WAITBLOCKED,TERMINED;
[1:70]'INT'TSEXPD,IOREQ;
[1:N ]'INT'PR;
[1:N]'REAL'PRTIME;
[1:N]'BOOL'NOTENT,IO;
[1:N]'BOOL'NEWJBOOL;
'MODE''INTRUPENT'='STRUCT'('REAL'INTRUPTIME,'BOOL'INTRUPTYP,'BOOL'IO,'INT'JOBNO,
                              'REF''INTRUPENT'NXT);
'REF''INTRUPENT'IHEAD,IPTR1,IPTR2,ITAIL;
'REF''INTRUPENT'NIL='NIL';
'MODE''IOLISTENT'='STRUCT'('REAL'ICTIM,'BOOL'IO,'BOOL'INTRUPTYPE,
                           'INT'JOB,'REF''IOLISTENT'IONXT);
'REF''IOLISTENT'IOHEAD,IOTAIL,IOPTR1,IOPTR2;
'REF''IOLISTENT'NOTHING='NIL';
'MODE''JQENT'='STRUCT'('INT'PRO,'REAL'QTIME,'REF''JQENT'QNEXT);
'REF''JQENT'QHEAD,QPTR1,QPTR2;
'REF''JQENT'NILL='NIL';
'MODE''GAPENTRY'='STRUCT'('INT'SIZE,'INT'BEGIN,'INT'END,
```

```
                                'REF''GAPENTRY'PTR);
'REF''GAPENTRY'NULLUS='NIL';
'REF''GAPENTRY'GPTR1,GPTR2,GPTR3,GHEAD,GTAIL;
'MODE''JOBENTRY'='STRUCT'('INT'P,'REAL'TIME,'REF''JOBENTRY'NEXT);
'REF''JOBENTRY'NULL ='NIL';
'REF''JOBENTRY'PTR1,PTR2,HEAD,TAIL;

'C'GENERATES TWO DISTINCT SEQUENCES OF RANDOM NOS TO BE USED FOR GENERATING
    JOB CHARACTERISTICS AND SEQUENCES OF EVENTS RESPECTIVELY'C'
'PROC'RAND1='REAL':
    (NORMRAND:=SEED1;
     RV:=RANDOM;
     SEED1:=NORMRAND;RV);
'PROC'RAND2='REAL':
    (NORMRAND:=SEED2;
     RV:=RANDOM;
      SEED2:=NORMRAND;RV);

'C'CALCULATES TIME WHEN A CHANNEL WILL BE FREE TO CARRY OUT AN ISO OPERATION'C'

'PROC'CHANNELTIME='VOID':
   (IOTIME:=CHANNEL[1];CHA:=1;
    'FOR'I'TO'CHANO'DO'
      'IF'CHANNEL[I]<IOTIME
      'THEN'IOTIME:=CHANNEL[I];CHA:=I
      'FI');

'C'PLACES THE EVENT OF AN INTERRUPT ON A LIST 'C'

'PROC'INTRUPLIST=('REAL'TIME,'INT'JOBNO,'BOOL'BOOL,'BOOL'B)'VOID':
    ('IF'IHEAD'IS'NIL
    'THEN'IHEAD:=ITAIL:='INTRUPENT':=(TIME,BOOL,B,JOBNO,'NIL')
    'ELSE'CONTINUE:='TRUE';IPTR1:='NIL';IPTR2:=IHEAD;
          'WHILE'CONTINUE'AND'(NXT'OF'IPTR2'ISNT'NIL)'DO'
             'IF'TIME>=INTRUPTIME'OF'IPTR2
             'THEN'IPTR1:=IPTR2;IPTR2:=NXT'OF'IPTR2
             'ELSF'IPTR1'IS'NIL
             'THEN'IHEAD:='INTRUPENT':=(TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                                  (TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'FI';
          'IF'NXT'OF'IPTR2'IS'NIL
          'THEN''IF'INTRUPTIME'OF'IPTR2<=TIME
                'THEN'ITAIL:=NXT'OF'ITAIL:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,'NIL')
             'ELSF'IPTR1'IS'NIL
             'THEN'IHEAD:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,IPTR2)
             'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,IPTR2)
             'FI'
          'FI'
    'FI');

'C'PLACES AN ISO REQUEST ONTO A LIST WHICH IS SERVED BY THE CHANNELS'C'

'PROC'IOLIST=('INT'I)'VOID':
   ('IF'IOHEAD'IS'NOTHING
     'THEN'IOHEAD:=IOTAIL:='IOLISTENT':=
```

```
            (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
     'ELSE'CONTINUE:='TRUE';IOPTR1:='NIL';IOPTR2:=IOHEAD;
          'WHILE'CONTINUE'AND'(IONXT'OF'IOPTR2'ISNT'NOTHING)'DO'
           ('IF'PRTIME[I ]>=IOTIM'OF'IOPTR2
            'THEN'IOPTR1:=IOPTR2;IOPTR2:=IONXT'OF'IOPTR2
            'ELSF'IOPTR1'IS'NOTHING
            'THEN'IOHEAD:='IOLISTENT':=
                  (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                  CONTINUE:='FALSE'
            'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                  (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                  CONTINUE:='FALSE'
            'FI');
            'IF'IONXT'OF'IOPTR2'IS'NOTHING
          'THEN''IF'IOTIM'OF'IOPTR2<=PRTIME[I ]
                'THEN'IOTAIL:=IONXT'OF'IOTAIL:='IOLISTENT':=
                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
                 'ELSF'IOPTR1'IS'NOTHING
              'THEN'IOHEAD:='IOLISTENT':=
                       (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                 'FI'
            'FI'
     'FI');

'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
    IT WILL BE READY TO RUN'C'

'PROC'INSERT=('INT'I)'VOID':
   (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL;
'IF'HEAD'IS'NULL
   'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],'NIL')
   'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
         'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
          ('IF'PRTIME[I ]>=TIME'OF'PTR2
           'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
           'ELSF'PTR1'IS'NULL
           'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                 CONTINUE:='FALSE'
           'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                  CONTINUE:='FALSE'
           'FI');
           'IF'NEXT'OF'PTR2'IS'NULL
           'THEN''IF'TIME'OF'PTR2<=PRTIME[I ]
                 'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
                            (PR[I ],PRTIME[I ],'NIL')
                 'ELSF'PTR1'IS'NULL
                 'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
                 'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
                 'FI'
           'FI'
   'FI');

'C'INITIALISES I+O OPERATION AND BLOCKS JOB'C'

'PROC'INOUT=('INT'I)'VOID':
   (PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;JOBLOCKSTART[PR[I]]:=PRTIME[I];
    SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;

IO[I]:='TRUE';WAITBLOCKED[PR[I ]]:='TRUE';
 INTRUPTYPE:='TRUE';
 IOLIST(I));
```

```
'C'INITIATES ROLLING IN OF A JOB'C'
'PROC'ROLLIN=('INT'I)'VOID':
      (JOBSTART[PR[I]]:=PRTIME[I];
    PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;
    SYSTIME[PR[I])'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='TRUE';
   WAITBLOCKED[PR[I ]]:='TRUE';
    IOLIST(I));

'C'ASSIGNS FIRST ENTRY ON READY QUEUE TO A PROCESSOR'C'

'PROC'REALLOC=('INT'I)'VOID':
    ('IF'HEAD'IS'NULL
      'THEN'
          PR[I ]:=0
     'ELSE'PTR2:=HEAD;
          'IF'TIME'OF'PTR2> PRTIME[I ]
      'THEN'
                PR[I ]:=0
          'ELSF'NEXT'OF'PTR2'ISNT'NULL
        'THEN'HEAD:=NEXT'OF'PTR2;
              PR[I ]:=P'OF'PTR2
          'ELSE'HEAD:=TAIL:='NIL';
              PR[I ]:=P'OF'PTR2
          'FI'
    'FI';
   PRTIME[I ]:=PRTIME[I ]+REALLOCDEL;
        'IF'PR[1]>0'THEN'OSTIME[I]'PLUS'REALLOCDEL'FI';JBLISTIME:=PRTIME[I ]);

'C'GENERATES JOB CHARACTERISTICS'C'

'PROC'JOBSIZGEN='VOID':
      ('IF'LASTJOBIN'THEN' NO:='ENTIER'(RAND1*100);
    'IF'NO<=TYPPERCENT[1]'THEN'JOBTYPE:=1
    'ELSF'NO<=TYPPERCENT[2]'THEN'JOBTYPE:=2
    'ELSF'NO<=TYPPERCENT[3]'THEN'JOBTYPE:=3
    'ELSF'NO<=TYPPERCENT[4]'THEN'JOBTYPE:=4
    'ELSF'NO<=TYPPERCENT[5]'THEN'JOBTYPE:=5
    'ELSF'NO<=TYPPERCENT[6]'THEN'JOBTYPE:=6
    'ELSF'NO<=TYPPERCENT[7]'THEN'JOBTYPE:=7
   'ELSF'NO<=TYPPERCENT[8]'THEN'JOBTYPE:=8
   'ELSF'NO<=TYPPERCENT[9]'THEN'JOBTYPE:=9
   'ELSF'NO<=TYPPERCENT[10]'THEN'JOBTYPE:=10
   'ELSF'NO<=TYPPERCENT[11]'THEN'JOBTYPE:=11
   'ELSF'NO<=TYPPERCENT[12]'THEN'JOBTYPE:=12
   'ELSF'NO<=TYPPERCENT[13]'THEN'JOBTYPE:=13
   'ELSF'NO<=TYPPERCENT[14]'THEN'JOBTYPE:=14
   'ELSF'NO<=TYPPERCENT[15]'THEN'JOBTYPE:=15
   'ELSF'NO<=TYPPERCENT[16]'THEN'JOBTYPE:=16
   'ELSF'NO<=TYPPERCENT[17]'THEN'JOBTYPE:=17
   'ELSF'NO<=TYPPERCENT[18]'THEN'JOBTYPE:=18
   'ELSF'NO<=TYPPERCENT[19]'THEN'JOBTYPE:=19
    'ELSE'JOBTYPE:=20
    'FI';
     NO:='ENTIER'(RAND1*100);
          'IF'NO<=SIZPERCENT[1]'THEN'JOBSIZE:=JOBSIZES[1]
          'ELSF'NO<=SIZPERCENT[2]'THEN'JOBSIZE:=JOBSIZES[2]
          'ELSF'NO<=SIZPERCENT[3]'THEN'JOBSIZE:=JOBSIZES[3]
          'ELSF'NO<=SIZPERCENT[4]'THEN'JOBSIZE:=JOBSIZES[4]
          'ELSF'NO<=SIZPERCENT[5]'THEN'JOBSIZE:=JOBSIZES[5]
        'ELSF'NO<=SIZPERCENT[6]'THEN'JOBSIZE:=JOBSIZES[6]
```

```
          'ELSF'NO<=SIZPERCENT[7]'THEN'JOBSIZE:=JOBSIZES[7]
          'ELSE'JOBSIZE:=JOBSIZES[8]
          'FI'
    'ELSE'LASTJOBIN:='TRUE'
    'FI');

'C'ORGANISES QUEUE OF PROCESSORS WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'

'PROC'JQLIST=('INT'I)'VOID':
   ('IF'QHEAD'IS'NILL
    'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],'NIL')
    'ELSE'CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
          'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
          'DO'
           ('IF'PRTIME[I ]>=QTIME'OF'QPTR2
            'THEN'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
            'ELSE'QPTR1'IS'NILL
            'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],QPTR2);CONTINUE:='FALSE'
            'ELSE'QNEXT'OF'QPTR1:='JQENT':=(I,PRTIME[I ],QPTR2);
                  CONTINUE:='FALSE'
            'FI');
            'IF'QNEXT'OF'QPTR2'IS'NILL
            'THEN''IF'QTIME'OF'QPTR2<=PRTIME[I ]
                  'THEN'QNEXT'OF'QPTR2:='JQENT':=(I,PRTIME[I ],'NIL')
                  'ELSF'QPTR1'IS'NILL
                  'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],QPTR2)
                  'ELSE'QNEXT'OF'QPTR1:='JQENT':=(I,PRTIME[I ],QPTR2)
                  'FI'
            'FI'
            'FI';
             NOTENT[I]:='FALSE');

'C'INITIALISES SIMULATION VARIABLES AND STATEFROM WHICH SIMULATION BEGINS'C'

'PROC'INIT='VOID':
  (OK1:='TRUE';
   'FOR'I'FROM'0'TO'70'DO'
      'FOR'J'TO'4'DO'JOB[I,J]:=0;

READ(TITLE);PRINT((TITLE,NEWLINE,"--------------------",NEWLINE));
PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(FREECORE);PRINT(("FREE CORE =",FREECORE,NEWLINE));
    READ(MEMSIZE);PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE));
    READ(N);PRINT(("NO OF PROCESSORS =",N,NEWLINE));
    READ(CHANO);PRINT(("NO OF CHANNELS =",CHANO,NEWLINE));
    READ(IODEL);OUTF(STANDOUT,$"I!O DELAY ="<3.1>L$,(IODEL));
    READ(REALLOCDEL);OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL));
    READ(INSERTDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL));
    READ(ROLLOUTDEL);OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL));
    READ(ROLLINDEL);OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL));
    READ(RELOCDEL);OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL));
    READ(CHANNELDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL));
    READ(GAPCHECKDEL);OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL));
    READ(INTRUPDEL);OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL));
    READ(SWAPQDEL);OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAPQDEL));
    READ(HLSDEL);OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL));
    READ(LLSDEL);OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL));
    READ(ACCOUNTDEL);OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL));
    READ(SEGSIZE);PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE));
    READ(SEGNO);PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE));
    READ(CHAPCHDEL);OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL));
    READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
    READ(ESSENTFUNCT);PRINT(("MEMORY NEEDED FOR ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
```

```
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
    READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
    READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
    READ(TYPPERCENT[1]);PRINT(("% OF TYPE   1 ",TYPPERCENT[1],NEWLINE));
    'FOR'I'FROM'2'TO'JOBTYPNO'DO'
      (READ(TYPPERCENT[I]);
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
       TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1]);
    READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
    READ(SIZPERCENT[1]);
    READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
    'FOR'I'FROM'2'TO'JOBSIZNO'DO'
      (READ(SIZPERCENT[I]);
       READ(JOBSIZES[I]);
     PRINT((SIZPERCENT[I],"% OF JOBS ARE",JOBSIZES[I],"K",NEWLINE));
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1]);
    READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
    'FOR'I'TO'JOBTYPNO'DO'
    (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
      (READ(EVENTPROBS[I,J]);PRINT(EVENTPROBS[I,J])));
PRINT(NEWPAGE);
    'FOR'JOBNO'WHILE'OK1'AND'JOBLOAD#FREECORE'DO'
    (JOBSIZGEN;
        'IF'FREECORE-JOBLOAD>=JOBSIZE
        'THEN'JOB[JOBNO,1]:=JOBSIZE;
              JOB[JOBNO,2]:=JOBLOAD+1;
              JOB[JOBNO,3]:=JOBSIZE+JOBLOAD;
               JBBLOCKSTART[JOBNO]:=0.0;
              JOB[JOBNO,4]:=JOBTYPE;
           COREIMAGE'PLUS'1;
              PRINT(NEWLINE);
              JOBNEW:=JOBNO+1;
              JOBLOAD'PLUS'JOBSIZE
        'ELSE'          FRAGLEVEL:=FREECORE-JOBLOAD;
              JOBNEW:=JOBNO;
              FRAGCOUNT'PLUS'1;UNUSECORE:=FRAGLEVEL;
             GHEAD:=GTAIL:='GAPENTRY':=
             (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL');
             OK1:='FALSE';LASTJOBIN:='FALSE'
        'FI';
        'IF'JOBLOAD=FREECORE'THEN'GHEAD:=NULLUS;
                                 FRAGCOUNT'PLUS'1;
                                 UNUSECORE:=FRAGLEVEL
        'FI');
    'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
 'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JPBLOCKED[I]:=0.0;TSEXPD[I]:=0;IOREQ[I]:=0;
                   SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

  'FOR'I'TO'N'DO'(PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                  NEWJBOOL[I]:='FALSE';NOTENT[I]:='TRUE';
                  IO[I]:='FALSE';PROJQWAIT[I]:=0.0);
   'FOR'I'TO'70'DO'(NOTROLLEDOUT[I]:='FALSE';WAITBLOCKED[I]:='FALSE');
    IHEAD:=ITAIL:='NIL';
    QHEAD:='NIL';

  'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'
       PR[I]:=I;
    'IF'N>=JOBNEW-1
    'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;HEAD:=TAIL:='NIL'
    'ELSE'    HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
  'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'
   TAIL:='NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
   'FI');
```

```
'C'KEEPS A CHECK OF ALL GAPS IN CORE'C'
'PROC'GAPLIST=('INT'X,Y,Z)'VOID':
        (SIZ:=X;BEG:=Y;FIN:=Z;
     'IF'GHEAD'IS'NULLUS
    'THEN'GHEAD:=GTAIL:='GAPENTRY':=
            (SIZ,BEG,FIN,'NIL')
    'ELSE'GPTR3:=GHEAD;CONTINUE:='TRUE';OK1:='FALSE';
LOOPEND:'WHILE'CONTINUE'DO'
            ('IF'BEGIN'OF'GPTR3=FIN+1
             'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
                   BEGIN'OF'GPTR3:=BEG;
                   SIZ:=SIZE'OF'GPTR3;
                   BEG:=BEGIN'OF'GPTR3;
                   FIN:=END'OF'GPTR3;
                  'IF'OK1
                   'THEN'GPTR2:=GHEAD;GPTR1:='NIL';
                         'WHILE''IF'BEG=BEGIN'OF'GPTR2'AND'END'OF'GPTR2#FIN
                                 'THEN''IF'GPTR1'IS'NULLUS
                                        'THEN'GHEAD:=PTR'OF'GPTR2
                                        'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                        'FI'
                                  'FI';
                                  PTR'OF'GPTR2'ISNT'NULLUS
                              'DO'(GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
                   'FI';
                   OK1:='TRUE'
             'ELSF'END'OF'GPTR3=BEG-1
             'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
                   END'OF'GPTR3:=FIN;
                   SIZ:=SIZE'OF'GPTR3;
                   BEG:=BEGIN'OF'GPTR3;
                   FIN:=END'OF'GPTR3;
                   'IF'OK1
                         'THEN'GPTR2:=GHEAD;GPTR1:='NIL';
                               'WHILE''IF'FIN=END'OF'GPTR2'AND'BEGIN'OF'GPTR2#BEG
                                       'THEN''IF'GPTR1'IS'NULLUS
                                              'THEN'GHEAD:=PTR'OF'GPTR2
                                              'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                               'FI'
                                        'FI'
                                        PTR'OF'GPTR2'ISNT'NULLUS
                                    'DO'(GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
                               'FI';
                         OK1:='TRUE'
              'FI';
              'IF'PTR'OF'GPTR3'IS'NULLUS
              'THEN'CONTINUE:='FALSE';
                    'GOTO'LOOPEND
              'FI';
             GPTR3:=PTR'OF'GPTR3);
          'IF'OK1='FALSE'
          'THEN'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
                 (SIZ,BEG,FIN,'NIL')
          'FI'
    'FI' );

'C'CHECKS TO SEE IF ENOUGH SPACE IN CORE FOR A JOB TO BE ROLLED IN'C'

'PROC'SPACEINCORE=('INT'I)'BOOL':
        ('IF'GHEAD'ISNT'NULLUS
         'THEN'      GPTR1:='NIL';GPTR2:=GHEAD;
          CONTINUE:='TRUE';GAPFOUND:='FALSE';
          'WHILE'CONTINUE'AND'(PTR'OF'GPTR2'ISNT'NULLUS)'DO'
            'IF'SIZE'OF'GPTR2>=JOBSIZE
```

```
            'THEN'CONTINUE:='FALSE';GAPFOUND:='TRUE';
                  'IF'GPTR1'IS'NULLUS
                  'THEN'GHEAD:=PTR'OF'GPTR2
                  'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                  'FI'
            'ELSE'GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2
            'FI';
            'IF'PTR'OF'GPTR2'IS'NULLUS
            'THEN''IF'SIZE'OF'GPTR2>=JOBSIZE
                  'THEN'GAPFOUND:='TRUE';
                        'IF'GPTR1'IS'NULLUS
                        'THEN'GHEAD:=GTAIL:='NIL'
                        'ELSE'GTAIL:=GPTR1;PTR'OF'GTAIL:='NIL'
                        'FI'
                  'FI'
            'FI'
          'ELSE'GAPFOUND:='FALSE'
         'FI';
          PRTIME[I]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
           GAPFOUND);

'C'SUPERVISES SELECTION AND ROLLING IN OF NEW JOBS'C'

'PROC'NEWJOB=('INT'I)'VOID':
     (          OK:='TRUE';
          'WHILE' OK'DO'
            (PR[I]:=JOBNEW;JOBSIZGEN;
          'IF'SPACEINCORE(I)
               'THEN'FRAGLEVEL'MINUS'JOBSIZE;
              JOBLOAD'PLUS'JOBSIZE;
              JOB[JOBNEW,1]:=JOBSIZE;
              JOB[JOBNEW,2]:=BEGIN'OF'GPTR2  ;
              JOB[JOBNEW,3]:=JOBSIZE+BEGIN'OF'GPTR2-1;
              JOB[JOBNEW,4]:=JOBTYPE;
              'IF'JOB[JOBNEW,3]<END'OF'GPTR2
              'THEN''IF'GHEAD'IS'NULLUS
                    'THEN'GHEAD:=GTAIL:='GAPENTRY':=
                          (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1,
                           END'OF'GPTR2,'NIL')
                    'ELSE'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
                          (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1 ,
                           END'OF'GPTR2,'NIL')
                    'FI'
              'FI';
               ROLLIN(I); PRTIME[I]'PLUS'LLSDEL;
                  OSTIME[I]'PLUS'LLSDEL;
               COREIMAGE'PLUS'1;
              JOBNEW'PLUS'1
          'ELSE'  LASTJOBIN:='FALSE';OK:='FALSE'
         'FI'));

'C'CHECKS IF THERE ARE ANY INTERRUPTS THAT HAVE YET TO BE DEALT WITHA
     AFTER PROCESSOR HAS JUST DEALT WITH ONE'C'

'PROC'INTRUPCHECK=('INT'I)'VOID':
   ('IF'IHEAD'ISNT'NIL
    'THEN''IF'INTRUPTIME'OF'IHEAD<=PRTIME[I ]
          'THEN'PR[I]:=-1
          'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
          'FI'
    'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
    'FI');
'PROC'INTERUPT=('INT'I)'VOID':
```

```
        ('IF'NOTENT[I]'THEN'PRTIME[I]:=REALTIME
    'ELSE''IF'PRTIME[I ]>JBLISTIME
                          'THEN'JBLISTIME:=PRTIME[I ]
          'ELSE''IF'PR[I]#0'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I] 'FI';
                          PRTIME[I ]:=JBLISTIME
            'FI';
          NOTENT[I]:='TRUE';
          CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
          'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
          'DO'('IF'PRO'OF'QPTR2=I
               'THEN''IF'QPTR1'IS'NILL
                     'THEN'QHEAD:=QNEXT'OF'QHEAD
                     'ELSE'QNEXT'OF'QPTR1:=QNEXT'OF'QPTR2
                     'FI';
                     CONTINUE:='FALSE'
               'ELSE'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
               'FI');
               'IF'QNEXT'OF'QPTR2'IS'NILL
               'THEN''IF'PRO'OF'QPTR2=I
                     'THEN''IF'QPTR1'IS'NILL
                              'THEN'QHEAD:='NIL'
                              'ELSE'QNEXT'OF'QPTR1:='NIL'
                              'FI'
                     'FI'
               'FI'
     'FI';
     'IF'PR[I ]>0
     'THEN''IF'NEWJBOOL[I]
           'THEN'NEWJBOOL[I]:='FALSE'
           'ELSE'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
                       OSTIME[I]'PLUS'INSERTDEL
           'FI'
     'FI';
'IF'INTRUPTYP'OF'IHEAD
         'THEN''IF'IO'OF'IHEAD
          'THEN'PRTIME[I]'PLUS'SWAPODEL;OSTIME[I]'PLUS'SWAPODEL;
                JBLISTIME'PLUS'SWAPODEL;
                PR[]:=JOBNO'OF'IHEAD;
                 JBBLOCKED[PR[I]]'PLUS'PRTIME[I]-JBBLOCKSTART[PR[I]];
                WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
          'ELSE'PR[I]:=JOBNO'OF'IHEAD;
                 WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
          'FI';
       IHEAD:=NXT'OF'IHEAD
'ELSE'JOBNO:=JOBNO'OF'IHEAD;
     NOTROLLEDOUT[JOBNO]:='FALSE';
      FRAGLEVEL'PLUS'JOB[JOBNO,1];
      JOBLOAD'MINUS'JOB[JOBNO,1];
          COREIMAGE'MINUS'1;
      GAPLIST(JOB[JOBNO,1],JOB[JOBNO,2],JOB[JOBNO,3]);
     PRTIME[I ]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
     JOB[JOBNO,2]:=JOB[JOBNO,3]:=0;OSTIME[I]'PLUS'INTRUPDEL;
          PRTIME[I ]'PLUS'INTRUPDEL;
         NEWJOB(1);IHEAD:=NXT'OF'IHEAD
   'FI');

'C'WHEN A JOB TERMINATES THIS ROUTINE SETS UP THE ROLING OUT OF THAT JOB'C'

'PROC'ENDJOB=('INT'I)'VOID':
   (NOTROLLEDOUT[PR[I ]]:='TRUE';
    PRTIME[I ]'PLUS'CHANNELDEL;SYSTIME[PRI[I]]'PLUS'CHANNELDEL;
      OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='FALSE';     IOLIST(I));
```

```
'C'GENERATES NEXT EVENT TO OCCUR WITHIN A PARTIC. JOBTHAT IS RUNNING'C'

'PROC'INSTRGEN=('INT'I,JOBTYPE)'VOID':
  (PRTIME[I ]:=REALTIME;
   'IF'PR[I ]>0
   'THEN'NO:='ENTIER'(RAND2*100000);
         JBSIZE[PR[I] ]'PLUS'1.0;PRTIME[I]'PLUS'1.0;
         'IF'TOTAL:=EVENTPROBS[JOBTYPE,1];NO<TOTAL
         'THEN'TSEXPD[PR[I]]'PLUS'1;
               JQLIST(I)
         'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,2];NO<TOTAL
         'THEN'IOREQ[PR[I]]'PLUS'1;
               INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
         'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,3];NO<TOTAL
         'THEN'JOBCOUNT'PLUS'1;TERMINED[PR[I]]:='TRUE';
               NEWJBOOL[I]:='TRUE';PRTIME[I]'PLUS'ACCOUNTDEL;ENDJOB(I);JQLIST(I)
         'FI'
   'ELSF'PR[I ]=0'THEN'NEWJBOOL[I]:='TRUE';JQLIST(I)
   'FI');

'C'
     DETERMINES TIME OF NEXT EVENT THROUGHOUT THE SYSTEM'C'

'PROC'NXTEVENTIME='VOID':
   (TEMP:=SAMPLEVAR;
    'FOR'I'TO'N'DO'
       'IF'NOTENT[I]'AND'PR[I ]>0
       'THEN''IF'TEMP>PRTIME[I ]
             'THEN'TEMP:=PRTIME[I ]
             'FI'
       'FI';
    'IF'CHANNELTIME;TEMP>IOTIME
    'THEN'TEMP:=IOTIME
    'FI';

    'IF'TEMP>JBLISTIME
    'THEN''IF'HEAD'ISNT'NULL
          'THEN''IF'TIME'OF'HEAD<=JBLISTIME
                'THEN'TEMP:=JBLISTIME
                'FI'
          'FI';
         'FOR'I'TO'N'DO'
         'IF''NOT'NEWJBOOL[I]'AND''NOT'NOTENT[I]'THEN'TEMP:=JBLISTIME'FI'
    'FI';
    'IF'IHEAD'ISNT'NIL
    'THEN'
    'IF'TEMP>INTRUPTIME'OF'IHEAD
    'THEN'TEMP:=INTRUPTIME'OF'IHEAD
    'FI'
    'FI');


'C' **MAIN BODY**

     ORGANISES RUNNING OF JOBS,QUEUE AROUND CRITICAL REGION ,IRQ QUEUE,
     RESULTS TO BE PRINTED OUT DURING RUN,TIME-KEEPING,INTERRUPTS,RESULTS AT END OF RUN'C'

INIT;
'WHILE'JOBCOUNT# 12'DO'
  ('FOR'I'TO'N'DO'
   ('IF'PRTIME[I]<=REALTIME'AND'PR[I ]=0 'THEN'IDLETIME[I]'PLUS'1.0'FI';
    'IF'NOTENT[I]'AND'PR[I ]>=0'THEN'
```

```
        'IF'PRTIME[I ]<= REALTIME'THEN'INSTRGEN(I,JOB[PR[I ],4])'FI'
        'FI');
    'IF'QHEAD'ISNT'NILL
    'THEN''IF'REALTIME>=QTIME'OF'QHEAD'AND'REALTIME>=JBLISTIME
          'THEN'I:=PRO'OF'QHEAD;
                'IF'PRTIME[I]>JBLISTIME
                 'THEN'JBLISTIME:=PRTIME[I]
                 'ELSE''IF'PR[I]#0
                       'THEN'PROJQWAITE[I]'PLUS'JBLISTIME-PRTIME[I]
                        'FI';
                      PRTIME[I]:=JBLISTIME
                  'FI';
                NOTENT[I]:='TRUE';
                'IF'NEWJBOOL[I]
                'THEN'REALLOC(I);NEWJBOOL[I]:='FALSE'
                'ELSE'INSERT(I);REALLOC(I)
                'FI';
                'IF'QNEXT'OF'QHEAD'ISNT'NILL
                'THEN'QHEAD:=QNEXT'OF'QHEAD
                'ELSE'QHEAD:='NIL'
                'FI'
          'FI'
    'FI';

    'FOR'I'TO'CHANO'DO'
    ('IF'CHANNEL[I]<REALTIME
    'THEN'CHANNELIDLE[I]'PLUS'REALTIME-CHANNEL[I];
          CHANNEL[I]:=REALTIME
    'FI';
      'IF'IOHEAD'ISNT'NOTHING
      'THEN''IF'CHANNEL[I]<=REALTIME'AND'IOTIM'OF'IOHEAD<=REALTIME
            'THEN''IF'IO'OF'IOHEAD
                  'THEN'IOTIM'OF'IOHEAD:=REALTIME+IODEL;
                        INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                   INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                       CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                  'ELSE'IOTIM'OF'IOHEAD:=REALTIME+ROLLINDEL;
                        INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                   INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                       'IF''NOT'INTRUPTYPE'OF'IOHEAD
                        'THEN'JOBFINISH[JOB'OF'IOHEAD]:=IOTIM'OF'IOHEAD
                        'FI';

                       CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                  'FI'
             'FI'
      'FI');
     'IF'REALTIME>=SAMPLEVAR
     'THEN'SAMPLEVAR'PLUS'50.0 ;UNUSECORE'PLUS'FRAGLEVEL;FRAGCOUNT'PLUS'1;
           USERCORE'PLUS'JOBLOAD;USERCORECOUNT'PLUS'1;
            SYSCORE'PLUS'MEMSIZE-FREECORE;SYSCORECOUNT'PLUS'1
     'FI';
     REALTIME'PLUS'1.0;
    NXTEVENTIME;
    'IF'REALTIME<TEMP
    'THEN''FOR'I'TO'N'DO'
           'IF'PR[I ]=0
           'THEN'IDLETIME[I]'PLUS'TEMP-REALTIME
            'FI';
          REALTIME:=TEMP
    'FI';
    'IF'IHEAD'ISNT'NIL
    'THEN''IF'REALTIME>=INTRUPTIME'OF'IHEAD'AND'PRTIME[N ]<=REALTIME
          'THEN'INTERUPT(N);INTRUPCHECK(N)
         'FI'
    'FI');
```

```
      PRINT(NEWPAGE);
 2    AVGUNUSED:=UNUSECORE'/'FRAGCOUNT;USERCOREAVG:=USERCORE'/'USERCORECOUNT;
      SYSCOREAVG:=SYSCORE'/'SYSCORECOUNT;PERCENT:=AVGUNUSED*100'/'MEMSIZE;
 4    PRINT(NEWLINE);
      OUTF(STANDOUT,$"REALTIME IS -"<8.3>2L$,(REALTIME));
 6    PRINT(("PROCESSOR USAGE",NEWLINE,"---------------",NEWLINE));
      PRINT(("PRO   PROIDLETIME %TIME PRO IDLE WAIT IN JOB Q %TIME IN JOB Q    OSTIME     % TIME IN OS",NEWLINE));
 8    'FOR'I'TO'N'DO'
       (OUTF(STANDOUT,$<1> ,<8.3>8X,<3.1>3X,<8.3>6X,<2.1>6X,<8.3>4X,<2.1>L$,
 10          (I,IDLETIME[I],IDLETIME[I] *100/ REALTIME ,PROJQWAIT[I],
                  PROJQWAIT[I] *100/ REALTIME ,OSTIME[I],OSTIME[I]*100/REALTIME));
 12     PROIDLESUM'PLUS'IDLETIME[I];PROJQWAITSUM'PLUS'PROJQWAIT[I]);
      OUTF(STANDOUT,$"AVG % TIME PROS IDLE IS"<2.1>"%"L$,
 14       (PROIDLESUM *100/ N / REALTIME ));
      OUTF(STANDOUT,$"AVG % TIME PROS IN JOB Q IS"<2.1>"%"L$,
 16        (PROJQWAITSUM *100/ N / JBLISTIME ));
      PRINT(("CHANNEL USAGE",NEWLINE,"-------------",NEWLINE));
 18   PRINT(("CHANNEL TIME CHANNEL IDLE % TIME CHANNEL IDLE",NEWLINE));
      'FOR'I'TO'CHANO'DO'
 20    (OUTF(STANDOUT,$<1>6X,<8.3>8X,<3.1>"%"L$,(I,CHANNELIDLE[I],
       '       CHANNELIDLE[I] *100/ IOTIME ));
 22     CHANNELIDLESUM'PLUS'CHANNELIDLE[I]);
      OUTF(STANDOUT,$"AVG % TIME CHANNELS IDLE IS"<2.1>"%"L$,
 24       (CHANNELIDLESUM *100/ IOTIME  / CHANO));
  'FOR'I'TO'JOBNEW-1'DO'IOSUM'PLUS'IOREQ[I];
 26 OUTF(STANDOUT,$"AVG FILESTORE I/O PER SEC ="<3.1>L$,
       (IOSUM/REALTIME*1000));
 28   PRINT(("MEMORY USAGE",NEWLINE,"------------",NEWLINE));
      OUTF(STANDOUT,$
 30                  "AVG AMOUNT OF MEMORY UNUSED"<3>"K"L,
                     "AVG % OF MEMORY UNUSED"<2>"%"L ,
 32                  "AVG AMOUNT OF MEM USED FOR CORE IMAGES"<3>"K"L,
                      "AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS"<3>"K"L$,
 34       (AVGUNUSED,PERCENT,USERCOREAVG,SYSCOREAVG));
      PRINT(("JOB DESCRIPTIONS",NEWLINE,"----------------",NEWLINE));
 36   PRINT(("JOBNO JOBTYPE JOBTIME(MSECS) T/S'S    IOREQS    BLOCKTIME  SYSTEM TIME  TERMED",
  NEWLINE));
 38   'FOR'I'TO'JOBNEW-1'DO'
       (OUTF(STANDOUT, $<3>3X,<2>2X,<8.3>2X ,<4>4X ,<5> ,<8.3> ,<8.3>3X$,
 40         (I,JOB[I,4],JBSIZE[I],TSEXPD[I] ,IOREQ[I],JBBLOCKED[I],SYSTIME[I]));
      PRINT(TERMINED[I]);
        JBLOCKSUM'PLUS'JBBLOCKED[I] ;
       'IF'WAITBLOCKED[I]'AND'IOREQ[I]>0
 44    'THEN'JBLOCKSUM'PLUS'REALTIME-JBBLOCKSTART[I];
            PRINT(("   -STILL BLOCKED",NEWLINE))
 46    'ELSE'PRINT(NEWLINE)
       'FI');
 48 PRINT(NEWPAGE);
      PRINT(("JOBNO TIME JOB INITIATED   TIME JOB FINISHED     TIME IN SYSTEM",
 50         NEWLINE));
      'FOR'I'TO'JOBNEW-1'DO'
 52    'IF''NOT'('NOT'(JOBFINISH[I]<0.0)'AND''NOT'(0.0<JOBFINISH[I]))
       'THEN'OUTF(STANDOUT,$<3>3X,<8.3>8X,<8.3>7X,<8.3>L$,
 54       (I,JOBSTART[I],JOBFINISH[I],JOBFINISH[I]-JOBSTART[I]))
        'ELSE'OUTF(STANDOUT,$<3>3X,<8.3>3X"          --              --"L$,
 56             (I,JOBSTART[I]))
          'FI'
 58 'END'
    'FINISH'
 60
    ********************************************************************************************************************************
 62 ********                                                                                                                        67
    ********        NUMBER OF PAGES   12                                                                                            
 64 ********                                                                                                                        64
    ********************************************************************************************************************************
```

SKELETON DOCUMENTATION

CHANNEL TIME (1)

1. Checks all the times of the channels to find the lowest.

2. This is stored in the variable IOTIME and constitutes the time when the next channel will be free.

NOTE FOR USE

1. Only called by NXTEVENTIME (1). Therefore is unnecessary if NXTEVENTIME (1) not used.

INTRUPLIST (1)

1. Each entry on this list contains the time the interrupt will occur, the type of interrupt, whether generated after ordinary input/output request or because a job has been rolled in, and the number of the job it concerns. This information is passed as parameters.

2. If the list of interrupts is empty then the entry of an interrupt becomes the head of the list.

3. Otherwise searches down the list until the time of the new entry is less than an entry on the list. The new entry is then placed on the list before this entry.

4. If no entry on the list has a greater time than the new entry the new entry is placed at the end of the list.

NOTES FOR USE

1. Called from the main body after any input/output operation has been completed.

IOLIST (1)

1. Each entry on this list contains the time the entry would be placed on the list, the type of interrupt that will be generated after the input/output operation is completed, the type of input/output operation to be carried out, and the number of the job requesting this operation.

2. If the list is empty, the new entry becomes the head of the queue.

3. Otherwise the list is searched until an entry on it has a time which is greater than the new entry. The new entry is then placed before this on the list.

4. If no entry on the list has a greater time then the new entry is placed at the end of the queue.

INSERT (1)

1. Each entry on this list contains the number of the job and the time it is ready to run.

2. Adds on a delay for this operation onto the processor time, SYSTIME and OSTIME.

3. If this ready queue is empty, the new entry becomes the head of the queue.

4. Otherwise the list is searched until an entry on the list has a time which is greater than the new entry. The new entry is then placed on the list before this entry.

5. If no entry on the ready queue has a greater time, the new entry is placed at the end of the queue.

INOUT (1)

1. Adds on a delay associated with placing an entry on the queue for input/output channel resources to processor time, SYSTIME and OSTIME.

2. Records in JOBLOCKSTART the time when the job is blocked.

3. Sets the input/output operation to be performed to an ordinary input/output request

4. Sets the type of interrupt operation which will be required on completion of this operation to unblocking the job and swapping it from the blocked state to the ready queue (INTRUPTYPE:= 'TRUE').

5. Sets the flag WAITBLOCKED to signal that the job is blocked.

6. Calls IOLIST(1) to place an entry for this request onto the queue for input/output resources.

ROLLIN (1)

1. Makes a note of the time the job is initiated.

2. Adds on the delay associated with placing an entry on the queue for channel resources to the time of the processor, SYSTIME and OSTIME.

3. Sets the input/output operation to indicate rolling in of a job (IO:= 'FALSE').

4. Sets the type of interrupt operation to unblocking the job and placing on the ready queue (INTRUPTYPE:= 'TRUE').

5. Sets the flag WAITBLOCKED to indicate the job is blocked.

6. Calls IOLIST(1) to place an entry for this request onto the queue for input/output resources.

REALLOC (1)

1. Checks if ready queue is empty and if so makes processor idle (PR [I]: = 0).

2. If not, checks whether the time of the first entry on the list is greater than the time of the processor.

3. If this is the case the job is not ready to run at the time the processor entered the routine and so the processor is made idle.

4. If not, the processor is assigned that job by setting PR [I] equal to the number of the job.

5. A delay for this operation is added onto the processor's time.

6. If the processor is (still) idle the reallocation delay is added onto OSTIME.

7. JBLISTIME is set equal to the time at which the processor would leave the critical region.

NOTES FOR USE

1. Must be used in conjunction with INSERT (1) since REALLOC (1) relies on the list being time ordered, i.e. that the head of the list is the next job to be assigned processing time. If INSERT (1) is changed so that a different strategy is used instead of round robin, and this property (head of list always first to be assigned) no longer holds, REALLOC will require adjustment.

JOBSIZGEN (1)

1. If the last job generated by JOBSIZGEN (1) was used by the simulation a random number is generated and set equal to the variable NO.

2. Checks which range the number falls into and sets JOBTYPE accordingly.

3. Generates another random number in the range 1 - 100.

4. Checks which range the number falls into and sets JOBSIZE accordingly.

5. If the last job generated by JOBSIZGEN (1) was not used on the system then the specifications remain the same.

NOTES FOR USE

1. The distributions used in JOBSIZGEN (1) can be readily changed to depict differing workloads and to suit the figures from which the user has been working.

JQLIST (1)

1. Each entry in this list contains the number of the processor and the time it arrived at JQLIST (1).

2. If the head of the list is null then the new entry is placed at the head of the list.

3. Otherwise the list is searched until an entry on the list has a time which is greater than that of the new entry and the new entry is placed before this on the list.

4. If no entry is found to have a greater time then the new entry is placed at the end of the list.

5. Sets variable NOTENT (1) to show that processor is now an entry on the queue waiting to enter the critical region.

NOTES FOR USE

1. Entries in this queue are supervised through the critical region by the MAINBODY which constantly checks the time of the head of the list against the master clock.

2. Although a single processor system does not require such a queue, JQLIST (1) should still be included in the simulation model to preserve the structure of the model. Its inclusion does not simulate the processor waiting in this case but passes the processor straight through automatically and hence does not affect the results of the simulation study.

INIT (1)

1. Clears array JOB by setting all its entries to zero.

2. Reads in values for system and workload specifications.

3. If a job can be fitted in memory, this segment fills the free core with jobs generated by JOBSIZGEN noting their characteristics i.e. size, location of beginning of job, location of end of job and its type in array JOB.

4. Initialises JBBLOCKSTART to zero.

5. Notes number of core images on the system.

6. Increments JOBNEW - number of next job to be introduced to the system.

7. Increases JOBLOAD by the size of the job.

8. If the job generated is too large and cannot be fitted into memory the space left is assigned to FRAGLEVEL.

9. JOBNEW is set equal to the number of that last job.

10. A sample is taken of the fragmentation level.

11. The space is placed on the gap list.

12. LASTJOBIN is set to 'false' to indicate that the last job generated by JOBSIZGEN was not used.

13. If the memory is filled exactly the head of the gap list is set to null and a sample taken of the fragmentation (which will be zero).

14. All timings are set to 0.0 and all heads of lists set to null. All boolean variables are initialised.

15. The entries for the first jobs are assigned to processors.

16. Any other core images are placed on the ready queue.

NOTES FOR USE

1. INIT (1) must be used in conjunction with GAPLIST (1) SPACEINCORE (1) and NEWJOB (1) since all deal with the partitioned type of memory management scheme.

2. The state of the system from which the simulation starts can be altered by putting entries onto the lists, setting times to values other than zero and loading core such that fragmentation exists before the simulation starts.

GAPLIST (1)

1. The parameters passed to GAPLIST (1) include the size of the gap and the locations of its beginning and end in memory.

2. If the list is empty then the new entry is placed at the head of the list.

3. Else OK1, which signifies, whether the new gap joins with another on the list, is initialized to 'false'.

4. A check is made to see if the end of the new gap joins the beginning of the first gap on the list. i.e. if they are displaced by only one location. If so that entry is enlarged to include the new gap.

5. If OK1 is true, the new gap has already been found to join another gap on the list because its beginning joined the end of a gap on the list. Since this has been included into this entry the previous entry must be destroyed. The previous entry will have the same beginning but not the same end and a search for such an entry is made and the entry is then taken from the list when found.

6. OK1 is set to true to indicate that the new gap has joined with a gap on the list.

7. If the end of the new gap does not join the beginning of the one on the list a check is made to see if the beginning of the new gap joins the end of the one on the list. If so, that entry is enlarged to include the new entry.

8. If OK1 is true, the new gap has already joined with another on the list because its end joined with the beginning of one on the list. Since this has been included in this entry, the previous entry is destroyed. The ends of the two entries will be the same but not their beginnings.

9. OK1 is set to true to indicate that the new gap has joined with a gap on the list.

10. This process is continued for every entry on the list thus consolidating all gaps which join together.

11. If OK1 is 'false' i.e. the new gap did not join with any other gaps on the list, the new gap is placed as the last entry on the list.

NOTES FOR USE

1. GAPLIST (1) must be used in conjunction with INIT (1), NEWJOB (1) and SPACEINCORE (1) since they all deal with partitioned memory management schemes.

SPACEINCORE (1)

1. If the list is not empty, it is searched until a gap is found which is greater than or equal to the size of the job which is to be brought into memory.

2. This entry is removed from the list and a pointer left pointing to it.

3. GAPFOUND is set to 'true'.

4. If the list is empty or no gap is found GAPFOUND is set to 'false'.

5. A delay is added on to the time of the processor and OSTIME.

6. The value of GAPFOUND is passed as indication of whether a gap found or not.

NOTES FOR USE

1. SPACEINCORE (1) must be used in conjunction with INIT (1), NEWJOB (1) and GAPLIST (1), since they all deal with partitioned memory management schemes.

NEWJOB (1)

1. Generates a job by calling JOBSIZGEN.

2. Checks if there is space in memory to bring in a job by calling SPACEINCORE (1).

3. If there is, alters JOBLOAD and FRAGLEVEL accordingly.

4. Records size of job, its locations in memory and its type in array JOB; locations are determined by the pointer to the gap left by SPACEINCORE (1).

5. If the gap is larger than the job the remaining space is placed back onto the gap list.

6. The job is rolled in by ROLLIN.

7. Adds on a delay to the processor time and OSTIME, since the Low level scheduler would have been responsible for the choice of job to be brought in.

8. Increments number of coreimages.

9. Increments JOBNEW.

10. Continues this process until a job cannot be brought into memory where it sets LASTJOBIN to 'false' to indicate the last job generated by JOBSIZGEN was not used and OK to 'false' to stop the loop.

NOTES FOR USE

1. NEWJOB (1) must be used in conjunction with GAPLIST (1) INIT (1) SPACEINCORE (1) since they all deal with partitioned memory management schemes.

2. Relocation of core can be built into NEWJOB (1) (see NEWJOB (3)) but must be used under the same circumstances as NEWJOB (1).

INTRUPCHECK (1)

1. If the list is not empty, then if the time of the interrupt at the head of the list is less than or equal to the time of the processor, the processor is set to - 1 i.e. a null state. The processor will then be picked up by the main body and assigned to deal with the next interrupt.

2. Otherwise NEWJBOOL is set to show that the processor requires a new job and should be reallocated.

3. An entry for the processor is placed on the critical region queue by JQLIST.

NOTES FOR USE

1. This segment assumes that the same processor deals with all interrupts. If different interrupt algorithms are used this routine will be different. For example, if the processors deal with interrupts in turn then each processor would just be placed on the critical region queue to be reallocated with another job.

INTERUPT (1)

1. Sets the processor time equal to REALTIME if the processor is not on the critical region queue. (NOTENT [I] = 'TRUE')

2. Effectively interrupts that processor by checking various conditions and acting on these. The conditions are:

   a. If the processor is on the critical region queue its entry is taken from the queue and a note is made of its waiting time in the queue (PROJQWAIT), if it is not idle.

   b. If the processor requires a new job the segment changes the variable NEWJBOOL [I] to show it no longer needs a job.

   c. If it was already working on a job that job is inserted into the ready queue and the delays for this action added on to processor time and OSTIME.

   d. If the processor is idle PR [I] = 0 or null PR [I] = -1 (see INTRUPCHECK (1), nothing needs to be done.

3. Checks if interrupt requires swapping of an entry from the blocked state to the ready queue. i.e. input/output or the rolling in of a job has been completed.

4. If input/output has been completed delays are added for swapping the entries between the queues, a record is made of how long the job was blocked and en entry for the job is inserted into the ready queue.

5. If a job has been rolled in (IO = 'false') it is unblocked and placed onto the ready queue.

6. If the rolling out of a job is the cause of the 'interrupt', the boolean NOTROLLEDOUT is set to 'false' to show it has been completed. FRAGLEVEL and JOBLOAD are altered accordingly.

The number of coreimages is decremented. The gap is placed on the gap list. Delays are added on to processor time and OSTIME. NEWJOB is called to bring in another job.

7. The head of the list is moved onto the next entry.

ENDJOB (1)

1. The variable NOTROLLEDOUT is set to show that the job is in the process of being rolled out.

2. Delays for setting up an entry on the channel queue are added to the time of the processor and to OSTIME and SYSTIME.

3. The variable IO is set to 'false' to show it is not an ordinary input/output request.

4. The variable INTRUPTYPE is set to 'false' which indicates the type of interrupt generated after the rolling out is completed.

5. IOLIST is called to put an entry onto the channel queue.

INSTRGEN (1)

1. Sets processor time equal to REALTIME.

2. If the processor is not idle, the type of the job is used to determine the probabilities of events occurring to that job. The events are time-slice expired, input/output request, or job termination.

3. Generates a random number between 0-100,000 (100,000 was chosen because it is easy to work out the percentages of events - see section 6.2.2.)

4. The variable JBSIZE contains the mill-time used by the job and is incremented one time unit (or 1 millisec); as in the processor time.

5. If the random number falls in the range of time-slice expired a note is made of the number of time-slices used by that job and an entry for the processor is placed on the critical region queue where it will place an entry for the job on the ready queue and be reallocated with another.

6. If it falls in the range indicating input/output request, a note is made of the number of input/output requests made by the job. INOUT is called to block the job and initiate input/output. The processor is then put on the critical region queue with NEWJBOOL set to show it requires reallocation with another job.

7. If it falls in the range for job termination, a note is made of jobs which have terminated (JOBCOUNT). An accounting delay is added onto processor time. ENDJOB is called to roll out the job and an entry for the processor is placed on the critical region queue with NEWJBOOL set to show it requires reallocation with another job.

8. If it falls outside the above ranges it is assumed to be an ordinary instruction, see note 1.

9. If the processor is idle it is placed on the critical region queue with NEWJBOOL set to show it requires a new job.

NOTES FOR USE

1. The simulator only checks for events which require some operating system intervention. 1.0 millisecs is added onto the mill time because the simulator assumes that the instructions between such events require only arithmetic computation e.g. +, -, shift etc. The average time for such instructions to be processed is 0.002 millisecs and thus the simulator checks for an event requiring operating system intervention every 500 instructions. Although the results gained are slightly grosser than if a check was made every 0.002 millisecs, the simulation model is much faster. The model can be made to run faster by adding on a second or 10 millisecs and adjusting the probability of events accordingly, but the results become grosser.

NXTEVENTIME (1)

1. The variable TEMP is set equal to SAMPLEVAR.

2. Any processor which is not on the critical region queue and not idle is checked. If any one of their times is less than temp, temp is set equal to that time.

3. CHANNELTIME is called to calculate the time when the next channel will be ready to take up another operation and set this to IOTIME. TEMP is then tested against IOTIME and if IOTIME is smaller, TEMP is set equal to IOTIME.

4. The variable JBLISTIME is checked. This gives the time when the next processor can pass through the critical region. If this is less than TEMP and the time when the first job is ready to run is less than JBLISTIME then TEMP is set equal to JBLISTIME.

5. If any processor on the critical region queue wishes to insert a job entry onto the ready queue and JBLISTIME is less than TEMP then TEMP is set equal to JBLISTIME.

6. If the time the next interrupt is to occur is less than TEMP, TEMP is set equal to that time.

MAIN BODY

1. Calls segment INIT to set up the initial state of the simulator.

2. The MAIN BODY carries out the following loop until 12 jobs have been completed. (One loop per time unit).

3. If any processor is idle and is ready to run (PRTIME [I] <= REALTIME) its idle time is increased one time unit.

4. If any processor is not on the critical region queue and is executing a job, INSTRGEN is called to generate the next event for that job.

5. A check is made to see if there is a processor on the critical region queue which can now enter it - its time and JBLISTIME must be less than or equal to REALTIME. If there is its entry is removed from the list noting how long it waited on the list. The segments INSERT and REALLOC are called according to the boolean value in NEWJBOOL.

6. A check is made to see if there are any entries on the channel queue and if so, whether a channel is free to deal with them. If the time of the channel <= REALTIME then it has been stood idle and its idle-time is noted in CHANNELIDLE. If the time of the channel and the time of the entry on the queue are <= REALTIME, the channel can deal with the request. Delays are added onto to the time of the channel and the entry according to the type of input/output operation to be carried out. An interrupt is generated at the time the operation would have beencompleted and the entry is removed from the queue.

7. This next section of the MAIN BODY is used to take performance measurement snapshots during the simulation. The variable SAMPLEVAR, is used to take the measurements at regular intervals, in this case 50 millisecs. The amount of fragmentation is totalled, and FRAGCOUNT incremented; the amount of core being used by core images is totalled and USERCORECOUNT incremented; the amount of core used by the operating system is totalled and SYSCORECOUNT incremented.

8. One time unit is added to the master clock.

9. NXTEVENTIME is called to determine the time of the next event. If the time is greater than REALTIME, REALTIME is set to this and any processors which are idle have their idletimes increased.

10. A check is made for any interrupts that would occur at that time and if there are segment INTERRUPT is called to deal with them.

11. Prints out in tabular form the results of the simulation run after 12 jobs have terminated.

NOTES FOR USE

1. When taking the performance snapshots using SAMPLEVAR, far more information can be recorded and output if required. For example:

   a. Length of input/output and critical region queues (see IOQPROBE (1) and JQSIZPROBE (1).

   b. Number of coreimages.

   c. Number of input/output requests since last sample snapshot.

   d. Percentage of time processor(s) idle since last sample etc.

Appendix 8

LIBRARY DOCUMENTATION

INIT (2)

1. Reads in values for system plus workload specifications.

2. Loads memory segments with jobs generated by JOBSIZGEN, recording relevant information in array JOB.

3. Alters the fragmentation level and the job load accordingly.

4. Increments JOBNEW every time a job is placed in a memory segment, so each job has a unique number/name.

5. Initializes all heads of lists to null and timing variables to zero.

6. Assign processors to jobs, in memory:

   if numbers of processors is less than number of jobs in memory then record other jobs onto ready queue.
   else if number of jobs in memory is less than number of processors set other processors to idle (PR [I]: = 0).

NOTES FOR USE

1. Must be used in conjunction with NEWJOB (2) GAPLIST (2) SPACEINCORE (2), designed for segmented memory management.

```
MOP        :SSP8086.MOPPMB

#LISTING OF :SSP8086.INIT2(1/)  PRODUCED ON 23AUG78 AT 14.22.10

#GR.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.07.37 USING  U15

DOCUMENT    INIT2


'C'INITIALISES SIMULATION VARIABLES AND STATEFROM WHICH SIMULATION BEGINS'C'

'PROC'INIT='VOID':
 (OK1:='TRUE';
   'FOR'I'FROM'0'TO'70'DO'
      'FOR'J'TO'4'DO'JOB[I,J]:=0;

READ(TITLE);PRINT((TITLE,NEWLINE,"---------------------",NEWLINE));
PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE));
     READ(FREECORE);PRINT(("FREE CORE =",FREECORE,NEWLINE));
     READ(MEMSIZE);PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE));
     READ(N);PRINT(("NO OF PROCESSORS =",N,NEWLINE));
     READ(CHANO);PRINT(("NO OF CHANNELS =",CHANO,NEWLINE));
     READ(IODEL);OUTF(STANDOUT,$"I!O DELAY ="<3.1>L$,(IODEL));
     READ(REALLOCDEL);OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL));
     READ(INSERTDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL));
     READ(ROLLOUTDEL);OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL));
     READ(ROLLINDEL);OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL));
     READ(RELOCDEL);OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL));
     READ(CHANNELDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL));
     READ(GAPCHECKDEL);OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL));
     READ(INTRUPDEL);OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL));
     READ(SWAPQDEL);OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAPQDEL));
     READ(HLSDEL);OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL));
     READ(LLSDEL);OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL));
     READ(ACCOUNTDEL);OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL));
     READ(SEGSIZE);PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE));
     READ(SEGNO);PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE));
     READ(CHAPCHDEL);OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL));
     READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
     READ(ESSENTFUNCT);PRINT(("MEMORY NEEDED FOP ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
     READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
     READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
     READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
     READ(TYPPERCENT[1]);PRINT(("% OF TYPE   1 ",TYPPERCENT[1],NEWLINE));
     'FOR'I'FROM'2'TO'JOBTYPNO'DO'
       (READ(TYPPERCENT[I]);
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
        TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1]);
     READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
     READ(SIZPERCENT[1]);
     READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
     'FOR'I'FROM'2'TO'JOBSIZNO'DO'
       (READ(SIZPERCENT[I]);
```

```
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1]);
      READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
      'FOR'I'TO'JOBTYPNO'DO'
      (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
       (READ(EVENTPROBS[I,J]);PRINT(EVENTPROBS[I,J])));
PRINT(NEWPAGE);
            'FOR'JOBNO'TO'SEGNO'DO'
      (JOBSIZGEN;
           JOB[JOBNO,1]:=JOBSIZE;
                JOB[JOBNO,2]:=JOBNO;
                JOB[JOBNO,3]:=SEGSIZE-JOBSIZE;
                 JBBLOCKSTART[JOBNO]:=0.0;
                JOB[JOBNO,4]:=JOBTYPE;
               FRAGLEVEL'PLUS'SEGSIZE-JOBSIZE;

             COREIMAGE'PLUS'1;
                PRINT(NEWLINE);
                JOBNEW:=JOBNO+1;
                JOBLOAD'PLUS'JOBSIZE);
        FRAGCOUNT'PLUS'1;UNUSECORE'PLUS'FRAGLEVEL;
         FREECORE:=SEGNO*SEGSIZE;
           'FOR'I'TO'SEGNO'DO'SEGEMPTY[I]:='FALSE';
     'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
   'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JBBLOCKED[I]:=0.0;TSEXPD[I]:=0;IOREQ[I]:=0;
                       SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

    'FOR'I'TO'N'DO'(PR[I ]:=I;PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                     NEWJBOOL[I]:='FALSE';NOTENT[I]:='TRUE';
                     IO[I]:='FALSE';PROJGWAIT[I]:=0.0);
      'FOR'I'TO'70'DO'(NOTROLLEDOUT[I]:='FALSE';WAITBLOCKED[I]:='FALSE');
     IHEAD:=ITAIL:='NIL';
     GHEAD:='NIL';
      'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'
       PR[I]:=I;
      'IF'N>=JOBNEW-1
      'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;HEAD:=TAIL:='NIL'
      'ELSE'HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
           'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'
             TAIL:=NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
      'FI');
```

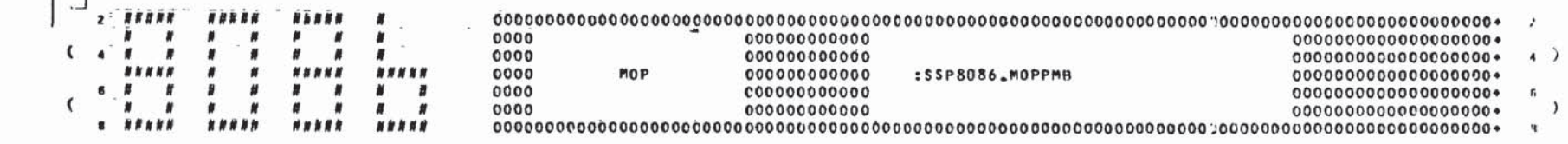
```
+DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD+
+DDDDDDD                                DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD+
+DDDDDDD        NUMBER OF PAGES    2    DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD+
+DDDDDDD                                DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD+
+DDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDDD+
```

GAPLIST (2)

1. This segment is called when a memory segment becomes free. The number of the segment is passed on as a parameter. A boolean array SEGEMPTY alters its corresponding elemement to give a 'true' value.

NOTES FOR USE

1. This segment differs from the skeleton GAPLIST (1) in that the number of the memory segment which is free is passed on as a parameter, not the size, beginning and end of the gap. Thus the call to GAPLIST (2) from INTERRUPT (1) must be altered. Must be used in conjunction with the segments INIT (2) SPACEINCORE (2), NEWJOB (2) designed for segmented memory management.



#LISTING OF :SSP8086_GAPLIST2(1/) PRODUCED ON 18JUL78 AT 15.22.08

#G8.63B AT ASTON IN ':SSP8086_MOPPMB' ON 18MAY79 AT 12.55.28 USING U15

DOCUMENT GAPLIST2

'C'KEEPS A CHECK ON ALL GAPS IN CORE'C'

```
'PROC'GAPLIST=('INT'SEG)'VOID':
   (SEGEMPTY[SEG]:='TRUE');
```

NUMBER OF PAGES 1

SPACEINCORE (2)

1. Sets boolean GAPFOUND to 'false'.

2. Searches down array SEGEMPTY for an element which gives a 'true' value i.e. the segment with that element number is free.

3. If one is found GAPFOUND is set to 'true'; the corresponding element in SEGEMPTY is now set to 'false'.

4. A delay is added to the processor time and the time it spent in the operating system for this operation.

NOTES FOR USE

1. As the algorithm for determining a free segment is simpler than determining a gap as in the partitioned memory scheme, the delay for this operation will be less than in the partioned memory SPACEINCORE (1).

2. Must be used in conjunction with GAPLIST (2), NEWJOB (2), INIT (2) designed for segmented memory management.

```
#LISTING OF :SSP8086.SPACEINCORE2(1/)   PRODUCED ON 18JUL78 AT 15.22.36

#68.63B AT ASTON    IN '=SSP8086.MOPPMB' ON 18MAY79 AT 12.55.23 USING  U15

DOCUMENT    SPACEINCORE2



'C'CHECKS TO SEE IF ENOUGH SPACE IN CORE FOR A JOB TO BE ROLLED IN'C'

 'PROC'SPACEINCORE=('INT'I)'BOOL':
   (GAPFOUND:='FALSE';'FOR'I'TO'SEGNO'DO'
         'IF'SEGEMPTY[I]
         'THEN'SEG:=I;GAPFOUND:='TRUE' -
         'FI';
         SEGEMPTY[SEG]:='FALSE' ;
        PRTIME[I]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
        GAPFOUND);
```

NEWJOB (2)

1. Checks if there is a segment free in memory; calls SPACEINCORE.
2. If there is, calls JOBSIZGEN to generate a new job.
3. Assigns entries to array JOB
    1. Job size
    2. Segment which it will occupy
    3. Segment space wasted
    4. Job type.
4. The fragmentation level and the jobload are altered according to the size of the job.
5. ROLLIN is called.
6. Adds on LLS delay as the low level scheduler would have been responsible for the choice of job to be brought in.
7. Increments JOBNEW - number of next job to be brought into system.
8. Continues this process until a job cannot be brought in.

NOTES FOR USE

1. Must be used in conjunction with INIT (2), GAPLIST (2), SPACEINCORE (2) designed for segmented memory management.

```
PPPP        MOP         PPPPPPPPPPPP    :SSP8086.MOPPMB

#LISTING OF :SSP8086.NEWJOB2(1/)    PRODUCED ON 18JUL78 AT 15.23.27

#GR.638 AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.55.34 USING  U15

DOCUMENT     NEWJOB2




'C'SUPER VISES SELECTION AND ROLLING IN OF NEW JOBS'C'

'PROC'NEWJOB=('INT'I)'VOID':
      (          OK:='TRUE';
             'WHILE' OK'DO'
               (PR[I]:=JOBNEW;
            'IF'SPACEINCORE(I)
                'THEN'PRINT((FRAGLEVEL, JOBLOAD,COREIMAGE,NEWLINE));
              JOBSIZGEN;
                JOBLOAD'PLUS'JOBSIZE;
                JOB[JOBNEW,1]:=JOBSIZE;
                JOB[JOBNEW,2]:=SEG  ;
                JOB[JOBNEW,3]:=SEGSIZE-JOBSIZE;
                JOB[JOBNEW,4]:=JOBTYPE;
                FRAGLEVEL'MINUS'JOBSIZE;
                 ROLLIN(I); PRTIME[I]'PLUS'LLSDEL;
                        OSTIME[I]'PLUS'LLSDEL;
                 COREIMAGE'PLUS'1;
                PRINT(("JOB",PR[I ],"IS BEING   ROLLED IN"));
                PRINT((JOBSIZE,"K"));
                PRINT(NEWLINE);
               PRINT((FRAGLEVEL,JOBLOAD,COREIMAGE,NEWLINE));
                JOBNEW'PLUS'1
             'ELSE' OK:='FALSE'
           'FI'));

NUMBER OF PAGES    1
```

NEWJOB (3)

1. Checks if number of tentatively started jobs is greater than zero.

2. If so, it generates a job by calling JOBSIZGEN.

3. Checks if there is space in memory to bring in the job by calling SPACEINCORE.

4. If there is, JOBLOAD and FRAGLEVEL are altered accordingly.

5. Records the size of job, its locations in memory and its type in array JOB: N.B. the locations are determined by the pointer to the gap left by SPACEINCORE.

6. If the gap is larger than the job the remaining space is placed back onto the gap list.

7. The job is then rolled in by ROLLIN.

8. Decrements number of tentatively started jobs.

9. Increments the number of core images.

10. Adds on a delay for low level scheduler which would have carried out this operation.

11. Increments JOBNEW, which will be the number of the next job.

12. If there is not enough space in memory (SPACEINCORE delivers a 'false' value), checks if the fragmentation level is greater than the job load.

13. If this is the case, segment RELOC is called to relocate memory. The number of times relocation took place is incremented.

14. The fragmentation level and job load are altered accordingly.

15. The job characteristics are recorded in array JOB as above.

16. A check is made that the relocation has taken place correctly (- this assists the user who wishes to alter the segment RELOC).

17. Any space which is still left in memory is recorded in the list of gaps.

18. The job is rolled in by ROLLIN. Decrements TENTSTART, increments COREIMAGE.

19. A delay for the low-level scheduler is added onto the time of the processor and the time it has spent in the operating system since it would have been responsible for the above operations.

20. Increments JOBNEW.

21. Continues this process until a job cannot be brought in, at which point LASTJOBIN is set to false to show that the last job generated by JOBSIZGEN was not used.

22. FREECOREADJ is called to alter FREECORE if necessary.

NOTES FOR USE

1. This segment recognises tentatively started jobs and calls FREECOREADJ, i.e. it is a segment developed for use in the George 3 operating system.
If a similar segment is required for the skeleton structure without these facilities see NEWJOB (4).

```
QQQQ        MOP          QQQQQQQQQQQQ    :SSP8086.MOPPMB

#LISTING OF :SSP8086.NEWJOB3(1/)    PRODUCED ON 18JUL78 AT 15.23.50

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.55.46 USING  U15

DOCUMENT    NEWJOB3


'C'INITIATES SELECTION OF AND ROLLING IN OF NEW JOBS'C'

'PROC'NEWJOB=('INT'I)'VOID':
    (          OK:='TRUE';
           'WHILE'TENTSTART#0'AND'OK'DO'
             (PR[I]:=JOBNEW;JOBSIZGEN;
          'IF'SPACEINCORE(I)'AND'TENTSTART>0
             'THEN'FRAGLEVEL'MINUS'JOBSIZE;
              JOBLOAD'PLUS'JOBSIZE;
              JOB[JOBNEW,1]:=JOBSIZE;
              JOB[JOBNEW,2]:=BEGIN'OF'GPTR2  ;
              JOB[JOBNEW,3]:=JOBSIZE+BEGIN'OF'GPTR2-1;
              JOB[JOBNEW,4]:=JOBTYPE;
              'IF'JOB[JOBNEW,5]<END'OF'GPTR2
              'THEN''IF'GHEAD'IS'NULLUS
                    'THEN'GHEAD:=GTAIL:='GAPENTRY':=
                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1,
                            END'OF'GPTR2,'NIL')
                    'ELSE'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1 ,
                            END'OF'GPTR2,'NIL')
                    'FI'
              'FI';
               ROLLIN(I);TENTSTART'MINUS'1;COREIMAGE'PLUS'1;PRTIME[I]'PLUS'LLSDEL;
                      OSTIME[I]'PLUS'LLSDEL;
              PRINT(("JOB",PR[I ],"IS BEING   ROLLED IN"));
              PRINT((JOBSIZE,"K"));
              PRINT(NEWLINE);
              JOBNEW'PLUS'1
         'ELSF'FRAGLEVEL>=JOBLOAD'AND'TENTSTART>0
        'THEN'RELOC(I);PRINT(("RELOCATION NECESSARY",NEWLINE));
             RELOCNO'PLUS'1;
              FRAGLEVEL'MINUS'JOBSIZE;
              JOBLOAD'PLUS'JOBSIZE;
              JOB[JOBNEW,1]:=JOBSIZE;
              JOB[JOBNEW,2]:=JOBLOAD+1-JOBSIZE;
              JOB[JOBNEW,3]:=JOBLOAD;
              JOB[JOBNEW,4]:=JOBTYPE;
              'IF'FRAGLEVEL#FREECORE-JOBLOAD
              'THEN'FRAGLEVEL:=FREECORE-JOBLOAD;
                    PRINT("MISTAKE IN GAPTABLE")
              'FI';
               'IF'FRAGLEVEL>0
               'THEN'              GHEAD:=GTAIL:='GAPENTRY':=
              (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL')
              'ELSE'GHEAD:=GTAIL:='NIL'
               'FI';
```

```
                ROLLIN(I);COREIMAGE'PLUS'1;TENTSTART'MINUS'1;PRTIME[I]'PLUS'LLSDEL;
                         OSTIME[I]'PLUS'LLSDEL;
                PRINT(("JOB",PR[I ],"IS BEING   ROLLED IN"));
                PRINT((JOBSIZE,"K"));
                PRINT(NEWLINE);
                JOBNEW'PLUS'1
              'ELSE'LASTJOBIN:='FALSE';OK:='FALSE'
            'FI');
       FREECORE(ADJ);

+QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ                                 QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ        NUMBER OF PAGES    2     QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ                                 QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
```

RELOC (1)

1. Check if any jobs are on the ready queue. If there are place these in memory, starting from the bottom, For each job, alter its entry in array JOB which holds the information concerning the jobs position in memory.

2. Any jobs currently assigned to processors are then relocated and array JOB altered accordingly.

3. Any jobs which are blocked are dealt with in a similar manner.

4. Increase channel times by relocation delay.

5. Any interrupts which are to occur are set back by the delay.

6. The delay is added to the time of the processor carrying out relocation and the time it spends in the operating system.

7. The delay is added to the master clock REALTIME and from this all other timing variables i.e. JBLISTIME and processor times are altered.

NOTES FOR USE

1. Called from NEWJOB (3) (see listing of NEWJOB (3) in library which includes relocation).

```
#####  #####  #####  #       RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR+
#   #  #   #  #   #  #       RRRR                    RRRRRRRRRRRR                                      RRRRRRRRRRRRRRRRRRRR+
#   #  #   #  #   #  #       RRRR                    RRRRRRRRRRRR                                      RRRRRRRRRRRRRRRRRRRR+
#####  #   #  #####  #####   RRRR         MOP        RRRRRRRRRRRR    :SSP8086.MOPPMB                   RRRRRRRRRRRRRRRRRRRR+
#   #  #   #  #   #  #   #   RRRR                    RRRRRRRRRRRR                                      RRRRRRRRRRRRRRRRRRRR+
#   #  #   #  #   #  #   #   RRRR                    RRRRRRRRRRRR                                      RRRRRRRRRRRRRRRRRRRR+
#####  #####  #####  #####   RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR+

#LISTING OF :SSP8086.RELOC1(1/) PRODUCED ON 18JUL78 AT 15.24.13

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 1MAY79 AT 12.55.53 USING  U15

DOCUMENT    RELOC1


'C'RELOCATES JOBS IN COR'C'


'PROC'RELOC=('INT'I)'VOID':
  (CONTINUE:='FALSE';PTR1:=HEAD;JOBLOAD:=0;
   'IF'PTR1'ISNT'NULL'THEN'
   'WHILE'NEXT'OF'PTR1'ISNT'NULL'DO'
     (JOBNO:=P'OF'PTR1;
      JOB[JOBNO,2]:=JOBLOAD+1;
      JOBLOAD'PLUS'JOB[JOBNO,1];
      JOB[JOBNO,3]:=JOBLOAD;
      PTR1:=NEXT'OF'PTR1);
   'IF'NEXT'OF'PTR1'IS'NULL
   'THEN'JOBNO:=P'OF'PTR1;
         JOB[JOBNO,2]:=JOBLOAD+1;
         JOBLOAD'PLUS'JOB[JOBNO,1];
         JOB[JOBNO,3]:=JOBLOAD
   'FI'
   'FI';
   'FOR'I'TO'N-1'DO'
     ('IF'PR[I ]#0'AND''NOT'NEWJBOOL[I]
      'THEN'JOBNO:=PR[I ];
            JOB[JOBNO,2]:=JOBLOAD+1;
             JOBLOAD'PLUS'JOB[JOBNO,1];
            JOB[JOBNO,3]:=JOBLOAD
      'FI');
   'FOR'JOBNO'TO'70'DO'
      'IF'NOTPOLLEDOUT[JOBNO]'OR'WAITBLOCKED[JOBNO]
      'THEN'JOB[JOBNO,2]:=JOBLOAD+1;
             JOBLOAD'PLUS'JOB[JOBNO,1];
             JOB[JOBNO,3]:=JOBLOAD
      'FI';
    'FOR'CHA'TO'CHANO'DO'
       CHANNEL[CHA]'PLUS'RELOCDEL;
    'WHILE'INTRUPTIME'OF'IPTR2'PLUS'RELOCDEL;
            NXT'OF'IPTR2'ISNT'NIL'DO'
              IPTR2:=NXT'OF'IPTR2;
   PRTIME[I ]'PLUS'RELOCDEL;OSTIME[I]'PLUS'RELOCDEL;
   REALTIME'PLUS'RELOCDEL;
   'IF'JBLISTIME<REALTIME
   'THEN'JBLISTIME:=REALTIME
   'FI';
       'FOR'I'TO'N'DO'
     ('IF'PRTIME[I ]<REALTIME
      'THEN'PRTIME[I ]:=REALTIME
      'FI'));
```

JQSIZEPROBE (1)

1. If head of list is null then queue is empty.

2. If not, count number of entries in the list whose times are less than or equal to the masterclock, REALTIME i.e. which would be on the queue at time of sample.

3. Print out size of queue.

NOTES FOR USE

1. For use in sample snapshots.

```
#####  #####  #####  #     VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
#   #  #   #  #   #  #     VVVV                VVVVVVVVVVVV                                   VVVVVVVVVVVVVVVVVVVV*
#   #  #   #  #   #  #     VVVV                VVVVVVVVVVVV                                   VVVVVVVVVVVVVVVVVVVV*
#####  #   #  #####  ##### VVVV       MOP      VVVVVVVVVVVV    :SSP8086.MOPPMB                VVVVVVVVVVVVVVVVVVVV*
#   #  #   #  #   #  #   # VVVV                VVVVVVVVVVVV                                   VVVVVVVVVVVVVVVVVVVV*
#   #  #   #  #   #  #   # VVVV                VVVVVVVVVVVV                                   VVVVVVVVVVVVVVVVVVVV*
#####  #####  #####  ##### VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*

#LISTING OF :SSP8086.JQSIZPROBE1(1/)    PRODUCED ON 18JUL78 AT 15.24.47

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.57.50 USING  U15

DOCUMENT    JQSIZPROBE1


'C'NOTES SIZE OF QUEUE WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'

'PROC'JQSIZEPROBE='VOID':
  ('IF'QHEAD'IS'NILL
   'THEN'PRINT((NEWLINE,"JQSIZE IS",JQSIZE,NEWLINE))
   'ELSE'    QPTR2:=QHEAD;
   'WHILE''IF'QTIME'OF'QPTR2<=REALTIME'THEN'JQSIZE'PLUS'1'FI';QNEXT'OF'QPTR2'ISNT'NILL'DO'
      QPTR2:=QNEXT'OF'QPTR2 ;
   PRINT((NEWLINE,"JQSIZE IS",JQSIZE ,NEWLINE));
   PRINT(NEWLINE);
   JQSIZE:=0
   'FI');


*VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
*VVVVVVV                       VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
*VVVVVVV    NUMBER OF PAGES  1 VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
*VVVVVVV                       VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
*VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV*
```

IOQPROBE (1)

1. If the head of the list is null then the list is empty.

2. If not, count the number of entries in the list whose times are less than or equal to REALTIME i.e. which would be on the queue at the time of the sample.

3. Print out size of queue.

NOTES FOR USE

1. For use in sample snapshots.

MOP :SSP8086.MOPPMB

```
#LISTING OF :SSP8086.IOQPROBE1(1/)  PRODUCED ON 18JUL78 AT 15.25.10

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.57.54 USING  U15

DOCUMENT     IOQPROBE1


'C'
    NOTES SIZE OF ISO REQUEST QUEUE'C'

'PROC'IOQPROBE='VOID':
       ('IF'IOHEAD'IS'NOTHING
      'THEN'PRINT((NEWLINE,"IQSIZE IS",IOQSIZE,NEWLINE))
       'ELSE'IOPTR2:=IOHEAD;
     'WHILE''IF'IOTIM'OF'IOPTR2<=REALTIME'THEN'IOQSIZE'PLUS'1'FI';IONXT'OF'IOPTR2'ISNT'NOTHING'DO'
        IOPTR2:=IONXT'OF'IOPTR2 ;
       PRINT((NEWLINE,"IQSIZE IS",IOQSIZE ,NEWLINE));
       PRINT(NEWLINE);
       IOQSIZE:=0
      'FI');
```

NUMBER OF PAGES 1

HLS (1)

1. Adds on delay for high-level scheduler to processor time and OSTIME.

2. Totals user's max size specifications (assumed to be 40K for simplicity) of all tentatively started jobs.

3. If this total is less than 200K and there are some jobs in the well a job is tentatively started.

NOTES FOR USE

1. This segment was developed for the George 3 simulation.

```
MOP    :SSP8086.MOPPMB

#LISTING OF :SSP8086.HLS1(1/)   PRODUCED ON 25SEP78 AT 15.02.47

#G8.638 AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.19.24 USING  U15

DOCUMENT    HLS1

'C'DECIDES WHETHER OR NOT TO TENTATIVELY START A JOB'C'

'PROC'HLS=('INT'I)'VOID':
   (PRTIME[I]'PLUS'HLSDEL;OSTIME[I]'PLUS'HLSDEL;
    HLSVAR:=TENTSTART*40;
     'IF'HLSVAR<200'AND'JOBSINWELL>0
     'THEN'TENTSTART'PLUS'1;JOBSINWELL'MINUS'1
     'FI');

NUMBER OF PAGES    1
```

FREECOREADJ (1)

1. The variable SAVE is set to what FREECORE should be.

2. If FREECORE is less than SAVE then a gap in memory is created to allow that extra core to be freed to users.

3. If FREECORE is greater than SAVE a check is made down the list of gaps for one at the top of the user's core area.

4. If this gap is greater than the amount needed for adjusting FREECORE the end of the gap is altered so that the remainder stays on the list and FREECORE is adjusted.

5. If this gap is smaller than the amount needed its entry is removed from the list and FREECORE is brought as near as possible to SAVE.

NOTES FOR USE

1. Called from segment NEWJOB (3).

2. Although this segment does not always give FREECORE a value that is ideal according to the formulas given in the George 3 manual (see segments CHAPQUOTA and REDTAPEQUOTA) the difference is negligible. Also the actual system itself flunctuates since these formulas only act as guidelines.

```
#####  #####  #####  #        XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
#   #  #   #  #   #  #        XXXX                    XXXXXXXXXXXX                                         XXXXXXXXXXXXXXXXXX*
#   #  #   #  #   #  #        XXXX                    XXXXXXXXXXXX                                         XXXXXXXXXXXXXXXXXX*
#####  #   #  #####  #####    XXXX        MOP         XXXXXXXXXXXX    :SSP8086.MOPPMB                      XXXXXXXXXXXXXXXXXX*
#   #  #   #  #   #  #   #    XXXX                    XXXXXXXXXXXX                                         XXXXXXXXXXXXXXXXXX*
#   #  #   #  #   #  #   #    XXXX                    XXXXXXXXXXXX                                         XXXXXXXXXXXXXXXXXX*
#####  #####  #####  #####    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*

#LISTING OF :SSP8086.FREECOREADJ1(1/)   PRODUCED ON 18JUL78 AT 15.26.01

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.57.58 USING  U15

DOCUMENT    FREECOREADJ1


'C'DETERMINES THE AMOUNT OF CORE LEFT FOR CORE IMAGESOF JOBS'C'

'PROC'FREECOREADJ='VOID':
    (SAVE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
    'IF'SAVE>FREECORE
    'THEN'FRAGLEVEL'PLUS'SAVE-FREECORE;
            GAPLIST(SAVE-FREECORE,FREECORE+1,SAVE);FREECORE:=SAVE
    'ELSF'SAVE<FREECORE
    'THEN''IF'GHEAD'ISNT'NULLUS
           'THEN'GPTR2:=GHEAD;GPTR1:='NIL';CONTINUE:='TRUE';
                  'WHILE''IF'END'OF'GPTR2=FREECORE

                          'THEN'CONTINUE:='FALSE';
                                 'IF'BEGIN'OF'GPTR2<SAVE
                                 'THEN'END'OF'GPTR2:=SAVE;
                             SIZE'OF'GPTR2'MINUS'FREECOPE-SAVE;
                                        FRAGLEVEL'MINUS'FREECORE-SAVE
                                       ;FREECORE:=SAVE
                                'ELSE'SAVE:=BEGIN'OF'GPTR2;
                                         FRAGLEVEL'MINUS'FREECORE-BEGIN'OF'GPTR2;
                                        FREECORE:=SAVE;
                                        'IF'GPTR1'IS'NULLUS
                                        'THEN''IF'PTR'OF'GPTR2'IS'NULLUS
                                               'THEN'GHEAD:='NIL'
                                               'ELSE'GHEAD:=PTR'OF'GHEAD
                                               'FI'
                                        'ELSE''IF'PTR'OF'GPTR2'IS'NULLUS
                                               'THEN'PTR'OF'GPTR1:='NIL';GTAIL:=GPTR1
                                                'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                                'FI'
                                         'FI'
                                 'FI'
                          'FI';
                          CONTINUE'AND'(PTR'OF'GPTR2'ISNT'NULLUS)'DO'
                          (GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
           'FI'
    'FI');

*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
*XXXXXXX                                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
*XXXXXXX        NUMBER OF PAGES    1      XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
*XXXXXXX                                  XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*
```

CHAPQUOTA (1)

1. Calculates total number of jobs started.

2. According to this total the amount of core required for George chapters is calculated. The lower the total number of jobs the less core is required for George chapters for each one.

3. This value is then rounded up to an integer.

NOTES FOR USE

1. Formula specific to George 3 operating system. Used in segment FREECOREADJ (1).

```
#####  #####  #####  #        YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
#   #  #   #  #   #  #        YYYY                        YYYYYYYYYYYYY                                     YYYYYYYYYYYYYYYYYYYY*
#   #  #   #  #   #  #        YYYY                        YYYYYYYYYYYYY                                     YYYYYYYYYYYYYYYYYYYY*
#####  #   #  #####  #####    YYYY        MOP             YYYYYYYYYYYYY    :SSP8086.MOPPMB                  YYYYYYYYYYYYYYYYYYYY*
#   #  #   #  #   #  #   #    YYYY                        YYYYYYYYYYYYY                                     YYYYYYYYYYYYYYYYYYYY*
#   #  #   #  #   #  #   #    YYYY                        YYYYYYYYYYYYY                                     YYYYYYYYYYYYYYYYYYYY*
#####  #####  #####  #####    YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*

#LISTING OF :SSP8086.CHAPQUOTA1(1/) PRODUCED ON 18JUL78 AT 15.26.26

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.58.04 USING  U15

DOCUMENT    CHAPQUOTA1


'C'CALCULATES HOW MUCH CORE WOULD BE TAKEN UP WITH GEORGE CHAPTERS'C'

'PROC'CHAPQUOTA='INT':
    (TOTJOBSTART:=COREIMAGE+TENTSTART;
     'IF'TOTJOBSTART>10
     'THEN'QUOTA:=(TOTJOBSTART-10)*1.0+(5*1.5)+(5*2.0)
     'ELSF'TOTJOBSTART>5
     'THEN'QUOTA:=(TOTJOBSTART-5)*1.5+(5*2.0)
     'ELSE'QUOTA:=TOTJOBSTART*2.0
     'FI';
     'IF'COREIMAGE>5
     'THEN'QUOTA:=QUOTA+((COREIMAGE-5)*0.5)+(5*1.0)
     'ELSE'QUOTA:=QUOTA+COREIMAGE*1.0
     'FI';
     'ROUND'(QUOTA));

*YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
*YYYYYYY                             YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
*YYYYYYY      NUMBER OF PAGES    1   YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
*YYYYYYY                             YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
*YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY*
```

REDTAPEQUOTA (1)

1. Multiplies the number of core images by 2.5K and the number of tentatively started jobs by 2.0 K. The freepool of memory needed for the George 3 operating system to function properly and some memory necessary for essential functions is then added.

2. Rounds up the value gained to an integer.

NOTES FOR USE

1. Formula specific to George 3 operating system

   Used with segment FREECOREADJ (1).

```
#####   #####   #####   #        UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
#   #   #   #   #   #   #        UUUU            UUUUUUUUUUUU                               UUUUUUUUUUUUUUUUUUUU*
#   #   #   #   #   #   #        UUUU            UUUUUUUUUUUU                               UUUUUUUUUUUUUUUUUUUU*
#####   #   #   #####   #####    UUUU     MOP    UUUUUUUUUUUU   :SSP8086.MOPPMB             UUUUUUUUUUUUUUUUUUUU*
#   #   #   #   #   #   #   #    UUUU            UUUUUUUUUUUU                               UUUUUUUUUUUUUUUUUUUU*
#   #   #   #   #   #   #   #    UUUU            UUUUUUUUUUUU                               UUUUUUUUUUUUUUUUUUUU*
#####   #####   #####   #####    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*

#LISTING OF :SSP8086.REDTAPQUOTA1(1/)   PRODUCED ON 18JUL78 AT 15.27.13

#GE.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 12.57.45 USING  U15

DOCUMENT    REDTAPQUOTA1


'C'CALCULATES HOW MUCH CORE WOULD BE TAKEN UP WITH RED TAPE HEADINGS'C'

'PROC'REDTAPEQUOTA='INT':
   (QUOTA:=FREEPOOL+ESSENTFUNCT+(COREIMAGE*2.5)+(TENTSTART*2.0);
    'ROUND'(QUOTA));

*UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
*UUUUUUU                                UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
*UUUUUUU        NUMBER OF PAGES    1    UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
*UUUUUUU                                UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
*UUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUUU*
```

NEWJOB (4)

1. Generates a job by calling JOBSIZGEN.

2. Checks if there is enough space in memory for that job by calling SPACEINCORE.

3. If there is JOBLOAD and FRAGLEVEL are altered accordingly.

4. Records the size of the job, its locations in memory and its type in array JOB. N.B. The locations are determined by the pointer to the gap found by SPACEINCORE.

5. If the gap is larger than the job the remaining space is placed back onto the gap list.

6. The job is thenrolled in by ROLLIN.

7. Increments number of coreimages, and adds on delays for low-level scheduler which would have carried out this operation.

8. Increments JOBNEW.

9. If there is not enough space in memory (SPACEINCORE delivered a 'false' value), a check is made to see if the fragmentation level is greater than the job load.

10. If this is the case, RELOC is called to relocate memory. The number of times relocation was necessary is incremented.

11. JOBLOAD and FRAGLEVEL are altered accordingly.

12. The job characteristics are recorded as above (see 4).

13. A check is made that relocation has taken place correctly - this assists the user who wishes to alter the segment RELOC.

14. Any space which is still left in memory is recorded in the gap list.

15. The job is rolled in by ROLLIN.

16. The number of coreimages is incremented and delays for the low-level scheduler are added.

17. JOBNEW is incremented.

18. This process is continued until a job cannot be brought in, in which case LASTJOBIN is set to 'false' to show that the last job generated by JOBSIZGEN was not used.

```
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG+
GGGG                 GGGGGGGGGGGG                                          GGGGGGGGGGGGGGGGGGG+
GGGG                 GGGGGGGGGGGG                                          GGGGGGGGGGGGGGGGGGG+
GGGG       MOP       GGGGGGGGGGGG    :SSP8086.MOPPMB                       GGGGGGGGGGGGGGGGGGG+
GGGG                 GGGGGGGGGGGG                                          GGGGGGGGGGGGGGGGGGG+
GGGG                 GGGGGGGGGGGG                                          GGGGGGGGGGGGGGGGGGG+
GGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGGG+

#LISTING OF :SSP8086.NEWJOB4(1/)    PRODUCED ON 15SEP78 AT 12.28.49

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.10.47 USING  U15

DOCUMENT    NEWJOB4


'C'INITIATES SELECTION OF AND ROLLING IN OF NEW JOBS'C'

'PROC'NEWJOB=('INT'I)'VOID':
     (           OK:='TRUE';
            'WHILE'OK'DO'
              (PR[I]:=JOBNEW;JOBSIZGEN;
           'IF'SPACEINCORE(I)
              'THEN'FRAGLEVEL'MINUS'JOBSIZE;
               JOBLOAD'PLUS'JOBSIZE;
               JOB[JOBNEW,1]:=JOBSIZE;
               JOB[JOBNEW,2]:=BEGIN'OF'GPTR2  ;
               JOB[JOBNEW,3]:=JOBSIZE+BEGIN'OF'GPTR2-1;
               JOB[JOBNEW,4]:=JOBTYPE;
               'IF'JOB[JOBNEW,3]<END'OF'GPTR2
               'THEN''IF'GHEAD'IS'NULLUS
                     'THEN'GHEAD:=GTAIL:='GAPENTRY':=
                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1,
                            END'OF'GPTR2,'NIL')
                     'ELSE'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1 ,
                            END'OF'GPTR2,'NIL')
                     'FI'
               'FI';
                ROLLIN(I);COREIMAGE'PLUS'1;PRTIME[I]'PLUS'LLSDEL;
                        OSTIME[I]'PLUS'LLSDEL;
               JOBNEW'PLUS'1
          'ELSF'FRAGLEVEL>=JOBLOAD
         'THEN'RELOC(I);PRINT(("RELOCATION NECESSARY",NEWLINE));
              RELOCNO'PLUS'1;
               FRAGLEVEL'MINUS'JOBSIZE;
               JOBLOAD'PLUS'JOBSIZE;
               JOB[JOBNEW,1]:=JOBSIZE;
               JOB[JOBNEW,2]:=JOBLOAD+1-JOBSIZE;
               JOP[JOBNEW,3]:=JOBLOAD;
               JOB[JOBNEW,4]:=JOBTYPE;
               'IF'FRAGLEVEL#FREECORE-JOBLOAD
               'THEN'FRAGLEVEL:=FREECORE-JOPLOAD;
                     PRINT("MISTAKE IN GAPTABLE")
               'FI';
                'IF'FRAGLEVEL>0
                'THEN'                GHEAD:=GTAIL:='GAPENTRY':=
               (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL')
               'ELSE'GHEAD:=GTAIL:='NIL'
                'FI';
               ROLLIN(I);COREIMAGE'PLUS'1;PRTIME[I]'PLUS'LLSDEL;
                        OSTIME[I]'PLUS'LLSDEL;
```

```
                JOBNEW'PLUS'1
              'ELSE'LASTJOBIN:='FALSE';OK:='FALSE'
            'FI'));
```

INSERT (2)

1. Each entry on the list contains the number of the job, the time it is ready to run and it's priority. The priority of the job is based on its type.

2. Adds on a delay to the processor time, SYSTIME and OSTIME.

3. If the ready queue is empty, the new entry becomes the head of the list.

4. Otherwise the list is searched until an entry on the list has a priority which is greater than the new entry. The new entry is then placed on the list before this entry.

5. If no entry on the ready queue has a greater priority, the new entry is placed at the end of the queue.

NOTES FOR USE

1. Before using this segment an extra field must be included in the 'JOBENTRY' structure; an integer field called PRIORITY is used in this segment.

```
#####   #####   #####   #          PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
#   #   #   #   #   #   #          PPPP                       PPPPPPPPPPPP                                         PPPPPPPPPPPPPPPPPPPP+
#   #   #   #   #   #   #          PPPP                       PPPPPPPPPPPP                                         PPPPPPPPPPPPPPPPPPPP+
#####   #   #   #####   #####      PPPP         MOP           PPPPPPPPPPPP    :SSP8086.MOPPMB                      PPPPPPPPPPPPPPPPPPPP+
#   #   #   #   #   #   #   #      PPPP                       PPPPPPPPPPPP                                         PPPPPPPPPPPPPPPPPPPP+
#   #   #   #   #   #   #   #      PPPP                       PPPPPPPPPPPP                                         PPPPPPPPPPPPPPPPPPPP+
#####   #####   #####   #####      PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+

#LISTING OF :SSP8086.INSERT2(1/)    PRODUCED ON 31AUG78 AT 11.54.16

#68.639 AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.16.03 USING  U15

DOCUMENT    INSERT2



'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
    IT WILL BE READY TO RUN'C'

'PROC'INSERT=('INT'I)'VOID':
   (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL;
'IF'HEAD'IS'NULL
   'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],JOB[PR[I],4],'NIL')
   'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
         'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
           ('IF'JOB[PR[I],4]>=PRIORITY'OF'PTR2
            'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
            'ELSF'PTR1'IS'NULL
            'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],JOB[PR[I],4],PTR2);
                  CONTINUE:='FALSE'
            'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],JOB[PR[I],4],PTR2);
                   CONTINUE:='FALSE'
            'FI');
            'IF'NEXT'OF'PTR2'IS'NULL
            'THEN''IF'PRIORITY'OF'PTR2<=JOB[PR[I],4]
                  'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
                              (PR[I ],PRTIME[I ],JOB[PR[I],4],'NIL')
                  'ELSF'PTR1'IS'NULL
                  'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],JOB[PR[I],4],PTR2)
                  'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],JOB[PR[I],4],PTR2)
                  'FI'
            'FI'
   'FI');

+PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
+PPPPPPP                            PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
+PPPPPPP       NUMBER OF PAGES    1     PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
+PPPPPPP                            PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
+PPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPPP+
```

INSERT (3)

1. Each entry on the list contains the number of the job, the time it is ready to run and its priority. The priority of the job is based upon certain types of jobs.

2. Adds on delays to processor time, SYSTIME and OSTIME.

3. If the job is of an even numbered type, the priority of that job is 1 otherwise it is 2.

4. If the ready queue is empty, the new entry becomes the head of the list.

5. Otherwise the list is searched until an entry on the list has a priority which is greater than the new entry. The new entry is then placed on the list before this entry.

6. If no entry on the ready queue has a greater priority, the new entry is placed at the end of the queue.

NOTES FOR USE

1. Before using this segment an extra field must be included in the 'JOBENTRY ' structure; an integer field called PRIORITY is used in this segment. Also, a new variable THISPRIOR must be declared, to hold the priority of the job being placed on the list.

2. This method of picking certain jobtypes to have greater priority over others can be extended to having each type a different priority if necessary (but not necessarily in order as in INSERT (2)).

```
#####   #####   #####   #          ##########################################################################################
#   #   #   #   #   #   #          ####                    ##############                                                ####
#   #   #   #   #   #   #          ####                    ##############                                                ####
#####   #   #   #####   #####      ####         MOP        ##############    :SSP8086.MOPPMB                             ####
#   #   #   #   #   #   #   #      ####                    ##############                                                ####
#   #   #   #   #   #   #   #      ####                    ##############                                                ####
#####   #####   #####   #####      ##########################################################################################

#LISTING OF :SSP8086.INSERT3(1/)    PRODUCED ON 15SEP78 AT 11.40.39

#68.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.10.52 USING  U15

DOCUMENT    INSERT3


'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
    IT WILL BE READY TO RUN'C'

'PROC'INSERT=('INT'I)'VOID':
   (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL;
     'IF'JOB[PR[I],4]=2'OR'
          JOB[PR[I],4]=4'OR'
            JOB[PR[I],4]=6'OR'
            JOB[PR[I],4]=8
     'THEN'THISPRIOR=1
     'ELSE'THISPRIOR=2
      'FI';
'IF'HEAD'IS'NULL
   'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],THISPRIOR,'NIL')
   'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
         'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
           ('IF'THISPRIOR>=PRIORITY'OF'PTR2
            'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
            'ELSE'PTR1'IS'NULL
            'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],THISPRIOR,PTR2);
                  CONTINUE:='FALSE'
            'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],THISPRIOR,PTR2);
                   CONTINUE:='FALSE'
            'FI');
            'IF'NEXT'OF'PTR2'IS'NULL
            'THEN''IF'PRIORITY'OF'PTR2<=THISPRIOR
                  'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
                              (PR[I ],PRTIME[I ],THISPRIOR,'NIL')
                  'ELSE'PTR1'IS'NULL
                  'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],THISPRIOR,PTR2)
                  'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],THISPRIOR,PTR2)
                  'FI'
            'FI'
   'FI');

#########################################################################################################################
#######                                 ###############################################################################
########        NUMBER OF PAGES    1    ###############################################################################
#######                                 ###############################################################################
#########################################################################################################################
```

INIT (3)

1. Reads in values for system and workload specifications.

2. Sets number of tentatively started jobs to four.

3. Fills available memory (FREECORE) with jobs generated by JOBSIZGEN, Incrementing number of coreimages for each job.
N.B. FREECORE is not set but can fluctuate and its value is determined by the total amount of memory minus that required for George chapters and red tape headings.

4. Notes size, locations of job and its type in array JOB.

5. Sets JOBLOCKSTART to zero since the job entered the system at time zero.

6. Increase JOBLOAD by the size of the job.

7. If the job generated cannot be fitted into memory the number of coreimages is decremented.

8. Freecore is set to its new value as in 3.

9. The fragmentation level is noted.

10. JOBNEW is set equal to the number of the last job.

11. A sample is taken of the fragmentation level.

12. The gap left in memory is placed on the gaplist.

13. LASTJOBIN is set to 'false' to indicate that the last job generated by JOBSIZGEN was not used.

14. If the memory is filled exactly the head of the gaplist is set to null and a sample taken of the fragmentation level.

15. All timings are set to 0.0 and all heads of lists to null.

16. The entries for the first jobs are assigned to processors.

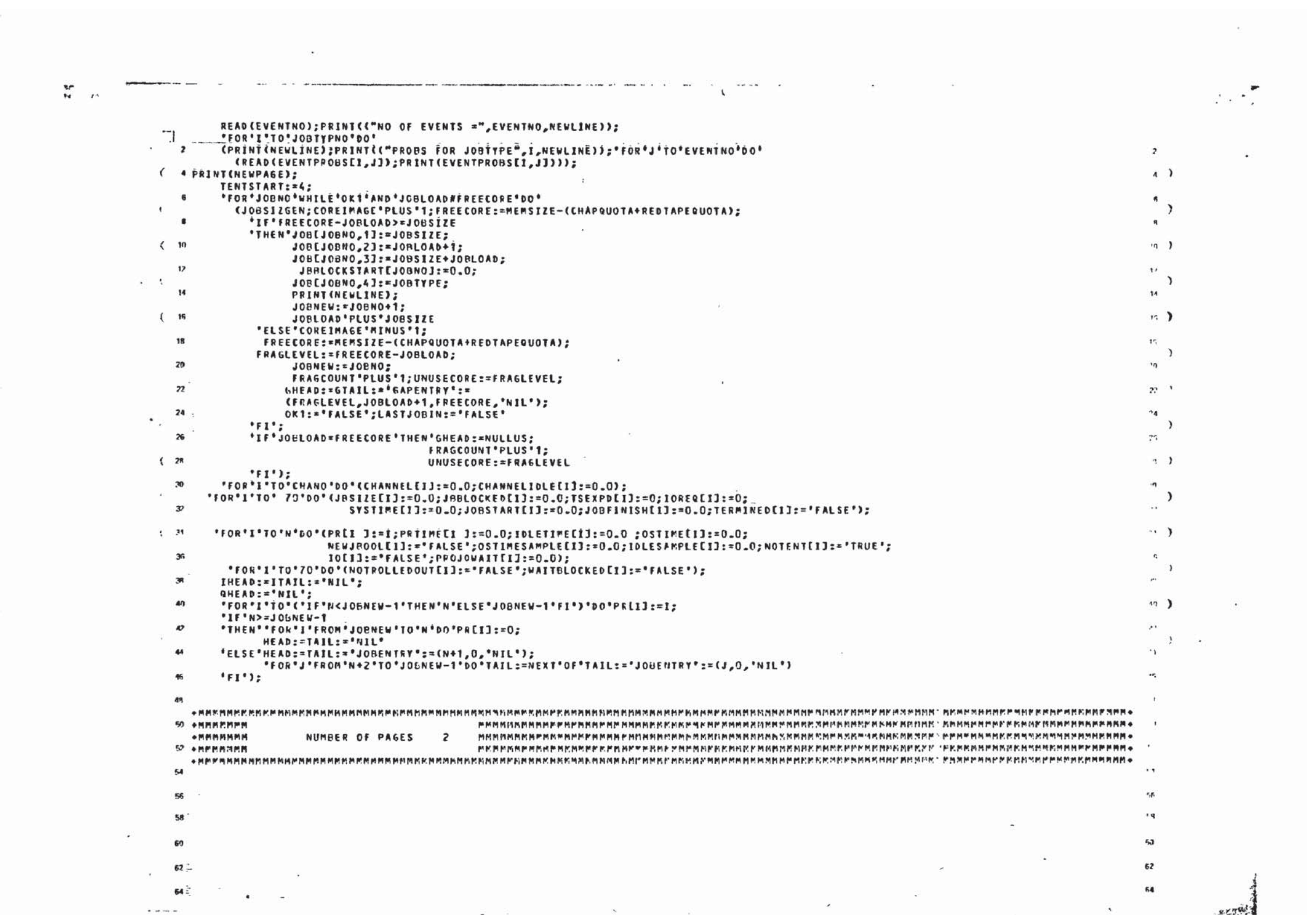
17. Any other core images are placed on the ready queue.

NOTES FOR USE

1. INIT (3) is for use with the other segments adapted for the George 3 system model (see INSTRGEN (2) and NEWJOB (3). Four other new routines were developed for this model. They are:- HLS (1), FREECOREADJ (1), CHAPQUOTA (1), REDTAPEQUOTA (1).

```
#LISTING OF :SSP8086_INIT3(1/)  PRODUCED ON 21SEP78 AT 15.20.25

#68.63B AT ASTON    IN ':SSP8086_MOPPMB' ON 18MAY79 AT 13.13.09 USING  U15

DOCUMENT    INIT3


'C'INITIALISES SIMULATION VARIABLES AND STATEFROM WHICH SIMULATION BEGINS'C'

'PROC'INIT='VOID':
   (OK1:='TRUE';
    'FOR'I'FROM'0'TO'7C'DO'
       'FOR'J'TO'4'DO'JOB[I,J]:=0;

READ(TITLE);PRINT((TITLE,NEWLINE,"--------------------",NEWLINE));
PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(FREECORE);PRINT(("FREE CORE =",FREECORE,NEWLINE));
    READ(MEMSIZE);PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE));
    READ(N);PRINT(("NO OF PROCESSORS =",N,NEWLINE));
    READ(CHANO);PRINT(("NO OF CHANNELS =",CHANO,NEWLINE));
    READ(IODEL);OUTF(STANDOUT,$"I!O DELAY ="<3.1>L$,(IODEL));
    READ(REALLOCDEL);OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL));
    READ(INSERTDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL));
    READ(ROLLOUTDEL);OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL));
    READ(ROLLINDEL);OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL));
    READ(RELOCDEL);OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL));
    READ(CHANNELDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL));
    READ(GAPCHECKDEL);OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL));
    READ(INTRUPDEL);OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL));
    READ(SWAPQDEL);OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAPQDEL));
    READ(HLSDEL);OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL));
    READ(LLSDEL);OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL));
    READ(ACCOUNTDEL);OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL));
    READ(SEGSIZE);PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE));
    READ(SEGNO);PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE));
    READ(CHAPCHDEL);OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL));
    READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
    READ(ESSENTFUNC1);PRINT(("MEMORY NEEDED FOR ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
    READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
    READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
    READ(TYPPERCENT[1]);PRINT(("% OF TYPE    1 ",TYPPERCENT[1],NEWLINE));
    'FOR'I'FROM'2'TO'JOBTYPNO'DO'
      (READ(TYPPERCENT[I]);
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
       TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1];
    READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
    READ(SIZPERCENT[1]);
    READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
    'FOR'I'FROM'2'TO'JOBSIZNO'DO'
      (READ(SIZPERCENT[I]);
       READ(JOBSIZES[I]);
     PRINT((SIZPERCENT[I],"% OF JOBS ARE",JOBSIZES[I],"K",NEWLINE));
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1];
```

```
      READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
      'FOR'I'TO'JOBTYPNO'DO'
      (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
        (READ(EVENTPROBS[I,J]);PRINT(EVENTPROBS[I,J])));
PRINT(NEWPAGE);
      TENTSTART:=4;
      'FOR'JOBNO'WHILE'OK1'AND'JOBLOAD#FREECORE'DO'
        (JOBSIZGEN;COREIMAGE'PLUS'1;FREECORE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
          'IF'FREECORE-JOBLOAD>=JOBSIZE
          'THEN'JOB[JOBNO,1]:=JOBSIZE;
                JOB[JOBNO,2]:=JOBLOAD+1;
                JOB[JOBNO,3]:=JOBSIZE+JOBLOAD;
                 JBBLOCKSTART[JOBNO]:=0.0;
                JOB[JOBNO,4]:=JOBTYPE;
                PRINT(NEWLINE);
                JOBNEW:=JOBNO+1;
                JOBLOAD'PLUS'JOBSIZE
           'ELSE'COREIMAGE'MINUS'1;
            FREECORE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
           FRAGLEVEL:=FREECORE-JOBLOAD;
                JOBNEW:=JOBNO;
                FRAGCOUNT'PLUS'1;UNUSECORE:=FRAGLEVEL;
               &HEAD:=&TAIL:='&APENTRY':=
               (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL');
               OK1:='FALSE';LASTJOBIN:='FALSE'
          'FI';
          'IF'JOBLOAD=FREECORE'THEN'&HEAD:=NULLUS;
                                   FRAGCOUNT'PLUS'1;
                                   UNUSECORE:=FRAGLEVEL
          'FI');
      'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
    'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JBBLOCKED[I]:=0.0;TSEXPD[I]:=0;IOREQ[I]:=0;
                      SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

     'FOR'I'TO'N'DO'(PR[I ]:=I;PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                    NEWJBOOL[I]:='FALSE';OSTIMESAMPLE[I]:=0.0;IDLESAMPLE[I]:=0.0;NOTENT[I]:='TRUE';
                    IO[I]:='FALSE';PROJOWAIT[I]:=0.0);
       'FOR'I'TO'70'DO'(NOTPOLLEDOUT[I]:='FALSE';WAITBLOCKED[I]:='FALSE');
      IHEAD:=ITAIL:='NIL';
      QHEAD:='NIL';
      'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'PR[I]:=I;
      'IF'N>=JOBNEW-1
      'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;
            HEAD:=TAIL:='NIL'
      'ELSE'HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
            'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
      'FI');
```



INSTRGEN (2)

1. Sets processor time equal to REALTIME.

2. If the processor is not idle, the type of the job is used to determine the probabilities of events occurring to that job.

3. Generates a random number between 0-100,000.

4. The variable JBSIZE notes the mill-time used by the job and is incremented one time unit (i.e. 1 millisec), as is the processor time.

5. If the random number falls in the range of the event time-slice expired, a number is made of the number of time-sliced used by that job and an entry for the processor is placed on the critical region queue from where it will place an entry for the job onto the ready queue and be reallocated with another.

6. If the random number falls in the range indicating input/output request, a note is made of the number of input/output requests made by the job. INOUT is called to block the job an initiate input/output. The processor is then put on the critical region queue with NEWJBOOL set to show it requires reallocation with another job.

7. If it falls in the range indicating job termination, a count is made of the number of jobs terminated. An accounting delay is added onto the processor time, and ENDJOB is called to roll out the job. The processor is then passed to the critical region queue from where it will be reallocated with another job.

8. If it falls in the range indicating chapter change a delay is added onto the processor time, SYSTIME and OSTIME and a note is made of the number of chapter changes that occur.

9. If it falls in the range indicating chapter transfer a delay is added for a chapter change since the location of the chapter still has to be determined. The transfer of the chapter from secondary storage to memory is then treated as an input/output request for simplicity.

10. If it falls in the range of a job entering the well or being started from a M.O.P. terminal, this acts as a kind of interrupt and the job must be placed on the ready queue. The number of jobs in the well is then increased and HLS is called to determine whether a job can be tentatively started. The processor is then put on the critical region queue from where it will be reallocated with another job.

11. If it falls outside the above ranges the next instruction is assumed to be an ordinary one (see NOTES FOR USE below).

12. If the processor is idle it is placed on the critical region queue with NEWJBOOL set to show it requires reallocation with another job.

NOTES FOR USE

1. This segment only checks for events which require some operating system intervention. 1.0 millisecs is added onto the mill time because the model assumes that the instructions between such events require only arithmetic computation eg. +, -, *, /, shift etc. The average time for such instructions is 0.002 millisecs and thus the segment check for events requiring operating system intervention every 500 instructions.

```
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL.
LLLL                LLLLLLLLLLLL                                      LLLLLLLLLLLLLLLLLL.
LLLL                LLLLLLLLLLLL                                      LLLLLLLLLLLLLLLLLL.
LLLL       MOP      LLLLLLLLLLLL    :SSP8086.MOPPMB                   LLLLLLLLLLLLLLLLLL.
LLLL                LLLLLLLLLLLL                                      LLLLLLLLLLLLLLLLLL.
LLLL                LLLLLLLLLLLL                                      LLLLLLLLLLLLLLLLLL.
LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL.

#LISTING OF :SSP8086.INSTRGEN2(1/)  PRODUCED ON 21SEP78 AT 15.21.12

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.13.04 USING  U15

DOCUMENT     INSTRGEN2


'C'GENERATES THE NEXT EVENT THAT IS TO OCCUR WITHIN EACH PARTICULARJOBTHAT
       IS RUNNING'C'


'PROC'INSTRGEN=('INT'I,JOBTYPE)'VOID':
 (PRTIME[I]:=REALTIME;
   'IF'PR[I ]>0
   'THEN'NO:='ENTIER'(RAND2*100000);
               JBSIZE[PR[I] ]'PLUS'1.0;PRTIME[I]'PLUS'1.0;
         'IF'TOTAL:=EVENTPROBS[JOBTYPE,1];NO<TOTAL
         'THEN'TSEXPD[PR[I]]'PLUS'1;
               JQLIST(I)
        'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,2];NO<TOTAL
        'THEN'IOREQ[PR[I]]'PLUS'1;
               INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
        'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,3];NO<TOTAL
         'THEN'JOBCOUNT'PLUS'1;TERMINED[PR[I]]:='TRUE';PRINT(("JOB",PR[I],"TERMED",NEWLINE));
              NEWJBOOL[I]:='TRUE';PRTIME[I]'PLUS'ACCOUNTDEL;HLS(I);ENDJOB(I);JQLIST(I)
         'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,4];NO<TOTAL
          'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
               OSTIME[I]'PLUS'CHAPCHDEL;
             CHAPCHNO'PLUS'1

          'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,5];NO<TOTAL
          'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
             CHAPTRANSNO'PLUS'1;OSTIME[I]'PLUS'CHAPCHDEL;
               INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
         'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,6];NO<TOTAL
          'THEN'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
                 NEWJBOOL[I]:='TRUE';
               JOBSINWELL'PLUS'1;HLS(I);
               JQLIST(I)
         'FI'
   'ELSF'PR[I ]=0'THEN'NEWJBOOL[I]:='TRUE';JQLIST(I)
   'FI');

+LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL+
+LLLLLLL                          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL+
+LLLLLLL     NUMBER OF PAGES   1  LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL+
+LLLLLLL                          LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL+
+LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLLL+
```

APPENDIX 9

SPECIFICATION VARIABLE LIST

SERIAL NUMBER	NAME	TYPE OF VALUE	MEANING	STANDARD VALUE
1.	FREECORE	INTEGER (K)	Amount of memory available for core images	97
2.	MEMSIZE	INTEGER (K)	Total size of memory	172
3.	N	INTEGER	Number of processors	1
4.	CHANO	INTEGER	Number of channels	4
5.	IODEL	REAL (MSECS)	Delay for carrying out input/output operation	70.0
6.	REALLOCDEL	REAL (MSECS)	Delay for reallocating a processor with a job from the ready queue	1.0
7.	INSERTDEL	REAL (MSECS)	Delay for inserting a job into the ready queue	1.0
8.	ROLLOUTDEL	REAL (MSECS)	Delay for rolling out a job onto secondary storage	600.0
9.	ROLLINDEL	REAL (MSECS)	Delay for rolling in a job from secondary storage	600,0
10.	RELOCDEL	REAL (MSECS)	Delay for relocating the jobs in memory	5.0
11.	CHANNELDEL	REAL (MSECS)	Delay for checking if there is enough space in memory	2.0
12.	GAPCHECKDEL	REAL (MSECS)	Delay for checking if there is enough space in memory for a job	2.0
13.	INTRUPDEL	REAL (MSECS)	Delay for dealing with interrupts	2.5
14.	SWAPQDEL	REAL (MSECS)	Delay for swapping a job from blocked state to the ready queue	2.0
15.	HLSDEL	REAL (MSECS)	Delay for high-level scheduler	1.0
16.	LLSDEL	REAL (MSECS)	Delay for low-level scheduler	1.5
17.	ACCOUNTDEL	REAL (MSECS)	Delay for accounting routines	3.0
18.	SEGSIZE	INTEGER (K)	Size of memory segments	30
19.	SEGNO	INTEGER	Number of segments for core images	3
20.	CHAPCHDEL	REAL (MSECS)	Delay for changing from one O.S. chapter to another (George 3)	1.0

cont ....

SERIAL NUMBER				STANDARD VALUE
21.	FREEPOOL	INTEGER (K)	Amount of memory to be free for George 3 to operate properly	5
22.	ESSENTFUNCT	INTEGER (K)	Amount of memory required by George 3 for essential functions	3

APPENDIX 10

PROCESS VARIABLES AND STRUCTURES

NAME	TYPE OF VALUE	MEANING
CONTINUED	BOOL	Used in terminating loops
OK	BOOL	Used in terminating loops
GAPFOUND	BOOL	Denotes whether a space in memory has been found for a job or not
OK1	BOOL	Used in segment GAPLIST (1) to denote whether a gap joins another or not
BOOL	BOOL	A parameter to INTRUPLIST, equivalent to INTRUPTYPE
INTRUPTYPE	BOOL	Denotes whether job requires swapping from blocked state to ready queue (T) or a job has just been rolled out (F)
LASTJOBIN	BOOL	Denotes whether last job generated by JOBSIZGEN was used or not
JOBLOCKSTART	REAL(millisecs)	Records when a job was blocked
RV	REAL	Holds random numbers generated in RAND 1 and RAND 2
SAVE	INTEGER (K)	Used to hold the value which FREECORE should have in FREECOREADJ
JOBNEW	INTEGER	Holds number of next job to be brought into memory
NO	INTEGER	Used to hold random number within the specified ranges
SIZ	INTEGER (K)	Specifies size of gap
BEG	INTEGER	Specifies location of beginning of a gap
FIN	INTEGER	Specifies location of end of a gap
HLS	INTEGER	Used in segment HLS to hold amount of memory taken by tentatively started jobs
JOBNO	INTEGER	Holds number of job
JOBSIZE	INTEGER	Holds size of job just generated
CHA	INTEGER	Holds number of a channel
REALTIME	REAL(millisecs)	Master-clock

NAME	TYPE OF VALUE	MEANING
IOTIME	REAL(millisecs)	Holds the time when the next channel is available
JBLISTIME	REAL(millisecs)	Holds the time when the next processor can pass through the critical region
TEMP	REAL(millisecs)	Used in NXTEVENTIME to hold the time of the next event throughout the system
JOB	INTEGER	Used to hold the characteristics of jobs on the system
CHANNEL	REAL(Millisecs)	Holds time each channel will be available
NOTROLLEDOUT	BOOL	Denotes whether each job is in the process of being rolled out or not
WAITBLOCKED	BOOL	Denotes whether each job is blocked or not
TERMINED	BOOL	Denotes whether each job has terminated or not
PR	INTEGER	Holds the number of the job associated with each processor. If PR[2] = 7 then processor 2 would be currently executing job number 7
PRTIME	REAL(millisecs)	Holds the time of each processor
NOTENT	BOOL	Denotes whether each processor is currently on the critical region queue or not
IO	BOOL	Denotes whether the processor is currently engaged in placing an entry for its job on the queue for input/output resources due to an input/output request by that job or due to a job being rolled in or out (F)
NEWJBOOL	BOOL	Denotes whether each processor requires only reallocation (T) or not (F)
(INTRUPENT	STRUCTURE	Structure of each entry in the interrupt list:-
(INTRUPTYPE	REAL(millisecs)	Time when interrupt should occur
(INTRUPTYP	BOOL	Equivalent to INTRUPTYPE
IHEAD	INTRUPENT STRUCTURE	Head of interrupt list
IPTR1	INTRUPENT STRUCTURE	Pointer to the interrupt list
IPTR2	INTRUPENT STRUCTURE	Pointer to the interrupt list

NAME	TYPE OF VALUE	MEANING
ITAIL	INTRUPENT STRUCTURE	End of interrupt list
(IOLISTENT	STRUCTURE	Structure of entry in the input/output queue
(		
(IOTIM	REAL(millisecs)	Time when entry was placed on the list
(		
(JOB	INTEGER	Number of job which requested input/ output or terminated or is being rolled in
(		
IOHEAD	IOLISTENT STRUCTURE	Head of queue for input/output resources
IOPTR1	IOLISTENT STRUCTURE	Pointer to the input/output queue
IOPTR 2	IOLISTENT STRUCTURE	Pointer to the input/output queue
IOTAIL	IOLISTENT STRUCTURE	End of the input/output queue
(JQENT	STRUCTURE	Structure of entry in the critical region queue
(		
(PRO	INTEGER	Number of the processor in the queue
(		
(QTIME	REAL(millisecs)	Time at which the processor entered the queue
(		
QHEAD	JQENT STRUCTURE	Head of the critical region queue
QPTR1	JQENT STRUCTURE	Pointer to critical region queue
QPTR 2	JQENT STRUCTURE	Pointer to critical region queue
(GAPENTRY	STRUCTURE	Structure of each entry in the gap list
(		
(SIZE	INTEGER (K)	Size of job
(		
(BEGIN	INTEGER	Location of beginning of job in memory
(		
(END	INTEGER	Location of end of job in memory
GPTR1	GAPENTRY STRUCTURE	Pointer to the gap list
GPTR 2	GAPENTRY STRUCTURE	Pointer to the gap list
GPTR 3	GAPENTRY STRUCTURE	Pointer to the gap list
GHEAD	GAPENTRY STRUCTURE	Head of the gap list

NAME	TYPE OF VALUE	MEANING
GTAIL	GAPENTRY STRUCTURE	End of the gap list
(JOBENTRY	STRUCTURE	Structure of each entry in the ready queue
(P	INTEGER	Number of the job
(TIME	REAL(millisecs)	Holds the time when the job is ready to run
(PRIORTY	INTEGER	Holds the priority of the job
PTR 1	JOBENTRY STRUCTURE	Pointer to the ready queue
PTR 2	JOBENTRY STRUCTURE	Pointer to the ready queue
HEAD	JOBENTRY STRUCTURE	Head of the ready queue
TAIL	JOBENTRY STRUCTURE	End of the ready queue
SEGRMPTY	BOOL	Denotes whether each memory segment is empty or not
TOTJOBSTART	INTEGER	Holds total number of jobs fully and tentatively started
QUOTA	REAL (K)	Used in segments REDTAPEQUOTA and CHAPQUOTA and holds the amount of memory taken up by either red tape headings or chapters
THISPRIOR	INTEGER	Holds the priority of the job being placed on the ready queue
SEED 1	INTEGER	Initializes the sequences of numbers gained from RAND1
SEED2	INTEGER	Initializes the sequences of numbers gained from RAND2.

APPENDIX 11

MEASURING VARIABLES AND STRUCTURES

NAME	TYPE OF VALUE	MEANING
JBSIZE	REAL(millisecs)	Records the mill time used by each job. Job number 1 mill time is found in element [1] of array JBSIZE
JBBLOCKED	REAL(millisecs)	Records the amount of time each job was blocked. Job numbers correspond to elements of the array
JOBSTART	REAL(millisecs)	Records the time each job was introduced to the system
JOBFINISH	REAL(millisecs)	Records the time each job left the system
SYSTIME	REAL(millisecs)	Records the amount of time spent in the operating system due to each job's operations. i.e. the operating system time for each job
SAMPLEVAR	REAL(millisecs)	Used to take sample snapshots of the system every x millisecs
PROIDLESUM	REAL(millisecs)	Records the total time idle (every processor)
CHANNELIDLESUM	REAL(millisecs)	Records the total time all the channels are idle
PROJQWAITSUM	REAL(millisecs)	Records the total time all processors spent in the critical region queue
JBBLOCKSUM	REAL(millisecs)	Records the total time all jobs are blocked
IOQSIZE	INTEGER	Records number of entries in the queue for input/output resources
JQSIZE	INTEGER	Records the number of entries in the critical region queue
COREIMAGE	INTEGER	Records the number of core images in memory
JOBCOUNT	INTEGER	Records the number of jobs which have terminated
JOBLOAD	INTEGER (K)	Records the amount of memory being used by coreimages
IOSUM	INTEGER	Counts the number of input/output requests
FRAGLEVEL	INTEGER (K)	Records the amount of memory unused
UNUSECORE	INTEGER (K)	Used in samples, totals fragmentation levels

NAME	TYPE OF VALUE	MEANING
FRAGCOUNT	INTEGER	Used with UNUSECORE in samples. Counts number of times the fragmentation level was totalled
AVGUNUSED	INTEGER (K)	Records average amount of memory unused
PERCENT	INTEGER (%)	Records average amount of memory unused as a percentage of total memory
SYSCORE	INTEGER (K)	Records amount of memory taken up by the operating system. Totalled up in samples
SYSCORECOUNT	INTEGER	Counts number of times SYSCORE is totalled up
SYSCOREAUG	INTEGER (K)	Records average amount of memory taken up by the operating system
USERCORE	INTEGER (K)	Records the amount of memory taken up by coreimages. Totalled up in samples
USERCORECOUNT	INTEGER	Counts number of times USERCORE is totalled up
USERCOREAUG	INTEGER (K)	Records average amount of memory used by core images
IDLETIME	REAL (millisecs)	Records amount of time each processor is idle
PROJQWAIT	REAL(millisecs)	Records the amount of time each processor waits in the critical region queue
OSTIME	REAL(millisecs)	Records the amount of time each processor spends in operating system routines
CHANNEL IDLE	REAL(millisecs)	Records the amount of time each channel is idle
TSEXPD	INTEGER	Counts the number of timeslices each job uses
IOREQ	INTEGER	Counts the number of input/output requests each job makes
SAMPLEPRINT	REAL(millisecs)	Used the same as SAMPLEVAR, except that specifies how often sample results should be output
OSTOT	REAL(%)	Records how much time all processors have spent in operating system routines since last sample in percentage form
IDLETOT	REAL (%)	Records how much time all processors have been idle since last sample in percentage form

NAME	TYPE OF VALUE	MEANING
OBJTIME	REAL(millisecs)	Records total time spent on user programs
TENTSTART	INTEGER	Records the number of jobs which have been tentatively started
JOBSINWELL	INTEGER	Records the number of jobs in the background queue
CHAPCHNO	INTEGER	Records the number of chapter changes
CHAPTRANSNO	INTEGER	Records the number of chapter transfers needed
OSTIMESAMPLE	REAL(millisecs)	Records the time spent in operating system routines, up to the last sample, by each processor
IDLESAMPLE	REAL(millisecs)	Records the time each processor was idle up to the time of the last sample

APPENDIX 12

LISTING OF SKELETON USING SEGMENTED MEMORY ACCESS/MANAGEMENT SCHEME

```
#####   #####   #####   #          VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
#   #   #   #   #   #   #          VVVV                    VVVVVVVVVVVV                                          VVVVVVVVVVVVVVVVVV+
#   #   #   #   #   #   #          VVVV                    VVVVVVVVVVVV                                          VVVVVVVVVVVVVVVVVV+
#####   #   #   #####   #####      VVVV        MOP         VVVVVVVVVVVV    :SSP8086.MOPPMB                       VVVVVVVVVVVVVVVVVV+
#   #   #   #   #   #   #   #      VVVV                    VVVVVVVVVVVV                                          VVVVVVVVVVVVVVVVVV+
#   #   #   #   #   #   #   #      VVVV                    VVVVVVVVVVVV                                          VVVVVVVVVVVVVVVVVV+
#####   #####   #####   #####      VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+

#LISTING OF :SSP8086.SCOPSEG(9/)    PRODUCED ON  5SEP78 AT 15.20.04

#G8.63B AT ASTON    IN ':SSP8086.MOPPMB' ON 18MAY79 AT 13.19.57 USING  U15

DOCUMENT    SCOPSEG


TIMESIM
'BEGIN'
'BOOL'CONTINUE:='TRUE',OK:='TRUE',GAPFOUND,OK1,BOOL,INTRUPTYPE,LASTJOBIN:='TRUE';
 [1:70 ]'REAL'JBSIZE,JBBLOCKED,JOBSTART,JOBFINISH,JBBLOCKSTART,SYSTIME;
[1:100]'INT'IOQ;
'REAL'RV,SAMPLEVAR:=50.0,PROIDLESUM:=0.0,CHANNELIDLESUM:=0.0,PROJQWAITSUM:=0.0,JBLOCKSUM:=0.0;
'INT' SEED1:=11,SEED2:=7,IOQSIZE:=0,JQSIZE:=0,FREECORE,SAVE,COREIMAGE:=0,TOTAL;
'INT'JOBNEW ,JOBCOUNT:=0,X,NO,I,SIZ,BEG,FIN,FRAGVAR,
        JOBNO,INTRUPPTR1:=1,INTRUPPTR2:=1,JOBLOAD:=0,JOBSIZE,IOSUM:=0,
     FRAGLEVEL:=0,NEXTJOB:=0,UNUSECORE:=0,FRAGCOUNT:=0,PERCENT,AVGUNUSED,
     CHA,NXTTYPE:=0,SLICEPROB,IOPROB,TERMPROB,JOBTYPE,SEG,
     SYSCORE:=0, SYSCORECOUNT:=0,USERCORE:=0,USERCORECOUNT:=0,
      USERCOREAVG,SYSCOREAVG;

'REAL'REALTIME:=0.0,IOTIME:=0.0,JBLISTIME:=0.0,TEMP,CHAPCHDEL;
'STRING'TITLE;
'INT'JOBTYPNO:=20,JOBSIZNO:=8,EVENTNO:=8;
[1:JOBTYPNO]'INT'TYPPERCENT;
[1:JOBSIZNO]'INT'JOBSIZES,SIZPERCENT;
[1:JOBTYPNO,1:EVENTNO]'INT'EVENTPROBS;
'INT'ESSENTFUNCT,FREEPOOL,SEGNO:=20,SEGSIZE;

'INT'MEMSIZE,N:=20,CHANO:=20;

'REAL'IODEL,REALLOCDEL,INSERTDEL,ROLLOUTDEL,
        RELOCDEL,CHANNELDEL,GAPCHECKDEL,ROLLINDEL,
        INTRUPDEL,SWAPQDEL,HLSDEL,LLSDEL,ACCOUNTDEL;
[0:70,1:4]'INT'JOB;
[1:N]'REAL'IDLETIME,PROJQWAIT,OSTIME;
[1:CHANO]'REAL'CHANNEL,CHANNELIDLE;
[1:SEGNO]'BOOL'SEGEMPTY;
[1:70]'BOOL'NOTROLLEDOUT,WAITBLOCKED,TERMINED;
[1:70]'INT'TSEXPD,IOREQ;
[1:N ]'INT'PR;
[1:N]'REAL'PRTIME;
[1:N]'BOOL'NOTENT,IO;
[1:N]'BOOL'NEWJBOOL;
'MODE''INTRUPENT'='STRUCT'('REAL'INTRUPTIME,'BOOL'INTRUPTYP,'BOOL'IO,'INT'JOBNO,
                           'REF''INTRUPENT'NXT);
'REF''INTRUPENT'IHEAD,IPTR1,IPTR2,ITAIL;
'REF''INTRUPENT'NIL='NIL';
'MODE''IOLISTENT'='STRUCT'('REAL'IOTIM,'BOOL'IO,'BOOL'INTRUPTYPE,
                           'INT'JOB,'REF''IOLISTENT'IONXT);
'REF''IOLISTENT'IOHEAD,IOTAIL,IOPTR1,IOPTR2;
'REF''IOLISTENT'NOTHING='NIL';
'MODE''JQENT'='STRUCT'('INT'PRO,'REAL'QTIME,'REF''JQENT'QNEXT);
'REF''JQENT'QHEAD,QPTR1,QPTR2;
'REF''JQENT'NILL='NIL';
```

```
'MODE''GAPENTRY'='STRUCT'('INT'SIZE,'INT'BEGIN,'INT'END,
                          'REF''GAPENTRY'PTR);
'REF''GAPENTRY'NULLUS='NIL';
'REF''GAPENTRY'GPTR1,GPTR2,GPTR3,GHEAD,GTAIL;
'MODE''JOBENTRY'='STRUCT'('INT'P,'REAL'TIME,'REF''JOBENTRY'NEXT);
'REF''JOBENTRY'NULL ='NIL';
'REF''JOBENTRY'PTR1,PTR2,HEAD,TAIL;

'C'GENERATES TWO DISTINCT SEQUENCES OF RANDOM NOS TO BE USED FOR GENERATING
    JOB CHARACTERISTICS AND SEQUENCES OF EVENTS RESPECTIVELY'C'

'PROC'RAND1='REAL':
    (NORMRAND:=SEED1;
     RV:=RANDOM;
     SEED1:=NORMRAND;RV);
'PROC'RAND2='REAL':
    (NORMRAND:=SEED2;
     RV:=RANDOM;
      SEED2:=NORMRAND;RV);

'C'CALCULATES TIME WHEN A CHANNEL WILL BE FREE TO CARRY OUT AN I&O OPERATION'C'

'PROC'CHANNELTIME='VOID':
   (IOTIME:=CHANNEL[1];CHA:=1;
    'FOR'I'TO'CHANO'DO'
      'IF'CHANNEL[I]<IOTIME
      'THEN'IOTIME:=CHANNEL[I];CHA:=I
      'FI');

'C'PLACES THE EVENT OF AN INTERRUPT ON A LIST 'C'

'PROC'INTRUPLIST=('REAL'TIME,'INT'JOBNO,'BOOL'BOOL,'BOOL'B)'VOID':
    ('IF'IHEAD'IS'NIL
    'THEN'IHEAD:=ITAIL:='INTRUPENT':=(TIME,BOOL,B,JOBNO,'NIL')
    'ELSE'CONTINUE:='TRUE';IPTR1:='NIL';IPTR2:=IHEAD;
          'WHILE'CONTINUE'AND'(NXT'OF'IPTR2'ISNT'NIL)'DO'
             'IF'TIME>=INTRUPTIME'OF'IPTR2
             'THEN'IPTR1:=IPTR2;IPTR2:=NXT'OF'IPTR2
             'ELSF'IPTR1'IS'NIL
             'THEN'IHEAD:='INTRUPENT':=(TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                                  (TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'FI';
          'IF'NXT'OF'IPTR2'IS'NIL
          'THEN''IF'INTRUPTIME'OF'IPTR2<=TIME
                'THEN'ITAIL:=NXT'OF'ITAIL:='INTRUPENT':=
                       (TIME,BOOL,B,JOBNO,'NIL')
                'ELSF'IPTR1'IS'NIL
                'THEN'IHEAD:='INTRUPENT':=
                       (TIME,BOOL,B,JOBNO,IPTR2)
                'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                       (TIME,BOOL,B,JOBNO,IPTR2)
                'FI'
           'FI'
    'FI');

'C'PLACES AN I&O REQUEST ONTO A LIST WHICH IS SERVED BY THE CHANNELS'C'

'PROC'IOLIST=('INT'I)'VOID':
    ('IF'IOHEAD'IS'NOTHING
```

```
        'THEN'IOHEAD:=IOTAIL:='IOLISTENT':=
              (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
  2   'ELSE'CONTINUE:='TRUE';IOPTR1:='NIL';IOPTR2:=IOHEAD;
            'WHILE'CONTINUE'AND'(IONXT'OF'IOPTR2'ISNT'NOTHING)'DO'
  4          ('IF'PRTIME[I ]>=IOTIM'OF'IOPTR2
              'THEN'IOPTR1:=IOPTR2;IOPTR2:=IONXT'OF'IOPTR2
              'ELSF'IOPTR1'IS'NOTHING
  6           'THEN'IOHEAD:='IOLISTENT':=
                    (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                    CONTINUE:='FALSE'
              'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
  8                 (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                    CONTINUE:='FALSE'
              'FI');
 10           'IF'IONXT'OF'IOPTR2'IS'NOTHING
            'THEN''IF'IOTIM'OF'IOPTR2<=PRTIME[I ]
                    'THEN'IOTAIL:=IONXT'OF'IOTAIL:='IOLISTENT':=
 12                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
                     'ELSF'IOPTR1'IS'NOTHING
                 'THEN'IOHEAD:='IOLISTENT':=
 14                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                    'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                          (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
 16                  'FI'
              'FI'
      'FI');
 18

 20 'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
     IT WILL BE READY TO RUN'C'
 22
  'PROC'INSERT=('INT'I)'VOID':
 24   (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL;
  'IF'HEAD'IS'NULL
 26   'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],'NIL')
      'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
 28        'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
            ('IF'PRTIME[I ]>=TIME'OF'PTR2
             'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
             'ELSF'PTR1'IS'NULL
 30          'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                  CONTINUE:='FALSE'
             'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                   CONTINUE:='FALSE'
 32          'FI');
             'IF'NEXT'OF'PTR2'IS'NULL
             'THEN''IF'TIME'OF'PTR2<=PRTIME[I ]
                  'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
 34                          (PR[I ],PRTIME[I ],'NIL')
                  'ELSF'PTR1'IS'NULL
                  'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
                  'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
 36               ''FI'
             'FI'
    'FI');
 38

 40 'C'INITIALISES I&O OPERATION AND BLOCKS JOB'C'

  'PROC'INOUT=('INT'I)'VOID':
 42   (PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;JPBLOCKSTART[PR[I]]:=PRTIME[I];
       SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
 44
  IO[I]:='TRUE';WAITBLOCKED[PR[I ]]:='TRUE';
 46  INTRUPTYPE:='TRUE';
      IOLIST(I));
```

```
'C'INITIATES ROLLING IN OF A JOB'C'

'PROC'ROLLIN=('INT'I)'VOID':
      (JOBSTART[PR[I]]:=PRTIME[I];
    PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;
    SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='TRUE';
  WAITBLOCKED[PR[I ]]:='TRUE';
    IOLIST(I));

'C'ASSIGNS FIRST ENTRY ON READY QUEUE TO A PROCESSOR'C'

'PROC'REALLOC=('INT'I)'VOID':
     ('IF'HEAD'IS'NULL
       'THEN'
            PR[I ]:=0
    'ELSE'PTR2:=HEAD;
          'IF'TIME'OF'PTR2> PRTIME[I ]
       'THEN'
                 PR[I ]:=0
           'ELSE'NEXT'OF'PTR2'ISNT'NULL
          'THEN'HEAD:=NEXT'OF'PTR2;
                PR[I ]:=P'OF'PTR2
           'ELSE'HEAD:=TAIL:='NIL';
                PR[I ]:=P'OF'PTR2
           'FI'
    'FI';
    PRTIME[I ]:=PRTIME[I ]+REALLOCDEL;
        'IF'PR[I]>0'THEN'OSTIME[I]'PLUS'REALLOCDEL'FI';JBLISTIME:=PRTIME[I ]);

'C'GENERATES JOB CHARACTERISTICS'C'

'PROC'JOBSIZGEN='VOID':
      ('IF'LASTJOBIN'THEN' NO:='ENTIER'(RAND1*100);
    'IF'NO<=TYPPERCENT[1]'THEN'JOBTYPE:=1
    'ELSF'NO<=TYPPERCENT[2]'THEN'JOBTYPE:=2
    'ELSF'NO<=TYPPERCENT[3]'THEN'JOBTYPE:=3
    'ELSF'NO<=TYPPERCENT[4]'THEN'JOBTYPE:=4
    'ELSF'NO<=TYPPERCENT[5]'THEN'JOBTYPE:=5
    'ELSF'NO<=TYPPERCENT[6]'THEN'JOBTYPE:=6
    'ELSF'NO<=TYPPERCENT[7]'THEN'JOBTYPE:=7
   'ELSF'NO<=TYPPERCENT[8]'THEN'JOBTYPE:=8
   'ELSF'NO<=TYPPERCENT[9]'THEN'JOBTYPE:=9
   'ELSF'NO<=TYPPERCENT[10]'THEN'JOBTYPE:=10
   'ELSF'NO<=TYPPERCENT[11]'THEN'JOBTYPE:=11
   'ELSF'NO<=TYPPERCENT[12]'THEN'JOBTYPE:=12
   'ELSF'NO<=TYPPERCENT[13]'THEN'JOBTYPE:=13
   'ELSF'NO<=TYPPERCENT[14]'THEN'JOBTYPE:=14
   'ELSF'NO<=TYPPERCENT[15]'THEN'JOBTYPE:=15
   'ELSF'NO<=TYPPERCENT[16]'THEN'JOBTYPE:=16
   'ELSF'NO<=TYPPERCENT[17]'THEN'JOBTYPE:=17
   'ELSF'NO<=TYPPERCENT[18]'THEN'JOBTYPE:=18
   'ELSF'NO<=TYPPERCENT[19]'THEN'JOBTYPE:=19
    'ELSE'JOBTYPE:=20
    'FI';
     NO:='ENTIER'(RAND1*100);
          'IF'NO<=SIZPERCENT[1]'THEN'JOBSIZE:=JOBSIZES[1]
          'ELSF'NO<=SIZPERCENT[2]'THEN'JOBSIZE:=JOBSIZES[2]
          'ELSF'NO<=SIZPERCENT[3]'THEN'JOBSIZE:=JOBSIZES[3]
          'ELSF'NO<=SIZPERCENT[4]'THEN'JOBSIZE:=JOBSIZES[4]
```

```
            'ELSF'NO<=SIZPERCENT[5]'THEN'JOBSIZE:=JOBSIZES[5]
           'ELSF'NO<=SIZPERCENT[6]'THEN'JOBSIZE:=JOBSIZES[6]
           'ELSF'NO<=SIZPERCENT[7]'THEN'JOBSIZE:=JOBSIZES[7]

            'ELSE'JOBSIZE:=JOBSIZES[8]
            'FI'
     'ELSE'LASTJOBIN:='TRUE'
     'FI');

'C'ORGANISES QUEUE OF PROCESSORS WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'

'PROC'JQLIST=('INT'I)'VOID':
    ('IF'QHEAD'IS'NILL
     'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],'NIL')
     'ELSE'CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
           'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
           'DO'
            ('IF'PRTIME[I ]>=QTIME'OF'QPTR2
             'THEN'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
             'ELSF'QPTR1'IS'NILL
             'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],QPTR2);CONTINUE:='FALSE'
             'ELSE'QNEXT'OF'QPTR1:='JQENT':=(I,PRTIME[I ],QPTR2);
                   CONTINUE:='FALSE'
             'FI');
             'IF'QNEXT'OF'QPTR2'IS'NILL
             'THEN''IF'QTIME'OF'QPTR2<=PRTIME[I ]
                   'THEN'QNEXT'OF'QPTR2:='JQENT':=(I,PRTIME[I ],'NIL')
                   'ELSF'QPTR1'IS'NILL
                   'THEN'QHEAD:='JQENT':=(I,PRTIME[I ],QPTR2)
                   'ELSE'QNEXT'OF'QPTR1:='JQENT':=(I,PRTIME[I ],QPTR2)
                   'FI'
             'FI'
             'FI';
              NOTENT[I]:='FALSE');


'C'INITIALISES SIMULATION VARIABLES AND STATEFROM WHICH SIMULATION BEGINS'C'

'PROC'INIT='VOID':
  (OK1:='TRUE';
   'FOR'I'FROM'0'TO'70'DO'
      'FOR'J'TO'4'DO'JOB[I,J]:=0;

READ(TITLE);PRINT((TITLE,NEWLINE,"--------------------",NEWLINE));
PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(FREECORE);PRINT(("FREE CORE =",FREECORE,NEWLINE));
    READ(MEMSIZE);PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE));
    READ(N);PRINT(("NO OF PROCESSORS =",N,NEWLINE));
    READ(CHANO);PRINT(("NO OF CHANNELS =",CHANO,NEWLINE));
    READ(IODEL);OUTF(STANDOUT,$"I!O DELAY ="<3.1>L$,(IODEL));
    READ(REALLOCDEL);OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL));
    READ(INSERTDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL));
    READ(ROLLOUTDEL);OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL));
    READ(ROLLINDEL);OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL));
    READ(RELOCDEL);OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL));
    READ(CHANNELDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL));
    READ(GAPCHECKDEL);OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL));
    READ(INTRUPDEL);OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL));
    READ(SWAPODEL);OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAPODEL));
    READ(HLSDEL);OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL));
    READ(LLSDEL);OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL));
    READ(ACCOUNTDEL);OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL));
    READ(SEGSIZE);PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE));
    READ(SEGNO);PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE));
    READ(CHAPCHDEL);OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL));
```

```
      READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
      READ(ESSENTFUNCT);PRINT(("MEMORY NEEDED FOR ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
      READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
      READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
      READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
      READ(TYPPERCENT[1]);PRINT(("% OF TYPE   1 ",TYPPERCENT[1],NEWLINE));
      'FOR'I'FROM'2'TO'JOBTYPNO'DO'
        (READ(TYPPERCENT[I]);
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
        TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1]);
      READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
      READ(SIZPERCENT[1]);
      READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
      'FOR'I'FROM'2'TO'JOBSIZNO'DO'
        (READ(SIZPERCENT[I]);
         READ(JOBSIZES[I]);
       PRINT((SIZPERCENT[I],"% OF JOBS ARE",JOBSIZES[I],"K",NEWLINE));
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1]);
      READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
      'FOR'I'TO'JOBTYPNO'DO'
      (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
        (READ(EVENTPROBS[I,J]);PRINT(EVENTPROBS[I,J])));
PRINT(NEWPAGE);
            'FOR'JOBNO'TO'SEGNO'DO'
      (JOBSIZGEN;
           JOB[JOBNO,1]:=JOBSIZE;
                JOB[JOBNO,2]:=JOBNO;
                JOB[JOBNO,3]:=SEGSIZE-JOBSIZE;
                 JBBLOCKSTART[JOBNO]:=0.0;
                JOB[JOBNO,4]:=JOBTYPE;
               FRAGLEVEL'PLUS'SEGSIZE-JOBSIZE;

              COREIMAGE'PLUS'1;
                 PRINT(NEWLINE);
                 JOBNEW:=JOBNO+1;
                 JOBLOAD'PLUS'JOBSIZE);
        FRAGCOUNT'PLUS'1;UNUSECORE'PLUS'FRAGLEVEL;
         FREECORE:=SEGNO*SEGSIZE;
           'FOR'I'TO'SEGNO'DO'SEGEMPTY[I]:='FALSE';
    'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
  'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JBBLOCKED[I]:=0.0;TSEXPD[I]:=0;IOREQ[I]:=0;
                     SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

  'FOR'I'TO'N'DO'(PR[I ]:=I;PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                  NEWJBOOL[I]:='FALSE';NOTENT[I]:='TRUE';
                  IO[I]:='FALSE';PROJCWAIT[I]:=0.0);
      'FOR'I'TO'70'DO'(NOTROLLEDOUT[I]:='FALSE';WAITDLOCKED[I]:='FALSE');
     IHEAD:=ITAIL:='NIL';
     OHEAD:='NIL';
      'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'
       PR[I]:=I;
      'IF'N>=JOBNEW-1
      'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;HEAD:=TAIL:='NIL'
      'ELSE'HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
            'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'
              TAIL:=NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
      'FI');


'C'KEEPS A CHECK ON ALL GAPS IN CORE'C'

'PROC'GAPLIST=('INT'SEG)'VOID':
   (SEGEMPTY[SEG]:='TRUE');
```

```
'C'CHECKS TO SEE IF ENOUGH SPACE IN CORE FOR A JOB TO BE ROLLED IN'C'

'PROC'SPACEINCORE=('INT'I)'BOOL':
   (GAPFOUND:='FALSE';'FOR'I'TO'SEGNO'DO'
        'IF'SEGEMPTY[I]
        'THEN'SEG:=I;GAPFOUND:='TRUE'
        'FI';
        SEGEMPTY[SEG]:='FALSE' ;
       PRTIME[I]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
       GAPFOUND);


'C'SUPER VISES SELECTION AND ROLLING IN OF NEW JOBS'C'

'PROC'NEWJOB=('INT'I)'VOID':
    (          OK:='TRUE';
          'WHILE' OK'DO'
            (PR[I]:=JOBNEW;
           'IF'SPACEINCORE(I)
            'THEN'JOBSIZGEN;
              JOBLOAD'PLUS'JOBSIZE;
              JOB[JOBNEW,1]:=JOBSIZE;
              JOB[JOBNEW,2]:=SEG  ;
              JOB[JOBNEW,3]:=SEGSIZE-JOBSIZE;
              JOB[JOBNEW,4]:=JOBTYPE;
              FRAGLEVEL'MINUS'JOBSIZE;
               POLLIN(I); PRTIME[I]'PLUS'LLSDEL;
                     OSTIME[I]'PLUS'LLSDEL;
               COREIMAGE'PLUS'1;
              JOBNEW'PLUS'1
            'ELSE' OK:='FALSE'
          'FI'));

'C'CHECKS IF THERE ARE ANY INTERRUPTS THAT HAVE YET TO BE DEALT WITHA
     AFTER PROCESSOR HAS JUST DEALT WITH ONE'C'

'PROC'INTRUPCHECK=('INT'I)'VOID':
   ('IF'IHEAD'ISNT'NIL
    'THEN''IF'INTRUPTIME'OF'IHEAD<=PRTIME[I ]
          'THEN'PR[I]:=-1
          'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
          'FI'
    'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
    'FI');
'PROC'INTERUPT=('INT'I)'VOID':
     ('IF'NOTENT[I]'THEN'PRTIME[I]:=REALTIME
   'ELSE''IF'PRTIME[I ]>JBLISTIME
                    'THEN'JBLISTIME:=PRTIME[I ]
            'ELSE''IF'PR[I]NO'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I] 'FI';
                      PRTIME[I ]:=JBLISTIME
          'FI';
         NOTENT[I]:='TRUE';
         CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
         'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
         'DO'('IF'PRO'OF'QPTR2=I
              'THEN''IF'QPTR1'IS'NILL
                     'THEN'QHEAD:=QNEXT'OF'QHEAD
                   'ELSE'QNEXT'OF'QPTR1:=QNEXT'OF'QPTR2
                   'FI';
                   CONTINUE:='FALSE'
              'ELSE'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
              'FI');
              'IF'QNEXT'OF'QPTR2'IS'NILL
```

```
                'THEN''IF'PRO'OF'QPTR2=I
                     'THEN''IF'QPTR1'IS'NILL
                          'THEN'QHEAD:='NIL'
                          'ELSE'QNEXT'OF'QPTR1:='NIL'
                          'FI'
                     'FI'
                'FI'
     'FI';
     'IF'PR[I ]>0
     'THEN''IF'NEWJBOOL[I]
          'THEN'NEWJBOOL[I]:='FALSE'
          'ELSE'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
                     OSTIME[I]'PLUS'INSERTDEL
          'FI'
     'FI';
'IF'INTRUPTYP'OF'IHEAD
        'THEN''IF'IO'OF'IHEAD
          'THEN'PRTIME[I]'PLUS'SWAPODEL;OSTIME[I]'PLUS'SWAPODEL;
               JBLISTIME'PLUS'SWAPODEL;
               PR[I]:=JOBNO'OF'IHEAD;
                JBBLOCKED[PR[I]]'PLUS'PRTIME[I]-JBBLOCKSTART[PR[I]];
               WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
          'ELSE'PR[I]:=JOBNO'OF'IHEAD;
                WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
          'FI';
       IHEAD:=NXT'OF'IHEAD
'ELSE'JOBNO:=JOBNO'OF'IHEAD;
      NOTROLLEDOUT[JOBNO]:='FALSE';
       FRAGLEVEL'PLUS'JOB[JOBNO,1];
       JOBLOAD'MINUS'JOB[JOBNO,1];
            COREIMAGE'MINUS'1;
       GAPLIST(JOB[JOBNO,2]);
      PRTIME[I ]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
      JOB[JOBNO,2]:=JOB[JOBNO,3]:=0;OSTIME[I]'PLUS'INTRUPDEL;
            PRTIME[I ]'PLUS'INTRUPDEL;
           NEWJOB(I);IHEAD:=NXT'OF'IHEAD
   'FI');

'C'WHEN A JOB TERMINATES THIS ROUTINE SETS UP THE ROLING OUT OF THAT JOB'C'

'PROC'ENDJOB=('INT'I)'VOID':
   (NOTROLLEDOUT[PR[I ]]:='TRUE';
    PRTIME[I ]'PLUS'CHANNELDEL;SYSTIME[PR[I]]'PLUS'CHANNELDEL;
      OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='FALSE';    IOLIST(I));

'C'GENERATES NEXT EVENT TO OCCUR WITHIN A PARTIC. JOBTHAT IS FUNNING'C'

'PROC'INSTRGEN=('INT'I,JOBTYPE)'VOID':
  (PRTIME[I ]:=REALTIME;
   'IF'PR[I ]>0
   'THEN'NO:='ENTIER'(RAND2*100000);
         JBSIZE[PR[I] ]'PLUS'1.0;PRTIME[I]'PLUS'1.0;
         'IF'TOTAL:=EVENTPROBS[JOBTYPE,1];NO<TOTAL
         'THEN'TSEXPD[PR[I]]'PLUS'1;
               JOLIST(I)
         'ELSE'TOTAL'PLUS'EVENTPROBS[JOBTYPE,2];NO<TOTAL
         'THEN'IOREQ[PR[I]]'PLUS'1;
                INOUT(I);NEWJBOOL[I]:='TRUE';JOLIST(I)
         'ELSE'TOTAL'PLUS'EVENTPROBS[JOBTYPE,3];NO<TOTAL
         'THEN'JOBCOUNT'PLUS'1;TERMINED[PR[I]]:='TRUE';
               NEWJBOOL[I]:='TRUE';PRTIME[I]'PLUS'ACCOUNTDEL;ENDJOB(I);JOLIST(I)
         'FI'
```

```
      'ELSF'PR[I ]=0'THEN'NEWJBOOL[I]:='TRUE';JQLIST(I)
      'FI');

'C'
      DETERMINES TIME OF NEXT EVENT THROUGHOUT THE SYSTEM'C'

'PROC'NXTEVENTIME='VOID':
   (TEMP:=SAMPLEVAR;
    'FOR'I'TO'N'DO'
       'IF'NOTENT[I]'AND'PR[I ]>0
       'THEN''IF'TEMP>PRTIME[I ]
             'THEN'TEMP:=PRTIME[I ]
             'FI'
       'FI';
    'IF'CHANNELTIME;TEMP>IOTIME
    'THEN'TEMP:=IOTIME
    'FI';

    'IF'TEMP>JBLISTIME
    'THEN''IF'HEAD'ISNT'NULL
          'THEN''IF'TIME'OF'HEAD<=JBLISTIME
                'THEN'TEMP:=JBLISTIME
                'FI'
          'FI';
         'FOR'I'TO'N'DO'
         'IF''NOT'NEWJBOOL[I]'AND''NOT'NOTENT[I]'THEN'TEMP:=JBLISTIME'FI'
    'FI';
    'IF'IHEAD'ISNT'NIL
    'THEN'
    'IF'TEMP>INTRUPTIME'OF'IHEAD
    'THEN'TEMP:=INTRUPTIME'OF'IHEAD
    'FI'
    'FI');


'C' **MAIN BODY**

      ORGANISES RUNNING OF JOBS,QUEUE AROUND CRITICAL REGION ,I?O QUEUE,
      RESULTS TO BE PRINTED OUT DURING RUN,TIME-KEEPING,INTERRUPTS,RESULTS AT END OF RUN'C'

INIT;
'WHILE'JOBCOUNT# 12'DO'
  ('FOR'I'TO'N'DO'
   ('IF'PRTIME[I]<=REALTIME'AND'PR[I ]=0 'THEN'IDLETIME[I]'PLUS'1.0'FI';
     'IF'NOTENT[I]'AND'PR[I ]>=0'THEN'
      'IF'PRTIME[I ]<=  REALTIME'THEN'INSTRGEN(I,JOB[PR[I ],4])'FI'
      'FI');
    'IF'QHEAD'ISNT'NILL
    'THEN''IF'REALTIME>=QTIME'OF'QHEAD'AND'REALTIME>=JBLISTIME
          'THEN'I:=PRG'OF'QHEAD;
                'IF'PRTIME[I]>JBLISTIME
                 'THEN'JBLISTIME:=PRTIME[I]
                 'ELSE''IF'PR[I]#0
                       'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I]
                        'FI';
                       PRTIME[I]:=JBLISTIME
                 'FI';
                NOTENT[I]:='TRUE';
                'IF'NEWJBOOL[I]
                'THEN'REALLOC(I);NEWJBOOL[I]:='FALSE'
                'ELSE'INSERT(I);REALLOC(I)
                'FI';
                'IF'QNEXT'OF'QHEAD'ISNT'NILL
```

```
                'THEN'QHEAD:=QNEXT'OF'QHEAD
                'ELSE'QHEAD:='NIL'
                'FI'
          'FI'
    'FI';

    'FOR'I'TO'CHANO'DO'
   ('IF'CHANNEL[I]<REALTIME
    'THEN'CHANNELIDLE[I]'PLUS'REALTIME-CHANNEL[I];
          CHANNEL[I]:=REALTIME
    'FI';
      'IF'IOHEAD'ISNT'NOTHING
      'THEN''IF'CHANNEL[I]<=REALTIME'AND'IOTIM'OF'IOHEAD<=REALTIME
            'THEN''IF'IO'OF'IOHEAD
                  'THEN'IOTIM'OF'IOHEAD:=REALTIME+IODEL;
                        INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                   INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                       CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                  'ELSE'IOTIM'OF'IOHEAD:=REALTIME+ROLLINDEL;
                        INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                   INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                       'IF''NOT'INTRUPTYPE'OF'IOHEAD
                        'THEN'JOBFINISH[JOB'OF'IOHEAD]:=IOTIM'OF'IOHEAD
                        'FI';

                       CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                  'FI'
            'FI'
      'FI'};
     'IF'REALTIME>=SAMPLEVAR
     'THEN'SAMPLEVAR'PLUS'50.0 ;UNUSECORE'PLUS'FRAGLEVEL;FRAGCOUNT'PLUS'1;
           USERCORE'PLUS'JOBLOAD;USERCORECOUNT'PLUS'1;
            SYSCORE'PLUS'MEMSIZE-FREECORE;SYSCORECOUNT'PLUS'1
     'FI';
     REALTIME'PLUS'1.0;
    NXTEVENTIME;
    'IF'REALTIME<TEMP
    'THEN''FOR'I'TO'N'DO'
           'IF'PR[I ]=0
           'THEN'IDLETIME[I]'PLUS'TEMP-REALTIME
            'FI';
          REALTIME:=TEMP
    'FI';
    'IF'IHEAD'ISNT'NIL
    'THEN''IF'REALTIME>=INTRUPTIME'OF'IHEAD'AND'PRTIME[N ]<=REALTIME
          'THEN'INTERUPT(N);INTRUPCHECK(N)
       ''FI'
    'FI');
     PRINT(NEWPAGE);

    AVGUNUSED:=UNUSECORE'/'FRAGCOUNT;USERCOREAVG:=USERCORE'/'USERCORECOUNT;
    SYSCOREAVG:=SYSCORE'/'SYSCORECOUNT;PERCENT:=AVGUNUSED*100'/'MEMSIZE;
    PRINT(NEWLINE);
    OUTF(STANDOUT,$"REALTIME IS -"<8.3>2L$,(REALTIME));
    PRINT(("PROCESSOR USAGE",NEWLINE,"---------------",NEWLINE));
    PRINT(("PRO   PROIDLETIME %TIME PRO IDLE WAIT IN JOB Q %TIME IN JOB Q    OSTIME     % TIME IN OS",NEWLINE));
    'FOR'I'TO'N'DO'
     (OUTF(STANDOUT,$<1> ,<8.3>PX,<3.1>3X,<8.3>6X,<2.1>6X,<8.3>4X,<2.1>L$,
          (I,IDLETIME[I],IDLETIME[I] *100/ REALTIME ,PROJQWAIT[I],
                PROJQWAIT[I] *100/ REALTIME ,OSTIME[I],OSTIME[I]*100/REALTIME));
     PROIDLESUM'PLUS'IDLETIME[I];PROJQWAITSUM'PLUS'PROJQWAIT[I]);
    OUTF(STANDOUT,$"AVG % TIME PROS IDLE IS"<2.1>"%"L$,
        (PROIDLESUM *100/ N / REALTIME ));
    OUTF(STANDOUT,$"AVG % TIME PROS IN JOB Q IS"<2.1>"%"L$,
          (PROJQWAITSUM *100/ N / JOBLISTIME ));
```

```
     PRINT(("CHANNEL TIME CHANNEL IDLE % TIME CHANNEL IDLE",NEWLINE));
     'FOR'I'TO'CHANO'DO'
      (OUTF(STANDOUT,$<1>6X,<8.3>8X,<3.1>"%"L$,(I,CHANNELIDLE[I],
             CHANNELIDLE[I] *100/ IOTIME ));
       CHANNELIDLESUM'PLUS'CHANNELIDLE[I]);
     OUTF(STANDOUT,$"AVG % TIME CHANNELS IDLE IS"<2.1>"%"L$,
         (CHANNELIDLESUM *100/ IOTIME  / CHANO));
'FOR'I'TO'JOBNEW-1'DO'IOSUM'PLUS'IOREQ[I];
OUTF(STANDOUT,$"AVC FILESTORE I/O PER SEC ="<3.1>L$,
      (IOSUM/REALTIME*1000));
     PRINT(("MEMORY USAGE",NEWLINE,"------------",NEWLINE));
     OUTF(STANDOUT,$
                       "AVG AMOUNT OF MEMORY UNUSED"<3>"K"L,
                       "AVG % OF MEMORY UNUSED"<2>"%"L ,
                       "AVG AMOUNT OF MEM USED FOR CORE IMAGES"<3>"K"L,
                         "AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS"<3>"K"L$,
         (AVGUNUSED,PERCENT,USERCOREAVG,SYSCOREAVG));
     PRINT(("JOB DESCRIPTIONS",NEWLINE,"----------------",NEWLINE));
     PRINT(("JOBNO JOBTYPE JOBTIME(MSECS) T/S'S    IOREQS    BLOCKTIME  SYSTEM TIME  TERMED",
NEWLINE));
     'FOR'I'TO'JOBNEW-1'DO'
      (OUTF(STANDOUT, $<3>3X,<2>2X,<8.3>2X ,<4>4X ,<5> ,<8.3> ,<8.3>3X$,
           (I,JOB[I,4],JBSIZE[I],TSEXPD[I] ,IOREQ[I],JBBLOCKED[I],SYSTIME[I]));
     PRINT(TERMINED[I]);
       JBLOCKSUM'PLUS'JBBLOCKED[I] ;
      'IF'WAITBLOCKED[I]'AND'IOREQ[I]>0
      'THEN'JBLOCKSUM'PLUS'REALTIME-JBBLOCKSTART[I];
             PRINT(("   -STILL BLOCKED",NEWLINE))
      'ELSE'PRINT(NEWLINE)
      'FI');
PRINT(NEWPAGE);
     PRINT(("JOBNO TIME JOB INITIATED   TIME JOB FINISHED     TIME IN SYSTEM",
              NEWLINE));
     'FOR'I'TO'JOBNEW-1'DO'
      'IF''NOT'('NOT'(JOBFINISH[I]<0.0)'AND''NOT'(0.0<JOBFINISH[I]))
      'THEN'OUTF(STANDOUT,$<3>3X,<8.3>2X,<8.3>7X,<8.3>L$,
           (I,JOBSTART[I],JOBFINISH[I],JOBFINISH[I]-JOBSTART[I]))
       'ELSE'OUTF(STANDOUT,$<3>3X,<8.3>3X"          --                --"L$,
                 (I,JOBSTART[I]))
        'FI'
'END'
'FINISH'

+VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
+VVVVVVV                           VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
+VVVVVVV       NUMBER OF PAGES   11     VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
+VVVVVVV                           VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
+VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV+
```

APPENDIX 13

LISTING OF INTERACTIVE PROGRAM FOR MODEL SPECIFICATION

```
MOP    :SSP8086_MOPNOUSERLF

LISTING OF :SSP8086.SCOPE-SPEC(21/)    PRODUCED ON 25AUG78 AT 11.01.34

#G8.63B AT ASTON    IN ':SSP8086_MOPNOUSERLF' ON 18MAY79 AT 14.09.34 USING  U15

DOCUMENT    SCOPE-SPEC

INTRO
'BEGIN'
'INT'PTR1:=0,PTR2,NO,EVENTNO,JOBSIZNO,JOBTYPNO,LIMIT:=22;
'BOOL'OK:='TRUE';
'STRING'VAL,SAME,S2,S1:="RJ MOPJOB1,UAALGOL68,PARAM(PROG            ,DATA OUTPUT,STORE 20000),JD(JT3000,MZ40K)";
'CHAR'CH;
'CHARPUT'INFIL,OUTFIL;
OBEYCOMMAND("AS *LP2,OUTPUT");OBEYCOMMAND("AS *FR1,INFO");
OPENC(INFIL,FILEREADER,1);OPENC(OUTFIL,LINEPRINTER,2);

PRINT((NEWLINE,"SCOPE ENTERED.",NEWLINE,"PLEASE PROVIDE THE SPECIFICATIONS FOR YOUR SYSTEM",NEWLINE,"
        [SEE SPECIFICATION VARIABLE LIST].",NEWLINE,"THE SPECIFICATION FILE MUST BE ALTERED SERIALLY,IF YOU REQUIRE CHANGES TO T -
    HE STANDARD SET ."
,NEWLINE,"N.B. USE THE TYPE OF VALUE GIVEN IN THE SPECIFICATION VARIABLE LIST"));
PRINT(NEWLINE);
PRINT(("PLEASE SUPPLY A TITLE FOR THIS EVALUATION EXPERIMENT",NEWLINE));
READ((NEWLINE,VAL));PUT(OUTFIL,(VAL,NEWLINE));
PRINT((NEWLINE,"DO YOU WISH TO ALTER ANY SYSTEM SPECIFICATIONS?-Y FOR YES,N FOR NO",NEWLINE));
READ((NEWLINE,CH));
'IF'CH="Y"
'THEN'
ALTER:'IF'OK
'THEN'PRINT(("PLEASE GIVE THE SERIAL NUMBER OF THE FIRST VARIABLE YOU WISH TO ALTER",NEWLINE));OK:='FALSE'
      'ELSE'PRINT(("PLEASE GIVE THE SERIAL NUMBER OF THE NEXT VARIABLE YOU WISH TO ALTER",NEWLINE))
        'FI';
READ((NEWLINE,PTR2));
'WHILE'PTR2>LIMIT'DO'
   (PRINT(("SERIAL NUMBER EXCEEDS LIMIT,PLEASE TRY AGAIN",NEWLINE));
    READ((NEWLINE,PTR2)));
'FOR'I'FROM'PTR1+1'TO'PTR2-1'DO'
 (GET(INFIL,(SAME,NEWLINE));
  PUT(OUTFIL,(SAME,NEWLINE)));
GET(INFIL,(VAL,NEWLINE));
PTR1:=PTR2;
PRINT(("THE STANDARD VALUE IS ",VAL,NEWLINE,
        "WHAT IS THE VALUE YOU REQUIRE?",NEWLINE));
READ((NEWLINE,VAL));
PUT(OUTFIL,(VAL,NEWLINE));
'IF'PTR2<LIMIT
'THEN'PRINT(("DO YOU WISH TO ALTER ANOTHER VARIABLE; 'Y' FOR YES,'N' FOR NO",NEWLINE));
READ((NEWLINE,CH));
'IF'CH="Y"
'THEN''GOTO'ALTER
'ELSE''FOR'I'FROM'PTR2+1'TO'LIMIT'DO'
        (GET(INFIL,(SAME,NEWLINE));
         PUT(OUTFIL,(SAME,NEWLINE)))
 'FI'
'FI'
'FI';
```

```
PRINT((NEWLINE,"PLEASE PROVIDE THE SPECIFICATIONS FOR YOUR WORKLOAD CHARACTERISTICS",NEWLINE));
PRINT(("WHAT IS YOUR INITIAL VALUE FOR SEED1?",NEWLINE));
READ((NEWLINE,VAL));
PUT(OUTFIL,(VAL,NEWLINE));
PRINT(("WHAT IS YOUR INITIAL VALUE FOR SEED2?",NEWLINE));
READ((NEWLINE,VAL));
PUT(OUTFIL,(VAL,NEWLINE));
PRINT(("HOW MANY JOB TYPES DO YOU REQUIRE ? [MAX - 20]",NEWLINE));
READ((NEWLINE,JOBTYPNO));PUT(OUTFIL,(JOBTYPNO,NEWLINE));
PRINT(("PLEASE GIVE THE PERCENTAGE OF EACH TYPE; TYPE(1)----TYPE(N)",
        NEWLINE));
'FOR'I'TO'JOBTYPNO'DO'
  (PRINT(("PERCENTAGE OF TYPE",I,NEWLINE));READ((NEWLINE,VAL));
   PUT(OUTFIL,(VAL,NEWLINE)));
PRINT(("HOW MANY AGGREGATE JOB SIZES DO YOU REQUIRE ?[MAX - 8]",NEWLINE));
READ((NEWLINE,JOBSIZNO));PUT(OUTFIL,(JOBSIZNO,NEWLINE));
PRINT(("PLEASE GIVE THE PERCENTAGE OF EACH JOB SIZE OCCURING AND THEN THE JOB SIZE ITSELF  (FOR EACH SIZE REQUIRED)",NEWLINE));
'FOR'I'TO'JOBSIZNO'DO'
    (PRINT(("PERCENTAGE OF JOBSIZE",I,NEWLINE));READ((NEWLINE,VAL));
     PUT(OUTFIL,(VAL,NEWLINE));
     PRINT(("SIZE OF JOBSIZE",I,NEWLINE));READ((NEWLINE,VAL));
     PUT(OUTFIL,(VAL,NEWLINE)));
PRINT(("HOW MANY EVENTS DO YOU WISH TO SIMULATE ?[MAX - 8]",NEWLINE));
READ((NEWLINE,EVENTNO));PUT(OUTFIL,(EVENTNO,NEWLINE));
PRINT(("PLEASE GIVE THE PROBABILITIES FOR THOSE EVENTS ,AS CALCULATED BY THE FOMULA GIVEN",NEWLINE,
"IN THE ORDER TIMESLICE,IO REQUEST,TERMINATION,ETC AS PROGRAMMED IN SEGMENT INSTRGEN",NEWLINE,
        "FOR EACH JOBTYPE IN TURN;TYPE(1)---TYPE(N).",NEWLINE));
'FOR'I'TO'JOBTYPNO'DO'(PRINT(("JOBTYPE",I,NEWLINE));
'FOR'J'TO'EVENTNO'DO'
  (PRINT(("EVENT",J,NEWLINE));READ((NEWLINE,VAL));
   PUT(OUTFIL,(VAL,NEWLINE))));
VAL:="****";PUT(OUTFIL,(VAL,NEWLINE));
PRINT(("YOUR SPECIFICATION FILE IS NOW COMPLETED AND THE SIMULATION RUN WILL BE STARTED  AUTOMATICALLY",NEWLINE,
        "WHAT IS THE NAME OF THE FILE WHICH HOLDS YOUR MODEL?",NEWLINE));
 READ((NEWLINE,S2));

S1[33:'UPB'S2+32]:=S2;
OBEYCOMMAND(S1)
'END'
'FINISH'
****

+QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ                                      QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ         NUMBER OF PAGES    2         QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQ                                      QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
+QQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQQ+
```

APPENDIX 14

RESULTS OF THE GEORGE 3 EXPERIMENT

Included in this appendix are two different listings of the George 3 experiment. The first listing is of the model described in Chapter 7. However, on completion of this model, it was discovered that the installation at the University of Aston includes a parameter "OBJQUOTA" which sets the maximum amount of memory available for core images and leaves the rest of memory for George chapters and red tape headings. This effectively reduced the memory management section of the model to a similar structure as the one adopted by the skeleton structure where FREECORE is set. Due to this discovery a new model was formulated reflecting this latter facility. However, both models are valid since the parameter OBJQUOTA is not always adhered to on the George 3 system and at certain times in the day the system runs as in the original George 3 system model.

The first listing is the original model and its results are comparable to some of those gained from the software monitor. In order that the reader can compare the results himself some actual performance figures from George 3 itself during March 1977 are given below:

	No. of coreimages	Amount of core OBJ	GEORGE	FREE	% of mill time OBJ	GEORGE	IDLE	input/output TRANSFERS/SEC.
1.	6	103	66	3	47	53	0	17.6
2.	7	104	65	3	47	54	0	12.0
3.	10	105	65	2	62	38	0	7.2
4.	8	103	65	4	24	76	0	20.8
5.	9	104	64	4	46	55	0	17.7
6.	8	106	61	5	32	65	4	22.0

A similar table of comparable performance figures when OBJQUOTA was in use is given below for comparison with the second listing

	No. of coreimages	Amount of core OBJ	GEORGE	FREE	% of mill time OBJ	GEORGE	IDLE	input/output TRANSFERS/SEC.
1.	5	63	105	4	44	56	0	19.8
2.	5	74	94	4	54	46	0	14.0
3.	4	74	95	3	53	47	0	12.8
4.	6	88	80	4	59	36	4	17.4
5.	8	70	97	5	42	58	0	22.8
6.	5	86	84	2	51	45	3	13.8

```
#####  #####  #####  #         TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
#   #  #   #  #   #  #         TTTT                      TTTTTTTTTTTT                                        TTTTTTTT
#   #  #   #  #   #  #         TTTT                      TTTTTTTTTTTT                                        TTTTTTTT
#####  #   #  #####  #####     TTTT          MOP         TTTTTTTTTTTT     :SSP8086,MOPGEO3                   TTTTTTTT
#   #  #   #  #   #  #   #     TTTT                      TTTTTTTTTTTT                                        TTTTTTTT
#   #  #   #  #   #  #   #     TTTT                      TTTTTTTTTTTT                                        TTTTTTTT
#####  #####  #####  #####     TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT

#LISTING OF :SSP8086,MOPGEO3(1/B180)    PRODUCED ON 21SEP78 AT 23.04.39

#G8.02S AT ASTON    IN ':SSP8086,MOPGEO3' ON 21SEP78 AT 23.04.56 USING  U14

DOCUMENT    :SSP8086,MOPGEO3(/B180)

STARTED :SSP8086,MOPGEO3,21SEP78 22.00.04 TYPE: BACK  10 ; STREAM D ; PRIORITY 1 ; ENTERED WELL 21SEP78 15.34.07
22.00.04→ RJ MOPGEO3,UAALGOL68,PARAM(PROG GEORGE3,DATA GEODATA,STORE 20000),JD(JT3000,MZ40K)
                         ****************************************************************************
                         *                                                                          *
                         *                          SOFTWARE CHANGES                                *
                         *                                                                          *
                         *        USERS ARE REMINDED THAT THE FOLLOWING SOFTWARE CHANGES TOOK       *
                         *        PLACE ON MONDAY 4TH SEPTEMBER 1978 :-                             *
                         *              1) UACLUSTANC WITHDRAWN                                     *
                         *              2) FSCE LIBRARY WITHDRAWN                                   *
                         *              3) EXPRESS SYSTEM MILL TIME LIMIT OF 20 SECONDS FOR         *
                         *                 COMPILATION , CONSOLIDATION AND EXECUTION INTRODUCED     *
                         *              4) EXPRESS SYSTEM OUTPUT LIMIT OF 500 LINES WITH A DEFAULT  *
                         *                 OF 100 LINES INTRODUCED.                                 *
                         *                                                                          *
                         *        FOR FURTHER INFORMATION SEE AUGUST 1978 NEWSLETTER (NO.18)        *
                         *                                                                          *
                         ****************************************************************************
22.00.05 JOB IS NOW FULLY STARTED
22.00.05→ UAALGOL68         PROG GEORGE3,DATA GEODATA,STORE 20000
22.00.05→          TA AB,CM,JL

   UAALGOL68: STATUS REPORT NUMBER 4      DATE 23 NOVEMBER 77

22.00.13 0.01 CORE GIVEN 2560
COMPILED BY THE ALGOL68-R COMPILING-SYSTEM   ON 21/09/78 AT 22/00/21        ASTON UNIVERSITY

22.00.24 0.02 CORE GIVEN 37888
   0    TIMESIM
   1    'BEGIN'
   2    'BOOL'CONTINUE:='TRUE',OK:='TRUE',GAPFOUND,OK1,BOOL,INTRUPTYPE,LASTJOBIN:='TRUE';
   3     [1:70 ]'REAL'JBSIZE,JBBLOCKED,JOBSTART,JOBFINISH,JBBLOCKSTART,SYSTIME;
   4    [1:100]'INT'IOQ;
   5    'REAL'RV,SAMPLEVAR:=50.0,PROIDLESUM:=0.0,CHANNELIDLESUM:=0.0,PROJQWAITSUM:=0.0,JBLOCKSUM:=0.0,
   6           SAMPLEPRINT:=50000.0,OSTOT,OBJTIME:=0.0,IDLETOT;
   7    'INT' SEED1:=11,SEED2:=7,IOQSIZE:=0,JQSIZE:=0,FREECORE,SAVE,COREIMAGE:=0,TOTAL,
   8           TENTSTART,TOTJOBSTART,NEWQUOTASIZ,JOBSINWELL:=5,CHAPCHNO:=0,
   9             CHAPTRANSNO:=0,RELOCNO:=0;
  10    'INT'JOBNEW ,JOBCOUNT:=0,X,NO,I,SIZ,BEG,FIN,HLSVAR,
  11          JOBNO,INTRUPPTR1:=1,INTRUPPTR2:=1,JOBLOAD:=0,JOBSIZE,IOSUM:=0,
  12        FRAGLEVEL:=0,NEXTJOB:=0,UNUSECORE:=0,FRAGCOUNT.=0,PERCENT,AVGUNUSED,
  13        CHA,NXTTYPE:=0,SLICEPROB,IOPROB,TERMPROB,JOBTYPE,
  14        SYSCORE:=0, SYSCORECOUNT:=0,USERCORE:=0,USERCORECOUNT:=0,
  15         USERCOREAVG,SYSCOREAVG;
  16
  17
```

```
        'FI');

'C'PLACES THE EVENT OF AN INTERRUPT ON A LIST 'C'

'PROC'INTRUPLIST=('REAL'TIME,'INT'JOBNO,'BOOL'BOOL,'BOOL'B)'VOID':
    ('IF'IHEAD'IS'NIL
    'THEN'IHEAD:=ITAIL:='INTRUPENT':=(TIME,BOOL,B,JOBNO,'NIL')
    'ELSE'CONTINUE:='TRUE';IPTR1:='NIL';IPTR2:=IHEAD;
          'WHILE'CONTINUE'AND'(NXT'OF'IPTR2'ISNT'NIL)'DO'
             'IF'TIME>=INTRUPTIME'OF'IPTR2
             'THEN'IPTR1:=IPTR2;IPTR2:=NXT'OF'IPTR2
             'ELSE'IPTR1'IS'NIL
             'THEN'IHEAD:='INTRUPENT':=(TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                                  (TIME,BOOL,B,JOBNO,IPTR2);
                   CONTINUE:='FALSE'
             'FI';
          'IF'NXT'OF'IPTR2'IS'NIL
          'THEN''IF'INTRUPTIME'OF'IPTR2<=TIME
                'THEN'ITAIL:=NXT'OF'ITAIL:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,'NIL')
                'ELSE'IPTR1'IS'NIL
                'THEN'IHEAD:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,IPTR2)
                'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                      (TIME,BOOL,B,JOBNO,IPTR2)
                'FI'
          'FI'
    'FI');

'C'PLACES AN ISO REQUEST ONTO A LIST WHICH IS SERVED BY THE CHANNELS'C'

'PROC'IOLIST=('INT'I)'VOID':
    ('IF'IOHEAD'IS'NOTHING
      'THEN'IOHEAD:=IOTAIL:='IOLISTENT':=
            (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
     'ELSE'CONTINUE:='TRUE';IOPTR1:='NIL';IOPTR2:=IOHEAD;
           'WHILE'CONTINUE'AND'(IONXT'OF'IOPTR2'ISNT'NOTHING)'DO'
            ('IF'PRTIME[I ]>=IOTIM'OF'IOPTR2
             'THEN'IOPTR1:=IOPTR2;IOPTR2:=IONXT'OF'IOPTR2
             'ELSE'IOPTR1'IS'NOTHING
             'THEN'IOHEAD:='IOLISTENT':=
                   (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                   CONTINUE:='FALSE'
             'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                   (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                   CONTINUE:='FALSE'
             'FI');
             'IF'IONXT'OF'IOPTR2'IS'NOTHING
           'THEN''IF'IOTIM'OF'IOPTR2<=PRTIME[I ]
                 'THEN'IOTAIL:=IONXT'OF'IOTAIL:='IOLISTENT':=
                       (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
                 'ELSE'IOPTR1'IS'NOTHING
              'THEN'IOHEAD:='IOLISTENT':=
                       (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                 'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                       (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                 'FI'
           'FI'
```

```
'FI');

'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
    IT WILL BE READY TO RUN'C'

'PROC'INSERT=('INT'I)'VOID';
   (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL
'IF'HEAD'IS'NULL
   'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],'NIL')
   'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
        'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
          ('IF'PRTIME[I ]>=TIME'OF'PTR2
           'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
           'ELSE'PTR1'IS'NULL
           'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                 CONTINUE:='FALSE'
           'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2);
                  CONTINUE:='FALSE'
           'FI');
           'IF'NEXT'OF'PTR2'IS'NULL
           'THEN''IF'TIME'OF'PTR2<=PRTIME[I ]
                 'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
                              (PR[I ],PRTIME[I ],'NIL')
                 'ELSE'PTR1'IS'NULL
                 'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
                 'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
                 'FI'
           'FI'
   'FI');

'C'INITIALISES ISO OPERATION AND BLOCKS JOB'C'

'PROC'INOUT=('INT'I)'VOID';
   (PRTIME[ ]:= PRTIME[ ]+CHANNELDEL;JOBLOCKSTART[PR[I]]:=PRTIME[I];
    SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;

IO[I]:='TRUE';WAITBLOCKED[PR[I ]]:='TRUE';
 INTRUPTYPE:='TRUE';
  IOLIST(I));


'C'INITIATES ROLLING IN OF A JOB'C'

'PROC'ROLLIN=('INT'I)'VOID';
      (JOBSTART[PR[I]]:=PRTIME[I];
    PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;
    SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='TRUE';
   WAITBLOCKED[PR[I ]]:='TRUE';
    IOLIST(I));


'C'ASSIGNS FIRST ENTRY ON READY QUEUE TO A PROCESSOR'C'

'PROC'PEALLOC=('INT'I)'VOID';
    ('IF'HEAD'IS'NULL
      'THEN'
          PR[I ]:=0
    'ELSE'PTR2:=HEAD)
```

```
                'IF'TIME'OF'PTR2> PRTIME[I ]
         'THEN'
                    PR[I ]:=0
                'ELSF'NEXT'OF'PTR2'ISNT'NULL
               'THEN'HEAD:=NEXT'OF'PTR2;
                    PR[I ]:=P'OF'PTR2
                'ELSE'HEAD:=TAIL:='NIL';
                    PR[I ]:=P'OF'PTR2
                'FI'
       'FI';
      PRTIME[I ]:=PRTIME[I ]+REALLOCDEL;
           'IF'PR[I]>0'THEN'OSTIME[I]'PLUS'REALLOCDEL'FI';JBLISTIME:=PRTIME[I ]);


'C'GENERATES JOB CHARACTERISTICS'C'

'PROC'JOBSIZGEN='VOID':
     ('IF'LASTJOBIN'THEN' NO:='ENTIER'(RANDI*100);
      'IF'NO<=TYPPERCENT[1]'THEN'JOBTYPE:=1
      'ELSF'NO<=TYPPERCENT[2]'THEN'JOBTYPE:=2
      'ELSF'NO<=TYPPERCENT[3]'THEN'JOBTYPE:=3
      'ELSF'NO<=TYPPERCENT[4]'THEN'JOBTYPE:=4
      'ELSF'NO<=TYPPERCENT[5]'THEN'JOBTYPE:=5
      'ELSF'NO<=TYPPERCENT[6]'THEN'JOBTYPE:=6
      'ELSF'NO<=TYPPERCENT[7]'THEN'JOBTYPE:=7
     'ELSF'NO<=TYPPERCENT[8]'THEN'JOBTYPE:=8
     'ELSF'NO<=TYPPERCENT[9]'THEN'JOBTYPE:=9
     'ELSF'NO<=TYPPERCENT[10]'THEN'JOBTYPE:=10
     'ELSF'NO<=TYPPERCENT[11]'THEN'JOBTYPE:=11
     'ELSF'NO<=TYPPERCENT[12]'THEN'JOBTYPE:=12
     'ELSF'NO<=TYPPERCENT[13]'THEN'JOBTYPE:=13
     'ELSF'NO<=TYPPERCENT[14]'THEN'JOBTYPE:=14
     'ELSF'NO<=TYPPERCENT[15]'THEN'JOBTYPE:=15
     'ELSF'NO<=TYPPERCENT[16]'THEN'JOBTYPE:=16
     'ELSF'NO<=TYPPERCENT[17]'THEN'JOBTYPE:=17
     'ELSF'NO<=TYPPERCENT[18]'THEN'JOBTYPE:=18
     'ELSF'NO<=TYPPERCENT[19]'THEN'JOBTYPE:=19
      'ELSE'JOBTYPE:=20
      'FI';
       NO:='ENTIER'(RANDI*100);
            'IF'NO<=SIZPERCENT[1]'THEN'JOBSIZE:=JOBSIZES[1]
            'ELSF'NO<=SIZPERCENT[2]'THEN'JOBSIZE:=JOBSIZES[2]
            'ELSF'NO<=SIZPERCENT[3]'THEN'JOBSIZE:=JOBSIZES[3]
            'ELSF'NO<=SIZPERCENT[4]'THEN'JOBSIZE:=JOBSIZES[4]
            'ELSF'NO<=SIZPERCENT[5]'THEN'JOBSIZE:=JOBSIZES[5]
           'ELSF'NO<=SIZPERCENT[6]'THEN'JOBSIZE:=JOBSIZES[6]
           'ELSF'NO<=SIZPERCENT[7]'THEN'JOBSIZE:=JOBSIZES[7]

            'ELSE'JOBSIZE:=JOBSIZES[8]
            'FI'
      'ELSE'LASTJOBIN:='TRUE'
      'FI');


'C'CALCULATES HOW MUCH CORE WOULD BE TAKEN UP WITH RED TAPE HEADINGS'C'


'PROC'REDTAPEQUOTA='INT':
    (QUOTA:=FREEPOOL+ESSENTFUNCT*(COREIMAGE*2.5)*(TENTSTART*2.0);
     'ROUND'(QUOTA));
```

```
'C'CALCULATES HOW MUCH CORE WOULD BE TAKEN UP WITH GEORGE CHAPTERS'C'

'PROC'CHAPQUOTA='INT';
    ('C'JOBSTART:=COREIMAGE+TENTSTART;
     'IF'TOTJOBSTART>10
     'THEN'QUOTA:=(TOTJOBSTART-10)*1.0+(5*1.5)+(5*2.0)
     'ELSE'TOTJOBSTART>5
     'THEN'QUOTA:=(TOTJOBSTART-5)*1.5+(5*2.0)
     'ELSE'QUOTA:=TOTJOBSTART*2.0
     'FI';
     'IF'COREIMAGE>5
     'THEN'QUOTA:=QUOTA+((COREIMAGE-5)*0.5)+(5*1.0)
     'ELSE'QUOTA:=QUOTA+COREIMAGE*1.0
     'FI';
     'ROUND'(QUOTA));


'C'DECIDES WHETHER OR NOT TO TENTATIVELY START A JOB'C'

'PROC'HLS=('INT'I)'VOID';
   (PRTIME[I]'PLUS'HLSDEL;OSTIME[I]'PLUS'HLSDEL;
    HLSVAR:=TENTSTART+40;
    'IF'HLSVAR<200'AND'JOBSINWELL>0
    'THEN'TENTSTART'PLUS'1;JOBSINWELL'MINUS'1
    'FI');


'C'ORGANISES QUEUE OF PROCESSORS WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'

'PROC'JQLIST=('INT'I)'VOID';
   ('IF'QHEAD'IS'NILL
    'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],'NIL')
    'ELSE'CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
          'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
          'DO'
           ('IF'PRTIME[I ]>=QTIME'OF'QPTR2
            'THEN'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
            'ELSF'QPTR1'IS'NILL
            'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],QPTR2);CONTINUE:='FALSE'
            'ELSE'QNEXT'OF'QPTR1:='JQENT'I:=(I,PRTIME[I ],QPTR2);
                  CONTINUE:='FALSE'
            'FI');
            'IF'QNEXT'OF'QPTR2'IS'NILL
            'THEN''IF'QTIME'OF'QPTR2<=PRTIME[I ]
                  'THEN'QNEXT'OF'QPTR2:='JQENT'I:=(I,PRTIME[I ],'NIL')
                  'ELSF'QPTR1'IS'NILL
                  'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],QPTR2)
                  'ELSE'QNEXT'OF'QPTR1:='JQENT'I:=(I,PRTIME[I ],QPTR2)
                  'FI'
            'FI'
            'FI';
             NOTENT[I]:='FALSE');


'C'INITIALISES SIMULATION VARIABLES AND STATE FROM WHICH SIMULATION BEGINS'C'

'PROC'INIT='VOID';
  (OK1:='TRUE';
```

```
'FOR'I'FROM'0'TO'70'DO'
  'FOR'J'TO'4'DO'JOB[I,J]:=0;

READ(TITLE);PRINT((TITLE,NEWLINE,"--------------------",NEWLINE));
PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(FREECORE);PRINT(("FREE CORE =",FREECORE,NEWLINE));
    READ(MEMSIZE);PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE));
    READ(N);PRINT(("NO OF PROCESSORS =",N,NEWLINE));
    READ(CHANO);PRINT(("NO OF CHANNELS =",CHANO,NEWLINE));
    READ(IODEL);OUTF(STANDOUT,$"I/O DELAY ="<3.1>L$,(IODEL));
    READ(REALLOCDEL);OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL));
    READ(INSERTDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL));
    READ(ROLLOUTDEL);OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL));
    READ(ROLLINDEL);OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL));
    READ(RELOCDEL);OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL));
    READ(CHANNELDEL);OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL));
    READ(GAPCHECKDEL);OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL));
    READ(INTRUPDEL);OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL));
    READ(SWAPQDEL);OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAP -
  QDEL));
    READ(HLSDEL);OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL));
    READ(LLSDEL);OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL));
    READ(ACCOUNTDEL);OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL));
    READ(SEGSIZE);PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE));
    READ(SEGNO);PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE));
    READ(CHAPCHDEL);OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL));
    READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
    READ(ESSENTFUNCT);PRINT(("MEMORY NEEDED FOR ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
    READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
    READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
    READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
    READ(TYPPERCENT[1]);PRINT(("% OF TYPE   1 =",TYPPERCENT[1],NEWLINE));
    'FOR'I'FROM'2'TO'JOBTYPNO'DO'
      (READ(TYPPERCENT[I]);
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
       TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1];
    READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
    READ(SIZPERCENT[1]);
    READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
    'FOR'I'FROM'2'TO'JOBSIZNO'DO'
      (READ(SIZPERCENT[I]);
       READ(JOBSIZES[I]);
      PRINT((SIZPERCENT[I],"% OF JOBS ARE",JOBSIZES[I],"K",NEWLINE));
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1];
    READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
    'FOR'I'TO'JOBTYPNO'DO'
    (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
      (READ(EVENTPROBS[I,J]);PRINT(EVEN.PROBS[I,J])));
PRINT(NEWPAGE);
    TENSTART:=4;
    'FOR'JOBNO'WHILE'OK1'AND'JOBLOAD#FREECORE'DO'
      (JOBSIZGEN;CORELMAGE'PLUS'1;FREECORE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
        'IF'FREECORE-JOBLOAD>=JOBSIZE
        'THEN'JOB[JOBNO,1]:=JOBSIZE;
              JOB[JOBNO,2]:=JOBLOAD+1;
              JOB[JOBNO,3]:=JOBSIZE+JOBLOAD;
               JOBLOCKSTART[JOBNO]:=0.0;
              JOB[JOBNO,4]:=JOBTYPE;
              PRINT(NEWLINE);
              JOBNEW:=JOBNO+1;
              JOBLOAD'PLUS'JOBSIZE
```

```
                    'ELSE'COREIMAGE'MINUS'1;
                     FREECORE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
                    FRAGLEVEL:=FREECORE-JOBLOAD;
                         JOBNEW:=JOBNO;
                         FRAGCOUNT'PLUS'1;UNUSECORE:=FRAGLEVEL;
                        GHEAD:=GTAIL:='GAPENTRY':=
                        (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL');
                        OK1:='FALSE';LASTJOBIN:='FALSE'
                  'FI';
                  'IF'JOBLOAD=FREECORE'THEN'GHEAD:=NULLUS;
                                            FRAGCOUNT'PLUS'1;
                                            UNUSECORE:=FRAGLEVEL
                  'FI');
              'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
            'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JBBLOCKED[I]:=0.0;TSEXPD[I]:=0;IOREQ[I]:=0;
                                  SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

             'FOR'I'TO'N'DO'(PR[I ]:=I;PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                              NEWJBOOL[I]:='FALSE';OSTIMESAMPLE[I]:=0.0;IDLESAMPLE[I]:=0.0;NOTENT[I]:='TRUE';
                              IO[I]:='FALSE';PROJQWAIT[I]:=0.0);
              'FOR'I'TO'70'DO'(NOTROLLEDOUT[I]:='FALSE';WAITBLOCKED[I]:='FALSE');
             IHEAD:=ITAIL:='NIL';
             QHEAD:='NIL';
             'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'PR[I]:=I;
             'IF'N>=JOBNEW-1
             'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;
                  HEAD:=TAIL:='NIL'
             'ELSE'HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
                  'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
             'FI');



          'C'KEEPS A CHECK OF ALL GAPS IN CORE'C'

          'PROC'GAPLIST=('INT'X,Y,Z)'VOID':
                (SIZ:=X;BEG:=Y;FIN:=Z;
              'IF'GHEAD'IS'NULLUS
             'THEN'GHEAD:=GTAIL:='GAPENTRY':=
                    (SIZ,BEG,FIN,'NIL')
             'ELSE'GPTR3:=GHEAD;CONTINUE:='TRUE';OK1:='FALSE';
          LOOPEND:'WHILE'CONTINUE'DO'
                   ('IF'BEGIN'OF'GPTR3=FIN+1
                    'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
                          BEGIN'OF'GPTR3:=BEG;
                          SIZ:=SIZE'OF'GPTR3;
                          BEG:=BEGIN'OF'GPTR3;
                          FIN:=END'OF'GPTR3;
                         'IF'OK1
                          'THEN'GPTR2:=GHEAD;GPTR1:='NIL';
                               'WHILE''IF'BEG=BEGIN'OF'GPTR2'AND'END'OF'GPTR2#FIN
                                     'THEN''IF'GPTR1'IS'NULLUS
                                           'THEN'GHEAD:=PTR'OF'GPTR2
                                           'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                           'FI'
                                     'FI';
                                     PTR'OF'GPTR2'ISNT'NULLUS
                               'DO'(GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
                          'FI';
                          OK1:='TRUE'
                    'ELSE'END'OF'GPTR3=BEG-1
                    'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
```

```
451                     END'OF'GPTR3;='FIN;
452                     SIZ:='SIZE'OF'GPTR3;
453                     BEG:='BEGIN'OF'GPTR3;
454                     FIN:='END'OF'GPTR3;
455                     'IF'OK1
456                          'THEN'GPTR2:='HEAD;GPTR1:='NIL';
457                               'WHILE''IF'FIN='END'OF'GPTR2'AND'BEGIN'OF'GPTR2#BEG
458                                    'THEN''IF'GPTR1'IS'NULLUS
459                                          'THEN'GHEAD:='PTR'OF'GPTR2
460                                          'ELSE'PTR'OF'GPTR1:='PTR'OF'GPTR2
461                                           'FI'
462                                    'FI'
463 ;                                  PTR'OF'GPTR2'ISNT'NULLUS
464                               'DO'(GPTR1:=GPTR2;GPTR2:='PTR'OF'GPTR2)
465                          'FI';
466                     OK1:='TRUE'
467               'FI';
468               'IF'PTR'OF'GPTR3'IS'NULLUS
469               'THEN'CONTINUE:='FALSE';
470                    'GOTO'LOOPEND
471               'FI';
472               GPTR3:='PTR'OF'GPTR3)F
473          'IF'OK1='FALSE'
474          'THEN'GTAIL:='PTR'OF'GTAIL:='GAPENTRY::=
475                    (SIZ,BEG,FIN,'NIL')
476          'FI'
477     'FI' );
478
479 'C'RELOCATES JOBS IN CORE'
480
481
482 'PROC'RELOC=('INT'I)'VOID';
483    (CONTINUE:='FALSE';PTR1:='HEAD'JOBLOAD:=0;
484     'IF'PTR1'ISNT'NULL'THEN'
485     'WHILE'NEXT'OF'PTR1'ISNT'NULL'DO'
486       (JOBNO:='P'OF'PTR1;
487        JOB[JOBNO,2]:=JOBLOAD+1;
488        JOBLOAD'PLUS'JOB[JOBNO,1];
489        JOB[JOBNO,3]:=JOBLOAD;
490        PTR1:='NEXT'OF'PTR1));
491     'IF'NEXT'OF'PTR1'IS'NULL
492     'THEN'JOBNO:='P'OF'PTR1;
493          JOB[JOBNO,2]:=JOBLOAD+1;
494          JOBLOAD'PLUS'JOB[JOBNO,1];
495          JOB[JOBNO,3]:=JOBLOAD
496     'FI'
497     'FI';
498     'FOR'I'TO'N-1'DO'
499       ('IF'PR[I ]JO'AND''NOT'NEWJROOL[I]
500        'THEN'JOBNO:=PR[I ];
501             JOB[JOBNO,2]:=JOBLOAD+1;
502              JOBLOAD'PLUS'JOB[JOBNO,1];
503             JOB[JOBNO,3]:=JOBLOAD
504        'FI'));
505     'FOR'JOBNO'TO'70'DO'
506         'IF'NOTROLLEDOUT[JOBNO]'OR'WAITBLOCKED[JOBNO]
507         'THEN'JOB[JOBNO,2]:=JOBLOAD+1;
508               JOBLOAD'PLUS'JOB[JOBNO,1];
509               JOB[JOBNO,3]:=JOBLOAD
510         'FI';
511      'FOR'CHA'TO'CHANO'DO'
512         CHANNEL[CHA]'PLUS'RELOCBEL;
```

```
513         'WHILE'INTRUPTIME'OF'IPTR2'PLUS'RELOCDEL;
514             NXT'OF'IPTR2'ISNT'NIL'DO'
515               IPTR2:=NXT'OF'IPTR2
516       PRTIME[I ]'PLUS'RELOCDEL;STIME[I]'PLUS'RELOCDEL;
517       REALTIME'PLUS'RELOCDEL;
518       'IF'JBLISTIME<REALTIME
519       'THEN'JBLISTIME:=REALTIME
520       'FI';
521           'FOR'I'TO'N'DO'
522         ('IF'PRTIME[I ]<REALTIME
523          'THEN'PRTIME[I ]:=REALTIME
524          'FI'));
525
526
527   'C'DETERMINES THE AMOUNT OF CORE LEFT FOR CORE IMAGESOF JOBS'C'
528
529
530   'PROC'FREECOREADJ='VOID';
531       (SAVE:=MEMSIZE-(CHAPQUOTA+REDTAPEQUOTA);
532       'IF'SAVE>FREECORE
533       'THEN'FRAGLEVEL'PLUS'SAVE-FREECORE;
534             GAPLIST(SAVE-FREECORE,FREECORE+1,SAVE);FREECORE:=SAVE
535       'ELSE'SAVE<FREECORE
536       'THEN''IF'GHEAD'ISNT'NULLUS
537            'THEN'GPTR2:=GHEAD;GPTR1:='NIL';CONTINUE:='TRUE';
538                 'WHILE''IF'END'OF'GPTR2=FREECORE
539
540                         'THEN'CONTINUE:='FALSE';
541                               'IF'BEGIN'OF'GPTR2<SAVE
542                               'THEN'END'OF'GPTR2:=SAVE;
543                           SIZE'OF'GPTR2'MINUS'FREECORE-SAVE;
544                                     FRAGLEVEL'MINUS'FREECORE-SAVE
545                                    ;FREECORE:=SAVE
546                                'ELSE'SAVE:=BEGIN'OF'GPTR2;
547                                       FRAGLEVEL'MINUS'FREECORE-BEGIN'OF'GPTR2;
548                                      FREECORE:=SAVE;
549                                      'IF'GPTR1'IS'NULLUS
550                                      'THEN''IF'PTR'OF'GPTR2'IS'NULLUS
551                                            'THEN'GHEAD:='NIL'
552                                            'ELSE'GHEAD:=PTR'OF'GHEAD
553                                            'FI'
554                                      'ELSE''IF'PTR'OF'GPTR2'IS'NULLUS
555                                            'THEN'PTR'OF'GPTR1:='NIL';GTAIL:=GPTR1
556                                             'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
557                                             'FI'
558                                       'FI'
559                               'FI'
560                          'FI';
561                          CONTINUE'AND'(PTR'OF'GPTR2'ISNT'NULLUS)'DO'
562                          (GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
563            'FI'
564        'FI');
565
566
567   'C'CHECKS TO SEE IF ENOUGH SPACE IN CORE FOR A JOB TO BE ROLLED IN'C'
568
569    'PROC'SPACEINCORE=('INT'I)'BOOL';
570           ('IF'GHEAD'ISNT'NULLUS
571            'THEN'      GPTR1:='NIL';GPTR2:=GHEAD;
572             CONTINUE:='TRUE';GAPFOUND:='FALSE';
573             'WHILE'CONTINUE'AND'(PTR'OF'GPTR2'ISNT'NULLUS)'DO'
574               'IF'SIZE'OF'GPTR2>=JOBSIZE
```

```
575                 'THEN'CONTINUE:='FALSE';GAPFOUND:='TRUE';
576                       'IF'GPTR1'IS'NULLUS
577                       'THEN'GHEAD:=PTR'OF'GPTR2
578                       'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
579                       'FI'
580                 'ELSE'GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2
581                 'FI';
582                 'IF'PTR'OF'GPTR2'IS'NULLUS
583                 'THEN''IF'SIZE'OF'GPTR2>=JOBSIZE
584                       'THEN'GAPFOUND:='TRUE';
585                             'IF'GPTR1'IS'NULLUS
586                             'THEN'GHEAD:=GTAIL:='NIL'
587                             'ELSE'GTAIL:=GPTR1;PTR'OF'GTAIL:='NIL'
588                             'FI'
589                       'FI'
590                 'FI'
591                'ELSE'GAPFOUND:='FALSE'
592               'FI';
593                PRTIME[I]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
594                GAPFOUND);
595
596     'C'INITIATES SELECTION OF AND ROLLING IN OF NEW JOBS'C'
597
598
599     'PROC'NEWJOB=('INT'I)'VOID':
600          (          OK:='TRUE';
601                'WHILE'TENTSTART#0'AND'OK'DO'
602                  (PR[I]:=JOBNEW(JOBSIZGEN);
603                'IF'SPACEINCORE(I)'AND'TENTSTART>0
604                   'THEN'FRAGLEVEL'MINUS'JOBSIZE;
605                    JOBLOAD'PLUS'JOBSIZE;
606                    JOB[JOBNEW,1]:=JOBSIZE;
607                    JOB[JOBNEW,2]:=BEGIN'OF'GPTR2  ;
608                    JOB[JOBNEW,3]:=JOBSIZE+BEGIN'OF'GPTR2-1;
609                    JOB[JOBNEW,4]:=JOBTYPE;
610                    'IF'JOB[JOBNEW,3]<END'OF'GPTR2
611                    'THEN''IF'GHEAD'IS'NULLUS
612                          'THEN'GHEAD:=GTAIL:='GAPENTRY':=
613                                (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1,
614                                 END'OF'GPTR2,'NIL')
615                          'ELSE'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
616                                (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1 ,
617                                 END'OF'GPTR2,'NIL')
618                          'FI'
619                    'FI';
620                     ROLLIN(I);TENTSTART'MINUS'1;COREIMAGE'PLUS'1;PRTIME[I]'PLUS'LLSDEL;
621                          OSTIME[I]'PLUS'LLSDEL;
622                    JOBNEW'PLUS'1
623               'ELSE'FRAGLEVEL>=JOBLOAD'AND'TENTSTART>0
624              'THEN'RELOC(I);PRINT(("RELOCATION NECESSARY",NEWLINE));
625                    RELOCNO'PLUS'1;
626                    FRAGLEVEL'MINUS'JOBSIZE;
627                    JOBLOAD'PLUS'JOBSIZE;
628                    JOB[JOBNEW,1]:=JOBSIZE;
629                    JOB[JOBNEW,2]:=JOBLOAD+1-JOBSIZE;
630                    JOB[JOBNEW,3]:=JOBLOAD;
631                    JOB[JOBNEW,4]:=JOBTYPE;
632                    'IF'FRAGLEVEL#FREECORE-JOBLOAD
633                    'THEN'FRAGLEVEL:=FREECORE-JOBLOAD;
634                          PRINT("MISTAKE IN GAPTABLE")
635                    'FI';
636                     'IF'FRAGLEVEL>0
```

```
637                    'THEN'                    GHEAD:=GTAIL:='GAPENTRY';=
638                   (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL')
639                   'ELSE'GHEAD:=GTAIL:='NIL'
640                    'FI';
641                   ROLLIN(I);CORE!MAGE'PLUS'I;TENTSTART'MINUS'I;PRTIME[I]'PLUS'LLSDEL;
642                             OSTIME[I]'PLUS'LLSDEL;
643                   JOBNEW'PLUS'I
644                  'ELSE'LASTJOBIN:='FALSE';OK:='FALSE'
645                'FI');
646           FREECOREADJ);
647
648
649    'C'CHECKS IF THERE ARE ANY INTERRUPTS THAT HAVE YET TO BE DEALT WITH
650         AFTER PROCESSOR HAS JUST DEALT WITH ONE'C'
651
652    'PROC'INTRUPCHECK=('INT'I)'VOID';
653       ('IF'IHEAD'ISNT'NIL
654        'THEN''IF'INTRUPTIME'OF'IHEAD<=PRTIME[I ]
655              'THEN'PR[I]:=-I
656              'ELSE'NEWJBOOL[I]:='TRUE';JOLIST(I)
657              'FI'
658         'ELSE'NEWJBOOL[I]:='TRUE';JOLIST(I)
659         'FI');
660    'PROC'INTERUPT=('INT'I)'VOID';
661          ('IF'NOTENT[I]'THEN'PRTIME[I]:=REALTIME
662       'ELSE''IF'PRTIME[I ]>JBLISTIME
663                              'THEN'JBLISTIME:=PRTIME[I ]
664             'ELSE''IF'PRLI]#0'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I] 'FI';
665                              PRTIME[I ]:=JBLISTIME
666           'FI';
667          NOTENT[I]:='TRUE';
668          CONTINUE;='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
669          'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
670          'DO'('IF'PRO'OF'QPTR2=I
671               'THEN''IF'QPTR1'IS'NILL
672                     'THEN'QHEAD:=QNEXT'OF'QHEAD
673                    'ELSE'QNEXT'OF'QPTR1:=QNEXT'OF'QPTR2
674                    'FI';
675                    CONTINUE;='FALSE'
676               'ELSE'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
677               'FI');
678               'IF'QNEXY'OF'QPTR2'IS'NILL
679               'THEN''IF'PRO'OF'QPTR2=I
680                     'THEN''IF'QPTR1'IS'NILL
681                           'THEN'QHEAD:='NIL'
682                           'ELSE'QNEXT'OF'QPTR1:='NIL'
683                           'FI'
684                     'FI'
685               'FI'
686       'FI';
687       'IF'PR[I ]>0
688       'THEN''IF'NEWJBOOL[I]
689             'THEN'NEWJBOOL[I]:='FALSE'
690             'ELSE'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
691                         OSTIME[I]'PLUS'INSERTDEL
692             'FI'
693       'FI';
694    'IF''INTRUPTYP'OF'IHEAD
695          'THEN''IF'ID'OF'IHEAD
696           'THEN'PRTIME[I]'PLUS'SWAPDEL;OSTIME[I]'PLUS'SWAPDEL;
697                 JBLISTIME'PLUS'SWAPDEL;
698                 PR[I]:=JOBNO'OF'IHEAD;
```

```
                    JRBLOCKED[PR[I]]'PLUS'PRTIME[I]-JRBLOCKSTART[PR[I]];
                  WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
              'ELSE'PR[I]:=JORNO'OF'IHEAD;
                    WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
            'FI';
          IHEAD:=NXT'OF'IHEAD
    'ELSE'JOBNO:=JORNO'OF'IHEAD;
         NOTROLLEDOUT[JOBNO]:='FALSE';
          FRAGLEVEL'PLUS'JOB[JORNO,1];
          JOBLOAD'MINUS'JOB[JOBNO,1];
              COREIMAGE'MINUS'1;
          GAPLIST(JOB[JOBNO,1],JOB[JOBNO,2],JOB[JOBNO,3]);
         PRTIME[I ]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
         JOB[JOBNO,2]:=JOB[JOBNO,3]:=0;OSTIME[I]'PLUS'INTRUPDEL;
              PRTIME[I ]'PLUS'INTRUPDEL;
            NEWJOB(I);IHEAD:=NXT'OF'IHEAD
       'FI');

'C'WHEN A JOB TERMINATES THIS ROUTINE SETS UP THE ROLING OUT OF THAT JOB'C'

'PROC'ENDJOB=('INT'I)'VOID';
   (NOTROLLEDOUT[PR[I ]]:='TRUE';
    PRTIME[I ]'PLUS'CHANNELDEL;SYSTIME[PR[I]]'PLUS'CHANNELDEL;
      OSTIME[I]'PLUS'CHANNELDEL;
    IO[I]:='FALSE';
    INTRUPTYPE:='FALSE';    IOLIST(I));

'C'NOTES SIZE OF QUEUE WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'


 'PROC'JQSIZEPROBE='VOID';
   ('IF'QHEAD'IS'NILL
    'THEN'PRINT((NEWLINE,"JQSIZE IS",JQSIZE,NEWLINE))
    'ELSE'    QPTR2:=QHEAD;
    'WHILE''IF'QTIME'OF'QPTR2'<='REALTIME'THEN'JQSIZE'PLUS'1'FI';QNEXT'OF'QPTR2'ISNT'NILL'DO'
       QPTR2:=QNEXT'OF'QPTR2 ;
    PRINT((NEWLINE,"JQSIZE IS",JQSIZE ,NEWLINE));
    PRINT(NEWLINE);
    JQSIZE:=0
    'FI');


'C'
   NOTES SIZE OF IO REQUEST QUEUE'C'

'PROC'IOQPROBE='VOID';
      ('IF'IOHEAD'IS'NOTHING
     'THEN'PRINT((NEWLINE,"IQSIZE IS",IOQSIZE,NEWLINE))
     'ELSE'IOPTR2:=IOHEAD;
    'WHILE''IF'IOTIME'OF'IOPTR2'<='REALTIME'THEN'IOQSIZE'PLUS'1'FI';IONXT'OF'IOPTR2'ISNT'NOTHING'DO'
      IOPTR2:='IONXT'OF'IOPTR2 ;
     PRINT((NEWLINE,"IQSIZE IS",IOQSIZE ,NEWLINE));
     PRINT(NEWLINE);
     IOQSIZE:=0
    'FI');

'C'GENERATES THE NEXT EVENT THAT IS TO OCCUR WITHIN EACH PARTICULARJOB'THAT
      IS RUNNING'C'
```

```
761
762
763     'PROC'INSTRGEN=('INT'I,JOBTYPE)'VOID';
764     (PRTIME[I]:=REALTIME;
765        'IF'PR[I ]>0
766        'THEN'NO:='ENTIER'(RAND2*100000);
767                 JOBSIZE[PR[I] ]'PLUS'1.0;PRTIME[I]'PLUS'1.0;
768              'IF'TOTAL:=EVENTPROBS[JOBTYPE,1];NO<TOTAL
769              'THEN'TSEXPD[PR[I]]'PLUS'1;
770                    JQLIST(I)
771              'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,2];NO<TOTAL
772              'THEN'IOREQ[PR[I]]'PLUS'1;
773                    INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
774              'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,3];NO<TOTAL
775              'THEN'JOBCOUNT'PLUS'1;TERMINED[PR[I]]:='TRUE';PRINT(("JOB",PR[I],"TERMED",NEWLINE));
776                    NEWJBOOL[I]:='TRUE';PRTIME[I]'PLUS'ACCOUNTDEL;HLS(I);ENDJOB(I);JQLIST(I)
777              'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,4];NO<TOTAL
778               'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
779                    OSTIME[I]'PLUS'CHAPCHDEL;
780                  CHAPCHNO'PLUS'1
781
782               'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,5];NO<TOTAL
783               'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
784                  CHAPTRANSNO'PLUS'1;OSTIME[I]'PLUS'CHAPCHDEL;
785                    INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
786              'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,6];NO<TOTAL
787               'THEN'INSERT(I);JRLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
788                     NEWJBOOL[I]:='TRUE';
789                    JOBSINWELL'PLUS'1;HLS(I);
790                    JQLIST(I)
791              'FI'
792        'ELSF'PR[I ]=0'THEN'NEWJBOOL[I]:='TRUE'JQLIST(I)
793        'FI');
794
795
796
797
798     'C'
799          DETERMINES TIME OF NEXT EVENT THROUGHOUT THE SYSTEM'C'
800
801     'PROC'NXTEVENTIME='VOID';
802        (TEMP:=SAMPLEVAR;
803         'FOR'I'TO'N'DO'
804            'IF'NOTENT[I]'AND'PR[I ]>0
805            'THEN''IF'TEMP>PRTIME[I ]
806                  'THEN'TEMP:=PRTIME[I ]
807                  'FI'
808            'FI';
809         'IF'CHANNELTIME<TEMP>IOTIME
810         'THEN'TEMP:=IOTIME
811         'FI';
812
813         'IF'TEMP>JRLISTIME
814         'THEN''IF'HEAD'ISNT'NULL
815               'THEN''IF'TIME'OF'HEAD<=JRLISTIME
816                     'THEN'TEMP:=JRLISTIME
817                     'FI'
818               'FI';
819              'FOR'I'TO'N'DO'
820              'IF''NOT'NEWJBOOL[I]'AND''NOT'NOTENT[I]'THEN'TEMP:=JRLISTIME'FI'
821         'FI';
822         'IF'HEAD'ISNT'NIL
```

```
823         'THEN'
824         'IF'TEMP>INTRUPTIME'OF'IOHEAD
825         'THEN'TEMP:=INTRUPTIME'OF'IOHEAD
826         'FI'
827         'FI');
828
829
830
831     'C' **MAIN BODY**
832
833          ORGANISES RUNNING OF JOBS,QUEUE AROUND CRITICAL REGION ,ISO QUEUE,
834          RESULTS TO BE PRINTED OUT DURING RUN,TIME-KEEPING,INTERRUPTS,RESULTS AT END OF RUN'C'
835
836
837     INIT;
838     'WHILE'JOBCOUNT# 12'DO'
839       ('FOR'I'TO'N'DO'
840         ('IF'PRTIME[I]<=REALTIME'AND'PR[I ]=0 'THEN'IDLETIME[I]'PLUS'1.0'FI'
841          'IF'NOTENT[I]'AND'PR[I ]>=0'THEN'
842           'IF'PRTIME[I ]<=  REALTIME'THEN'INSTRGEN(I,JOB[PR[I ],4])'FI'
843           'FI');
844         'IF'QHEAD'ISNT'NILL
845         'THEN''IF'REALTIME>=QTIME'OF'QHEAD'AND'REALTIME>=JRLISTIME
846               'THEN'I:=PRO'OF'QHEAD;
847                     'IF'PRTIME[I]>JRLISTIME
848                      'THEN'JRLISTIME:=PRTIME[I]
849                      'ELSE''IF'PR[I]#0
850                            'THEN'PROJQWAIT[I]'PLUS'JRLISTIME-PRTIME[I]
851                             'FI';
852                            PRTIME[I]:=JRLISTIME
853                       'FI';
854                     NOTENT[I]:='TRUE';
855                     'IF'NEWJBOOL[I]
856                     'THEN'REALLOC(I);NEWJBOOL[I]:='FALSE'
857                     'ELSE'INSERT(I);REALLOC(I)
858                     'FI';
859                     'IF'QNEXT'OF'QHEAD'ISNT'NILL
860                     'THEN'QHEAD:=QNEXT'OF'QHEAD
861                     'ELSE'QHEAD:='NIL'
862                     'FI'
863               'FI'
864         'FI';
865
866         'FOR'I'TO'CHANO'DO'
867        ('IF'CHANNEL[I]<REALTIME
868         'THEN'CHANNELIDLE[I]'PLUS'REALTIME-CHANNEL[I];
869               CHANNEL[I]:=REALTIME
870         'FI';
871           'IF'IOHEAD'ISNT'NOTHING
872           'THEN''IF'CHANNEL[I]<=REALTIME'AND'IOTIM'OF'IOHEAD<=REALTIME
873                 'THEN''IF'IO'OF'IOHEAD
874                       'THEN'IOTIM'OF'IOHEAD:=REALTIME+IODE[I]
875                             INTRUPLIST[IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
876                                        INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
877                            CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
878                       'ELSE'IOTIM'OF'IOHEAD:=REALTIME+POLLINDEL;
879                             INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
880                                        INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
881                             'IF''NOT'INTRUPTYPE'OF'IOHEAD
882                              'THEN'JOBFINISH[JOB'OF'IOHEAD]:=IOTIM'OF'IOHEAD
```

```
885                         CHANNEL[I];='IOTIM'OF'IOHEAD;IOHEAD;='IONXT'OF'IOHEAD
886                   'FI'
887                'FI'
888          'FI');
889          'IF'REALTIME>=SAMPLEVAR
890          'THEN'SAMPLEVAR'PLUS'50.0 ;UNUSECORE'PLUS'FRAGLEVEL[FRAGCOUNT'PLUS'1;
891                  USERCORE'PLUS'JOBLOAD;USERCORECOUNT'PLUS'1;
892                   SYSCORE'PLUS'MEMSIZE-FREECORE;SYSCORECOUNT'PLUS'1
893          'FI';
894          'IF'REALTIME>=SAMPLEPRINT
895          'THEN'SAMPLEPRINT'PLUS'50000.0;
896                OSTOT:=0.0;IDLETOT:=0.0;
897                'FOR'I'TO'N'DO'
898                  (OSTOT'PLUS'(OSTIME[I]-OSTIMESAMPLE[I])/50000*100;
899                IDLETOT'PLUS'(IDLETIME[I]-IDLESAMPLE[I])/50000*100;
900                     OSTIMESAMPLE[I];=OSTIME[I];IDLESAMPLE[I]:=IDLETIME[I]);
901                  PRINT(NEWLINE);
902                  PRINT("    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE",
903                    NEWLINE);
904                      OUTF(STANDOUT,$<8.1>,<2.1>3X,<2.1>7X,<2.1>4X,<3>4X,<3>4X,<3>L$,
905                  (REALTIME,OSTOT/N,IDLETOT/N,100-(OSTOT/N+IDLETOT/N),JOBLOAD,
906                          FRAGLEVEL,MEMSIZE-FREECORE));PRINT(NEWLINE);
907                   PRINT("CUREIMAGE TENTSTART JOBSINWELL",NEWLINE));
908                    PRINT(CUREIMAGE,TENTSTART,JOBSINWELL,NEWLINE,NEWLINE))
909            'FI';
910           REALTIME'PLUS'1.0;
911          N/TEVENTIME;
912          'IF'REALTIME<TEMP
913          'THEN''FOR'I'TO'N'DO'
914                  'IF'PR[I ]=0
915                  'THEN'IDLETIME[I]'PLUS'TEMP-REALTIME
916                   'FI';
917                 REALTIME:=TEMP
918          'FI';
919          'IF'IHEAD'ISNT'NIL
920          'THEN''IF'REALTIME>=INTRUPTIME'OF'IHEAD'AND'PRTIME[N ]<=REALTIME
921                  'THEN'INTERUPT(N);INTRUPCHECK(N)
922                'FI'
923          'FI');
924          PRINT(NEWPAGE);
925          AVGUNUSED:=UNUSECORE'/'FRAGCOUNT;USERCOREAVG:=USERCORE'/'USERCORECOUNT;
926          SYSCOREAVG:=SYSCORE'/'SYSCORECOUNT;PERCENT:=AVGUNUSED*100'/'MEMSIZE;
927          PRINT(NEWLINE);
928          OUTF(STANDOUT,$"REALTIME IS ="<8.3>2L$,(REALTIME));
929         PRINT("TOTAL NO OF CHAPCH IS -",CHAPCHNO,NEWLINE,
930                  "TOTAL NO OF CHAPTRANS IS -",CHAPTRANSNO,NEWLINE));
931     OUTF(STANDOUT,$"TOTAL CHAPCH+TRANS PER SEC ="<5.1>1$,
932          ((CHAPCHNO+CHAPTRANSNO)/REALTIME*100));
933     OUTF(STANDOUT,$"PERCENT REQ TRANS ="<2 1>L$,
934          (CHAPTRANSNO/CHAPCHNO*100));
935          PRINT("PROCESSOR USAGE",NEWLINE,"---------------",NEWLINE));
936          PRINT("PRO   PROIDLETIME %TIME PRO IDLE WAIT IN JOB Q %TIME IN JOB Q    OSTIME     % TIME IN OS",NEWLINE));
937          'FOR'I'TO'N'DO'
938           (OUTF(STANDOUT,$<I> ,<8,3>8X,<2.1>3X,<8.3>6X,<2.1>8X,<8.3>4X,<2,1>L$,
939                (I,IDLETIME[I],IDLETIME[I] *100/ REALTIME ,PROJQWAIT[I],
940                      PROJQWAIT[I] *100/ REALTIME ,OSTIME[I],OSTIME[I]*100/REALTIME));
941           PROIDLESUM'PLUS'IDLETIME[I];PROJQWAITSUM'PLUS'PROJQWAIT[I]);
942          OUTF(STANDOUT,$"AVG % TIME PROS IDLE IS"<2.1>"%"L$,
943              (PROIDLESUM *100/ N / REALTIME ));
944          OUTF(STANDOUT,$"AVG % TIME PROS IN JOB Q IS"<2.1>"%"L$,
945               (PROJQWAITSUM *100/ N / JOBTIME ));
946          PRINT("CHANNEL USAGE",NEWLINE,"-------------",NEWLINE));
```

```
PRINT(("CHANNEL TIME CHANNEL IDLE % TIME CHANNEL IDLE",NEWLINE));
 'FOR'I'TO'CHANO'DO'
  (OUTF(STANDOUT,$<1>3X,<8.3>8X,<2.1>"%"L$,(I,CHANNELIDLE[I],
         CHANNELIDLE[I] *100/ IOTIME ));
   CHANNELIDLESUM'PLUS'CHANNELIDLE[I]);
 OUTF(STANDOUT,$"AVG % TIME CHANNELS IDLE IS"<2.1>"%"L$,
     (CHANNELIDLESUM *100/ IOTIME  / CHANO));
'FOR'I'TO'JOBNEW-1'DO'IOSUM'PLUS'IOREQ[I];
  OUTF(STANDOUT,$"AVG FILESTORE I/O PER SEC ="<3.1>L$,
     (IOSUM/REALTIME*1000));
 PRINT(("MEMORY USAGE",NEWLINE,"------------",NEWLINE));
 OUTF(STANDOUT,$"NO OF MEMORY RELOCATIONS="<3>L,
              "AVG AMOUNT OF MEMORY UNUSED"<3>"K"L,
              "AVG % OF MEMORY UNUSED"<2>"%"L ,
              "AVG AMOUNT OF MEM USED FOR CORE IMAGES"<3>"K"L,
               "AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS"<3>"K"L$,
    (RELOCNO,AVGUNUSED,PERCENT,USERCOREAVG,SYSCOREAVG));
 PRINT(("JOB DESCRIPTIONS",NEWLINE,"----------------",NEWLINE));
 PRINT(("JOBNO JOBTYPE JOBTIME(MSECS) T/S'S    IOREQS    BLOCKTIME  SYSTEM TIME  TERMED",
NEWLINE));
 'FOR'I'TO'JOBNEW-1'DO'
  (OUTF(STANDOUT, $<3>3X,<2>2X,<8.3>3X ,<4>3X ,<5>3X ,<8.3> ,<8.3>$,
      (I,JOB[I,4],JBSIZE[I],TSEXPD[I] ,IOREQ[I],JBBLOCKED[I],SYSTIME[I]));
 PRINT((" ",TERMINED[I]));
  JBLOCKSUM'PLUS'JBBLOCKED[I] ;
 'IF'WAITBLOCKED[I]'AND'IOREQ[I]>0
 'THEN'JBLOCKSUM'PLUS'REALTIME-JBBLOCKSTART[I];
        PRINT(("   -STILL BLOCKED",NEWLINE))
 'ELSE'PRINT(NEWLINE)
 'FI');
 PRINT(NEWPAGE);
 PRINT(("JOBNO TIME JOB INITIATED   TIME JOB FINISHED     TIME IN SYSTEM",
        NEWLINE));
 'FOR'I'TO'JOBNEW-1'DO'
  'IF''NOT'('NOT'(JOBFINISH[I]<0.0)'AND''NOT'(0.0<JOBFINISH[I]))
  'THEN'OUTF(STANDOUT,$<3>3X,<8.3>8X,<8.3>7X,<8.3>L$,
      (I,JOBSTART[I],JOBFINISH[I],JOBFINISH[I]-JOBSTART[I]))
  'ELSE'OUTF(STANDOUT,$<3>3X,<8.3>3X"          --                    --"L$,
           (I,JOBSTART[I]))
   .  'FI'
'END'
'FINISH'
0.25 :HALTED :    COMPILED TIMESIM
22.03.18 FREE *DA11,161 TRANSFERS
22.03.21 FREE *LPO ,991 TRANSFERS
22.03.25 0.26 CORE GIVEN 36160,CLOCKED 0.20
DISPLAY : LD
25.30 :DELETED : OK
23.04.36 FREE *FHO ,26 TRANSFERS
23.04.36 FREE *DA12,111 TRANSFERS
23.04.36 FREE *DA11,77 TRANSFERS
23.04.36 FREE *TRO ,98 TRANSFERS
23.04.36 FREE *LPO ,371 TRANSFERS
23.04.37 FREE *DA15,8 TRANSFERS
23.04.37 25.30 DELETED,CLOCKED 25.23
PARAMETER NUMBER 1,2 UNACCESSED
DISPLAY: UAALGOL6RR: NORMAL EXIT
23.04.38 25.31 FINISHED : 2 LISTFILES
23.04.38 JOBTIME USED 1531 : MAXIMUM CORE USED 37384
23.04.38 JOB UNITS 3368

TTTTTTTT                                     TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTT       NUMBER OF PAGES   17          TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
TTTTTTTT                                     TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
```

```
PAUL RUN
--------------------

SYSTEM SPECIFICATIONS

FREE CORE =       +80
MEMORY SIZE =     +172
NO OF PROCESSORS =       +1
NO OF CHANNELS =       +3
I/O DELAY =  70.0
REALLOCATION DELAY =   1.0
DELAY FOR INSERTING ENTRY INTO READY QUEUE =   1.0
ROLL OUT DELAY = 600.0
ROLL IN DELAY = 600.0
RELOCATION DELAY =   5.0
DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE =   1.0
DELAY FOR SEARCHING FOR SPACE IN MEMORY =   2.0
INTERRUPT DELAY =   2.5
DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES =   2.0
HIGH LEVEL SCHEDULING DELAY =   1.0
LOW LEVEL SCHEDULING DELAY =   1.5
ACCOUNTING DELAY =   3.0
SIZE OF SEGMENTS =      +30
NO OF SEGMENTS =       +3
CHAPTER CHANGE DELAY =   1.0
FREE POOL OF MEMORY =       +5
MEMORY NEEDED FOR ESSENTIAL FUNCTION =       +3

WORKLOAD SPECIFICATIONS

SEED1 =      -11
SEED2 =       +7
NO OF JOBTYPES =       +8
% OF TYPE   1       +30
% OF TYPE       +2      +30
% OF TYPE       +3      +15
% OF TYPE       +4      +15
% OF TYPE       +5       +4
% OF TYPE       +6       +4
% OF TYPE       +7       +1
% OF TYPE       +8       +1
NO OF JOB SIZES =       +7
      +5% OF JOBS ARE       +6K
     +19% OF JOBS ARE      +10K
     +29% OF JOBS ARE      +14K
     +23% OF JOBS ARE      +18K
     +15% OF JOBS ARE      +22K
      +8% OF JOBS ARE      +26K
      +1% OF JOBS ARE      +30K
NO OF EVENTS =       +6

PROBS FOR JOBTYPE       +1
   +1000    +8000       +3   +80000    +7000       +1
PROBS FOR JOBTYPE       +2
   +1000    +4000       +3   +80000    +7000       +1
PROBS FOR JOBTYPE       +3
   +1000    +8000       +2   +80000    +7000       +1
PROBS FOR JOBTYPE       +4
   +1000    +4000       +2   +80000    +7000       +1
PROBS FOR JOBTYPE       +5
   +1000    +8000       +1   +80000    +7000       +1
PROBS FOR JOBTYPE       +6
   +1000    +4000       +1   +80000    +7000       +1
PROBS FOR JOBTYPE       +7
   +1000    +8000       +0   +80000    +7000       +1
```

```
PROBS FOR JOBTYPE       +3
    +1000    +4000      +0    +80000    +7000       +1
```

```
    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   50000.0 52.6      1.5        45.9      96      22      54

COREIMAGES TENTSTARTED JOBSINWELL
      +6       +4       +5

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  100000.0 52.5      1.7        45.8      96      22      54

COREIMAGES TENTSTARTED JOBSINWELL
      +6       +4       +5

JOB       +2TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  150000.0 52.3      1.7        46.0     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +4

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  200000.0 54.2      1.2        44.6     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +4

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  250000.0 55.2      1.0        43.8     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +4

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  300000.0 54.4      1.0        44.6     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +4

JOB       +5TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  350000.0 53.1      1.5        45.4     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +3

JOB       +3TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  400000.0 54.2      1.8        44.0     110      6      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7       +3       +2
```

```
      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 450000.0   53.8       1.5       44.7      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +2


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 500000.0   54.7       1.5       43.9      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +2


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 550000.0   53.5       1.4       45.1      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +4        +2


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 600000.0   53.8       1.7       44.5      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +4        +2

JOB       +9TERMED
JOB       +7TERMED

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 650809.0   51.8       1.8       46.4      106      10      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +1


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 700000.0   54.6       1.4       44.0      106      10      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +1


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 750000.0   54.9       1.5       43.6      106      10      56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +1

JOB       +12TERMED

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 800000.0   52.9       1.8       45.3      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +0


      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 850000.0   54.2       1.4       44.5      110      6       56

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +3        +0
```

```
JOB       +11TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  900000.0 55.8      0.9        43.2      110      10      52

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +2        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
  950000.0 56.3      0.8        42.9      110      10      52

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +2        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1000000.0 56.0      0.7        43.3      110      10      52

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +2        +0

JOB       +14TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1050000.0 56.1      1.0        42.9      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +1        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1100000.0 56.7      0.6        42.7      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +1        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1150000.0 55.1      1.1        43.8      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +2        +0

JOB       +1TERMED

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1200000.0 55.0      1.2        43.9      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +1        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1250000.0 55.8      1.2        43.0      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +1        +0

    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
 1300000.0 55.7      1.1        43.3      106      17      49

COREIMAGES TENTSTARTED JOBSINWELL
      +7        +2        +0
```

```
JOB       +15TERMED

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1350000.0 55.8      1.1        43.1      102      21      49

COREIMAGES TENTSTARTED JOBSINWELL
       +7        +1       +0

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1400000.0 56.8      0.6        42.6      102      21      49

COREIMAGES TENTSTARTED JOBSINWELL
       +7        +1       +0

JOB       +8TERMED

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1450000.0 55.1      1.2        43.7      88       32      47

COREIMAGES TENTSTARTED JOBSINWELL
       +6        +2       +0

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1500000.0 54.0      1.6        44.4      88       37      47

COREIMAGES TENTSTARTED JOBSINWELL
       +6        +2       +0

      TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1550000.0 53.8      1.6        44.7      88       37      47

COREIMAGES TENTSTARTED JOBSINWELL
       +6        +2       +0

JOB       +6TERMED
```

```
REALTIME IS -  1590088.000

TOTAL NO OF CHAPCH IS -  +394963
TOTAL NO OF CHAPTRANS IS -   +34670
TOTAL CHAPCH+TRANS PER SEC =   270.2
PERCENT REQ TRANS =  8.8
PROCESSOR USAGE
---------------
PRO   PROIDLETIME %TIME PRO IDLE WAIT IN JOB Q %TIME IN JOB Q    OSTIME     % TIME IN OS
 1    20468.000          1.3          0.000          0.0          867000.000     54.5
AVG % TIME PROS IDLE IS  1.3%
AVG % TIME PROS IN JOB Q IS  0.0%
CHANNEL USAGE
-------------
CHANNEL TIME CHANNEL IDLE % TIME CHANNEL IDLE
 1        71948.000          4.5%
 2        94977.000          6.0%
 3       137740.000          8.7%
AVG % TIME CHANNELS IDLE IS  6.4%
AVG FILESTORE I/O PER SEC =  18.2
MEMORY USAGE
------------
NO OF MEMORY RELOCATIONS=   0
AVG AMOUNT OF MEMORY UNUSED  14K
AVG % OF MEMORY UNUSED  8%
AVG AMOUNT OF MEM USED FOR CORE IMAGES 104K
AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS  52K
JOB DESCRIPTIONS
----------------
JOBNO JOBTYPE JOBTIME(MSECS) T/S'S    IOREQS    BLOCKTIME  SYSTEM TIME  TERMED
   1      3      47548.000     485      3791    762412.000    59489.000   T
   2      4       6473.000      70       264     66230.000     7548.000   T
   3      1      16926.000     175      1258    255753.000    21098.000   T
   4      1      64521.000     640      5018   1000521.000    80659.000   F
   5      1      12922.000     105      1071    200169.000    16158.000   T
   6      2      83351.000     841      3322    949140.000    98198.000   T
   7      3      21240.000     218      1708    345614.000    26573.000   T
   8      4      66587.000     712      2723    785570.000    78571.000   T
   9      1      11749.000     150       951    197470.000    14802.000   T
  10      1      46632.000     450      3825    750148.000    58669.000   F   -STILL BLOCKED
  11      1       8413.000     107       718    143255.000    10568.000   T
  12      2       7491.000      81       308     92392.000     8886.000   T
  13      2      41295.000     414      1654    460904.000    48448.000   F   -STILL BLOCKED
  14      2       7231.000      67       278     82837.000     8524.000   T
  15      4      14507.000     147       568    170524.000    17119.000   T
  16      6      20823.000     214       837    229419.000    24505.000   F
  17      4      15661.000     189       622    163880.000    18399.000   F
```

JOBNO	TIME JOB INITIATED	TIME JOB FINISHED	TIME IN SYSTEM
1	0.000	1192301.000	1192301.000
2	0.000	110966.000	110966.000
3	0.000	398063.000	398063.000
4	0.000	--	--
5	0.000	310993.000	310993.000
6	0.000	--	--
7	110972.500	643901.000	532928.500
8	110977.000	1420558.000	1309581.000
9	311002.500	613624.000	302621.500
10	398066.500	--	--
11	643909.500	866238.000	222328.500
12	643914.000	795488.000	151574.000
13	795897.500	--	--
14	866214.500	1010036.000	143821.500
15	1010045.500	1302175.000	292129.500
16	1192310.500	--	--
17	1302174.500	--	--

NUMBER OF PAGES 9

```
MOP                :SSP8086,MOPGEOCORE

#LISTING OF :SSP8086,MOPGEOCORE(1/8180) PRODUCED ON 21SEP78 AT 21.10.06

#G8.62S AT ASTON    IN ':SSP8086,MOPGEOCORE' ON 21SEP78 AT 21.10.29 USING  U14

DOCUMENT     :SSP8086,MOPGEOCORE(/8180)

STARTED :SSP8086,MOPGEOCORE,21SEP78 20.20.32 TYPE: BACK  10 : STREAM D : PRIORITY 1 : ENTERED WELL 21SEP78 15.34.50
20.20.32+ RJ MOPGEOCORE,UAALGOL68,PARAM(PROG SCOPGEO3,DATA GEODATA,STORE 20000),JD(JT3000,MZ40K)
**************************************************************************************
*                                   SOFTWARE CHANGES                                 *
*                                                                                    *
*        USERS ARE REMINDED THAT THE FOLLOWING SOFTWARE CHANGES TOOK                 *
*        PLACE ON MONDAY 4TH SEPTEMBER 1978 :-                                       *
*              1) UACLUSTANC WITHDRAWN                                               *
*              2) FSCE LIBRARY WITHDRAWN                                             *
*              3) EXPRESS SYSTEM WILL TIME LIMIT OF 20 SECONDS FOR                   *
*                 COMPILATION , CONSOLIDATION AND EXECUTION INTRODUCED               *
*              4) EXPRESS SYSTEM OUTPUT LIMIT OF 500 LINES WITH A DEFAULT            *
*                 OF 100 LINES INTRODUCED.                                           *
*                                                                                    *
*        FOR FURTHER INFORMATION SEE AUGUST 1978 NEWSLETTER (NO.18)                  *
*                                                                                    *
**************************************************************************************
20.20.32 JOB IS NOW FULLY STARTED
20.20.32+ UAALGOL68        PROG SCOPGEO3,DATA GEODATA,STORE 20000
20.20.32+        TA AB,CM,JL

   UAALGOL68: STATUS REPORT NUMBER 4      DATE 23 NOVEMBER 77

20.20.37 0.01 CORE GIVEN 2560
COMPILED BY THE ALGOL68-R COMPILING-SYSTEM   ON 21/09/78 AT 20/20/38        ASTON UNIVERSITY

20.20.39 0.02 CORE GIVEN 37888
    TIMESIM
    'BEGIN'
    'BOOL'CONTINUE:='TRUE',OK:='TRUE',GAPFOUND,OK1,BOOL,INTPUPTYPE,LASTJOBIN:='TRUE';
     [1:70 ]'REAL'JBSIZE,JBBLOCKED,JOBSTART,JOBFINISH,JBBLOCKSTART,SYSTIME;
    [1:100]'INT'IOQ;
    'REAL'RV,SAMPLEVAR:=50.0,PROIDLESUM:=0 0,CHANNELIDLESUM:=0.0,PROJQWAITSUM:=0.0,JBLOCKSUM:=0.0,
         SAMPLEPRINT:=50000.0,OSTOT,URATIME:=0.0,IDLETOT;
    'INT' SEED1:=11,SEED2:=7,IOQSIZE:=0,JQSIZE:=0,FREECORE,SAVE,COREIMAGE:=0,TOTAL,
         TENTSTART,TOTJOBSTART,NEWQUOTASIZ,JOBSINWELL:=5,CHAPCHNO:=0,
            CHAPTRANSNO:=0,RELOCNO:=0;
    'INT'JOBNEW ,JOBCOUNT:=0,X,NO,I,SIZ,REG,FIN,HLSVAR,
         JOBNO,INTRUPPTR1:=1,INTRUPPTR2:=1,JOBLOAD:=0,JOBSIZE,IOSUM:=0,
       FRAGLEVEL:=0,NEXTJOB:=0,UNUSECORE:=0,FRAGCOUNT:=0,PERCENT,AVGUNUSED,
       CHA,NXTTYPE:=0,SLICEPROB,IOPROB,TERMPROB,JOBTYPE,
       SYSCORE:=0, SYSCORECOUNT:=0,USERCORE:=0,USERCORECOUNT:=0,
        USERCOREAVG,SYSCOREAVG;
```

```
'REAL'REALTIME:=0.0,IOTIME:=0.0,JBLISTIME:=0.0,TEMP,CHAPCHDEL,QUOTA;
'STRING'TITLE;
'INT'JOBTYPNO:=20,JOBSIZNO:=8,EVENTNO:=8;
[1:JOBTYPNO]'INT'TYPPERCENT;
[1:JOBSIZNO]'INT'JOBSIZES,SIZPERCENT;
[1:JOBTYPNO,1:EVENTNO]'INT'EVENTPROBS;
'INT'ESSENTFUNCT,FREEPOOL,SEGNO,SEGSIZE;

'INT'MEMSIZE,N:=20,CHANO:=20;

'REAL'IODEL,REALLOCDEL,INSERTDEL,ROLLOUTDEL,
      RELOCDEL,CHANNELDEL,GAPCHECKDEL,ROLLINDEL,
      INTRUPDEL,SWAPQDEL,HLSDEL,LLSDEL,ACCOUNTDEL;
[0:70,1:4]'INT'JOB;
[1:N]'REAL'IDLETIME,PROJQWAIT,OSTIME,OSTIMESAMPLE,IDLESAMPLE;
[1:CHANO]'REAL'CHANNEL,CHANNELIDLE;
[1:70]'BOOL'NOTROLLEDOUT,WAITLOCKED,TERMINED;
[1:70]'INT'TSEXPD,IOREQ;
[1:N ]'INT'PR;
[1:N]'REAL'PRTIME;
[1:N]'BOOL'NOTENT,IO;
[1:N]'BOOL'NEWJBOOL;
'MODE''INTRUPENT'='STRUCT'('REAL'INTRUPTIME,'BOOL'INTRUPTYP,'BOOL'IO,'INT'JOBNO,
                           'REF''INTRUPENT'NXT);
'REF''INTRUPENT'INHEAD,IPTR1,IPTR2,ITAIL;
'REF''INTRUPENT'NIL='NIL';
'MODE''IOLISTENT'='STRUCT'('REAL'IOTIM,'BOOL'IO,'BOOL'INTRUPTYPE,
                           'INT'JOB,'REF''IOLISTENT'IONXT);
'REF''IOLISTENT'IOHEAD,IOTAIL,IOPTR1,IOPTR2;
'REF''IOLISTENT'NOTHING='NIL';
'MODE''JQENT'='STRUCT'('INT'PRO,'REAL'QTIME,'REF''JQENT'QNEXT);
'REF''JQENT'QHEAD,QPTR1,QPTR2;
'REF''JQENT'NILL='NIL';
'MODE''GAPENTRY'='STRUCT'('INT'SIZE,'INT'BEGIN,'INT'END,
                          'REF''GAPENTRY'PTR);
'REF''GAPENTRY'NULLUS='NIL';
'REF''GAPENTRY'GPTR1,GPTR2,GPTR3,GHEAD,GTAIL;
'MODE''JOBENTRY'='STRUCT'('INT'P,'REAL'TIME,'REF''JOBENTRY'NEXT);
'REF''JOBENTRY'NULL ='NIL';
'REF''JOBENTRY'PTR1,PTR2,HEAD,TAIL;


'C'GENERATES TWO DISTINCT SEQUENCES OF RANDOM NOS TO BE USED FOR GENERATING
    JOB CHARACTERISTICS AND SEQUENCES OF EVENTS RESPECTIVELY'C'

'PROC'RAND1='REAL';
    (NORMRAND:=SEED1;
     RV:=RANDOM;
     SEED1:=NORMRAND;RV);
'PROC'RAND2='REAL';
    (NORMRAND:=SEED2;
     RV:=RANDOM;
     SEED2:=NORMRAND;RV);


'C'CALCULATES TIME WHEN A CHANNEL WILL BE FREE TO CARRY OUT AN ISO OPERATION'C'

'PROC'CHANNELTIME='VOID';
   (IOTIME:=CHANNEL[1];CHA:=1;
    'FOR'I'TO'CHANO'DO'
      'IF'CHANNEL[I]<IOTIME
      'THEN'IOTIME:=CHANNEL[I];CHA:=I
```

```
        'FI');
        
    'C'PLACES THE EVENT OF AN INTERRUPT ON A LIST 'C'

    'PROC'INTRUPLIST=('REAL'TIME,'INT'JOBNO,'BOOL'BOOL,'BOOL'B)'VOID';
        ('IF'IHEAD'IS'NIL
        'THEN'IHEAD:=ITAIL:='INTRUPENT':=(TIME,BOOL,B,JOBNO,'NIL')
        'ELSE'CONTINUE:='TRUE';IPTR1:='NIL';IPTR2:=IHEAD;
              'WHILE'CONTINUE'AND'(NXT'OF'IPTR2'ISNT'NIL)'DO'
                 'IF'TIME>=INTRUPTIME'OF'IPTR2
                 'THEN'IPTR1:=IPTR2;IPTR2:=NXT'OF'IPTR2
                 'ELSE'IPTR1'IS'NIL
                 'THEN'IHEAD:='INTRUPENT':=(TIME,BOOL,B,JOBNO,IPTR2);
                       CONTINUE:='FALSE'
                 'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                                    (TIME,BOOL,B,JOBNO,IPTR2);
                       CONTINUE:='FALSE'
                 'FI';
              'IF'NXT'OF'IPTR2'IS'NIL
              'THEN''IF'INTRUPTIME'OF'IPTR2<=TIME
                    'THEN'ITAIL:=NXT'OF'ITAIL:='INTRUPENT':=
                          (TIME,BOOL,B,JOBNO,'NIL')
                   'ELSE'IPTR1'IS'NIL
                   'THEN'IHEAD:='INTRUPENT':=
                          (TIME,BOOL,B,JOBNO,IPTR2)
                   'ELSE'NXT'OF'IPTR1:='INTRUPENT':=
                          (TIME,BOOL,B,JOBNO,IPTR2)
                   'FI'
              'FI'
        'FI');

    'C'PLACES AN ISO REQUEST ONTO A LIST WHICH IS SERVED BY THE CHANNELS'C'

    'PROC'IOLIST=('INT'I)'VOID';
       ('IF'IOHEAD'IS'NOTHING
         'THEN'IOHEAD:=IOTAIL:='IOLISTENT':=
              (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
        'ELSE'CONTINUE:='TRUE';IOPTR1:='NIL';IOPTR2:=IOHEAD;
              'WHILE'CONTINUE'AND'(IONXT'OF'IOPTR2'ISNT'NOTHING)'DO'
               ('IF'PRTIME[I ]>=IOTIM'OF'IOPTR2
                'THEN'IOPTR1:=IOPTR2;IOPTR2:=IONXT'OF'IOPTR2
                'ELSE'IOPTR1'IS'NOTHING
                'THEN'IOHEAD:='IOLISTENT':=
                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                      CONTINUE:='FALSE'
                'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                      (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2);
                      CONTINUE:='FALSE'
                'FI');
                'IF'IONXT'OF'IOPTR2'IS'NOTHING
              'THEN''IF'IOTIM'OF'IOPTR2<=PRTIME[I ]
                    'THEN'IOTAIL:=IONXT'OF'IOTAIL:='IOLISTENT':=
                          (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],'NIL')
                   'ELSE'IOPTR1'IS'NOTHING
                  'THEN'IOHEAD:='IOLISTENT':=
                          (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                   'ELSE'IONXT'OF'IOPTR1:='IOLISTENT':=
                          (PRTIME[I ],IO[I],INTRUPTYPE,PR[I ],IOPTR2)
                   'FI'
              'FI'
```

```
142       'FI');
143
144
145   'C'PLACES AN ENTRY FOR A JOB ONTO THE READY QUEUE,NOTING THE TIME WHEN
146       IT WILL BE READY TO RUN'C'
147
148   'PROC'INSERT=('INT'I)'VOID';
149      (PRTIME[I]'PLUS'INSERTDEL;SYSTIME[PR[I]]'PLUS'INSERTDEL;OSTIME[I]'PLUS'INSERTDEL;
150   'IF'HEAD'IS'NULL
151      'THEN'HEAD:=TAIL:='JOBENTRY':=(PR[I ],PRTIME[I ],'NIL')
152      'ELSE'CONTINUE:='TRUE';PTR1:='NIL';PTR2:=HEAD;
153            'WHILE' CONTINUE'AND'(NEXT'OF'PTR2'ISNT'NULL)'DO'
154              ('IF'PRTIME[I ]>=TIME'OF'PTR2
155               'THEN'PTR1:=PTR2;PTR2:=NEXT'OF'PTR2
156               'ELSE'PTR1'IS'NULL
157               'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
158                     CONTINUE:='FALSE'
159               'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
160                      CONTINUE:='FALSE'
161               'FI');
162               'IF'NEXT'OF'PTR2'IS'NULL
163               'THEN''IF'TIME'OF'PTR2<=PRTIME[I ]
164                     'THEN'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=
165                                (PR[I ],PRTIME[I ],'NIL')
166                     'ELSE'PTR1':IS'NULL
167                     'THEN'HEAD:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
168                     'ELSE'NEXT'OF'PTR1:='JOBENTRY':=(PR[I ],PRTIME[I ],PTR2)
169                     'FI'
170               'FI'
171      'FI');
172
173
174   'C'INITIALISES I/O OPERATION AND BLOCKS JOB'C'
175
176   'PROC'INOUT=('INT'I)'VOID';
177      (PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;JOBLOCKSTART[PR[I]]:=PRTIME[I];
178       SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
179
180   IO[I]:='TRUE';WAITBLOCKED[PR[I ]]:='TRUE';
181    INTRUPTYPE:='TRUE';
182     IOLIST(I));
183
184
185   'C'INITIATES ROLLING IN OF A JOB'C'
186
187   'PROC'ROLLIN=('INT'I)'VOID';
188         (JOBSTART[PR[I]]:=PRTIME[I];
189        PRTIME[I ]:= PRTIME[I ]+CHANNELDEL;
190        SYSTIME[PR[I]]'PLUS'CHANNELDEL;OSTIME[I]'PLUS'CHANNELDEL;
191        IO[I]:='FALSE';
192        INTRUPTYPE:='TRUE';
193      WAITBLOCKED[PR[I ]]:='TRUE';
194       IOLIST(I));
195
196
197   'C'ASSIGNS FIRST ENTRY ON READY QUEUE TO A PROCESSOR'C'
198
199   'PROC'REALLOC=('INT'I)'VOID';
200       ('IF'HEAD'IS'NULL
201         'THEN'
202              PR[I ]:=0
203       'ELSE'PTR2:=HEAD;
```

```
                'IF'TIME'OF'PTR2> PRTIME[I ]
        'THEN'
                    PR[I ]:=0
                'ELSF'NEXT'OF'PTR2'ISNT'NULL
               'THEN'HEAD:='NEXT'OF'PTR2;
                    PR[I ]:='P'OF'PTR2
                'ELSF'HEAD:='TAIL:='NIL';
                    PR[I ]:='P'OF'PTR2
                'FI'
        'FI';
      PRTIME[I ]:=PRTIME[I ]+REALLOCDEL;
          'IF'PR[I]>0'THEN'OSTIME[I]'PLUS'REALLOCDEL'FI';JBLISTIME:=PRTIME[I ]);


'C'GENERATES JOB CHARACTERISTICS'C'

'PROC'JOBSIZGEN='VOID';
     ('IF'LASTJOBIN'THEN' NO:='ENTIER'(RAND1*100);
     'IF'NO<='TYPPERCENT[1]'THEN'JOBTYPE:=1
     'ELSF'NO<='TYPPERCENT[2]'THEN'JOBTYPE:=2
     'ELSF'NO<='TYPPERCENT[3]'THEN'JOBTYPE:=3
     'ELSF'NO<='TYPPERCENT[4]'THEN'JOBTYPE:=4
     'ELSF'NO<='TYPPERCENT[5]'THEN'JOBTYPE:=5
     'ELSF'NO<='TYPPERCENT[6]'THEN'JOBTYPE:=6
     'ELSF'NO<='TYPPERCENT[7]'THEN'JOBTYPE:=7
    'ELSF'NO<='TYPPERCENT[8]'THEN'JOBTYPE:=8
    'ELSF'NO<='TYPPERCENT[9]'THEN'JOBTYPE:=9
    'ELSF'NO<='TYPPERCENT[10]'THEN'JOBTYPE:=10
    'ELSF'NO<='TYPPERCENT[11]'THEN'JOBTYPE:=11
    'ELSF'NO<='TYPPERCENT[12]'THEN'JOBTYPE:=12
    'ELSF'NO<='TYPPERCENT[13]'THEN'JOBTYPE:=13
    'ELSF'NO<='TYPPERCENT[14]'THEN'JOBTYPE:=14
    'ELSF'NO<='TYPPERCENT[15]'THEN'JOBTYPE:=15
    'ELSF'NO<='TYPPERCENT[16]'THEN'JOBTYPE:=16
    'ELSF'NO<='TYPPERCENT[17]'THEN'JOBTYPE:=17
    'ELSF'NO<='TYPPERCENT[18]'THEN'JOBTYPE:=18
    'ELSF'NO<='TYPPERCENT[19]'THEN'JOBTYPE:=19
     'ELSE'JOBTYPE:=20
     'FI';
      NO:='ENTIER'(RAND1*100);
           'IF'NO<='SIZPERCENT[1]'THEN'JOBSIZE:=JOBSIZES[1]
           'ELSF'NO<='SIZPERCENT[2]'THEN'JOBSIZE:=JOBSIZES[2]
           'ELSF'NO<='SIZPERCENT[3]'THEN'JOBSIZE:=JOBSIZES[3]
           'ELSF'NO<='SIZPERCENT[4]'THEN'JOBSIZE:=JOBSIZES[4]
           'ELSF'NO<='SIZPERCENT[5]'THEN'JOBSIZE:=JOBSIZES[5]
          'ELSF'NO<='SIZPERCENT[6]'THEN'JOBSIZE:=JOBSIZES[6]
          'ELSF'NO<='SIZPERCENT[7]'THEN'JOBSIZE:=JOBSIZES[7]

           'ELSE'JOBSIZE:=JOBSIZES[8]
           'FI'
    'ELSE'LASTJOBIN:='TRUE'
    'FI');


'C'DECIDES WHETHER OR NOT TO TENTATIVELY START A JOB'C'

'PROC'HLS=('INT'I)'VOID';
   (PRTIME[I]'PLUS'HLSDEL;OSTIME[I]'PLUS'HLSDEL;
    HLSVAR:=TENTSTART+40;
    'IF'HLSVAR<200'AND'JOBSINQELL>0
```

```
266         'THEN'TENTSTART'PLUS'1;JOBSINWELL'MINUS'1
267         'FI')!
268
269
270
271     'C'ORGANISES QUEUE OF PROCESSORS WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'
272
273     'PROC'JQLIST=('INT'I)'VOID':
274        ('IF'QHEAD'IS'NILL
275         'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],'NIL')
276         'ELSE'CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
277               'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
278               'DO'
279                ('IF'PRTIME[I ]>=QTIME'OF'QPTR2
280                 'THEN'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
281                 'ELSF'QPTR1'IS'NILL
282                 'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],QPTR2);CONTINUE:='FALSE'
283                 'ELSE'QNEXT'OF'QPTR1:='JQENT'I:=(I,PRTIME[I ],QPTR2)!
284                       CONTINUE:='FALSE'
285                 'FI')!
286                 'IF'QNEXT'OF'QPTR2'IS'NILL
287                 'THEN''IF'QTIME'OF'QPTR2<=PRTIME[I ]
288                       'THEN'QNEXT'OF'QPTR2:='JQENT'I:=(I,PRTIME[I ],'NIL')
289                       'ELSF'QPTR1'IS'NILL
290                       'THEN'QHEAD:='JQENT'I:=(I,PRTIME[I ],QPTR2)
291                       'ELSE'QNEXT'OF'QPTR1:='JQENT'I:=(I,PRTIME[I ],QPTR2)
292                       'FI'
293                 'FI'
294                 'FI';
295                  NOTENT[I]:='FALSE')!
296
297
298     'C'INITIALISES SIMULATION VARIABLES AND STATEFROM WHICH SIMULATION BEGINS'C'
299
300     'PROC'INIT='VOID':
301       (OK1:='TRUE';
302        'FOR'I'FROM'0'TO'70'DO'
303           'FOR'J'TO'4'DO'JOB[I,J]:=0)!
304
305     READ(TITLE)!PRINT((TITLE,NEWLINE,"--------------------",NEWLINE))!
306     PRINT((NEWLINE,"SYSTEM SPECIFICATIONS",NEWLINE,NEWLINE))!
307         READ(FREECORE)!PRINT(("FREE CORE =",FREECORE,NEWLINE))!
308         READ(MEMSIZE)!PRINT(("MEMORY SIZE =",MEMSIZE,NEWLINE))!
309         READ(N)!PRINT(("NO OF PROCESSORS =",N,NEWLINE))!
310         READ(CHANO)!PRINT(("NO OF CHANNELS =",CHANO,NEWLINE))!
311         READ(IODEL)!OUTF(STANDOUT,$"I|O DELAY ="<3.1>L$,(IODEL))!
312         READ(REALLOCDEL)!OUTF(STANDOUT,$"REALLOCATION DELAY ="<3.1>L$,(REALLOCDEL))!
313         READ(INSERTDEL)!OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO READY QUEUE ="<3.1>L$,(INSERTDEL))!
314         READ(ROLLOUTDEL)!OUTF(STANDOUT,$"ROLL OUT DELAY ="<3.1>L$,(ROLLOUTDEL))!
315         READ(ROLLINDEL)!OUTF(STANDOUT,$"ROLL IN DELAY ="<3.1>L$,(ROLLINDEL))!
316         READ(RELOCDEL)!OUTF(STANDOUT,$"RELOCATION DELAY ="<3.1>L$,(RELOCDEL))!
317         READ(CHANNELDEL)!OUTF(STANDOUT,$"DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE ="<3.1>L$,(CHANNELDEL))!
318         READ(GAPCHECKDEL)!OUTF(STANDOUT,$"DELAY FOR SEARCHING FOR SPACE IN MEMORY ="<3.1>L$,(GAPCHECKDEL))!
319         READ(INTRUPDEL)!OUTF(STANDOUT,$"INTERRUPT DELAY ="<3.1>L$,(INTRUPDEL))!
320         READ(SWAPQDEL)!OUTF(STANDOUT,$"DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES ="<3.1>L$,(SWAP
        'DEL'))!
321         READ(HLSDEL)!OUTF(STANDOUT,$"HIGH LEVEL SCHEDULING DELAY ="<3.1>L$,(HLSDEL))!
322         READ(LLSDEL)!OUTF(STANDOUT,$"LOW LEVEL SCHEDULING DELAY ="<3.1>L$,(LLSDEL))!
323         READ(ACCOUNTDEL)!OUTF(STANDOUT,$"ACCOUNTING DELAY ="<3.1>L$,(ACCOUNTDEL))!
324         READ(SEGSIZE)!PRINT(("SIZE OF SEGMENTS =",SEGSIZE,NEWLINE))!
325         READ(SEGNO)!PRINT(("NO OF SEGMENTS =",SEGNO,NEWLINE))!
326         READ(CHAPCHDEL)!OUTF(STANDOUT,$"CHAPTER CHANGE DELAY ="<3.1>L$,(CHAPCHDEL))!
```

```
      READ(FREEPOOL);PRINT(("FREE POOL OF MEMORY =",FREEPOOL,NEWLINE));
      READ(ESSENTFUNCT);PRINT(("MEMORY NEEDED FOR ESSENTIAL FUNCTION =",ESSENTFUNCT,NEWLINE));
PRINT((NEWLINE,"WORKLOAD SPECIFICATIONS",NEWLINE,NEWLINE));
      READ(SEED1);PRINT(("SEED1 =",SEED1,NEWLINE));
      READ(SEED2);PRINT(("SEED2 =",SEED2,NEWLINE));
      READ(JOBTYPNO);PRINT(("NO OF JOBTYPES =",JOBTYPNO,NEWLINE));
      READ(TYPPERCENT[1]);PRINT(("% OF TYPE   1 =",TYPPERCENT[1],NEWLINE));
      'FOR'I'FROM'2'TO'JOBTYPNO'DO'
        (READ(TYPPERCENT[I]));
PRINT(("% OF TYPE",I,TYPPERCENT[I],NEWLINE));
        TYPPERCENT[I]:=TYPPERCENT[I]+TYPPERCENT[I-1];
      READ(JOBSIZNO);PRINT(("NO OF JOB SIZES =",JOBSIZNO,NEWLINE));
      READ(SIZPERCENT[1]);
      READ(JOBSIZES[1]);PRINT((SIZPERCENT[1],"% OF JOBS ARE",JOBSIZES[1],"K",NEWLINE));
      'FOR'I'FROM'2'TO'JOBSIZNO'DO'
        (READ(SIZPERCENT[I]));
        READ(JOBSIZES[I]);
       PRINT((SIZPERCENT[I],"% OF JOBS ARE",JOBSIZES[I],"K",NEWLINE));
SIZPERCENT[I]:=SIZPERCENT[I]+SIZPERCENT[I-1];
      READ(EVENTNO);PRINT(("NO OF EVENTS =",EVENTNO,NEWLINE));
      'FOR'I'TO'JOBTYPNO'DO'
      (PRINT(NEWLINE);PRINT(("PROBS FOR JOBTYPE",I,NEWLINE));'FOR'J'TO'EVENTNO'DO'
        (READ(EVENTPROBS[I,J]);PRINT(EVENTPROBS[I,J])));
PRINT(NEWPAGE);
      TENTSTART:=4;
      'FOR'JOBNO'WHILE'OK1'AND'JOBLOAD#FREECORE'DO'
        (JOBSIZGEN;COREIMAGE'PLUS'1;
          'IF'FREECORE-JOBLOAD>=JOBSIZE
          'THEN'JOB[JOBNO,1]:=JOBSIZE;
                JOB[JOBNO,2]:=JOBLOAD+1;
                JOB[JOBNO,3]:=JOBSIZE+JOBLOAD;
                 JBBLOCKSTART[JOBNO]:=0.0;
                JOB[JOBNO,4]:=JOBTYPE;
                PRINT(NEWLINE);
                JOBNEW:=JOBNO+1;
                JOBLOAD'PLUS'JOBSIZE
           'ELSE'COREIMAGE'MINUS'1;
           FRAGLEVEL:=FREECORE-JOBLOAD;
                JOBNEW:=JOBNO;
                FRAGCOUNT'PLUS'1;UNUSECORE:=FRAGLEVEL;
               GHEAD:=GTAIL:='GAPENTRY':=
               (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL');
               OK1:='FALSE';LASTJOBIN:='FALSE'
          'FI';
          'IF'JOBLOAD=FREECORE'THEN'GHEAD:=NULLUS;
                                   FRAGCOUNT'PLUS'1;
                                   UNUSECORE:=FRAGLEVEL
          'FI');
      'FOR'I'TO'CHANO'DO'(CHANNEL[I]:=0.0;CHANNELIDLE[I]:=0.0);
    'FOR'I'TO' 70'DO'(JBSIZE[I]:=0.0;JBBLOCKED[I]:=0.0;TSEXPO[I]:=0;IOREQ[I]:=0;
                      SYSTIME[I]:=0.0;JOBSTART[I]:=0.0;JOBFINISH[I]:=0.0;TERMINED[I]:='FALSE');

      'FOR'I'TO'N'DO'(PRII ]:=I;PRTIME[I ]:=0.0;IDLETIME[I]:=0.0 ;OSTIME[I]:=0.0;
                      NEWJBOOL[I]:='FALSE';OSTIMESAMPLE[I]:=0.0;IDLESAMPLE[I]:=0.0;NOTENT[I]:='TRUE';
                      IO[I]:='FALSE';PROJOWAIT[I]:=0.0);
       'FOR'I'TO'70'DO'(NOTPOLLEDOUT[I]:='FALSE';WAITBLOCKED[I]:='FALSE');
      IHEAD:=ITAIL:='NIL';
      QHEAD:='NIL';
      'FOR'I'TO'('IF'N<JOBNEW-1'THEN'N'ELSE'JOBNEW-1'FI')'DO'PR[I]:=I;
      'IF'N>=JOBNEW-1
      'THEN''FOR'I'FROM'JOBNEW'TO'N'DO'PR[I]:=0;
            HEAD:=TAIL:='NIL'
```

```
        'ELSE'HEAD:=TAIL:='JOBENTRY':=(N+1,0,'NIL');
              'FOR'J'FROM'N+2'TO'JOBNEW-1'DO'TAIL:=NEXT'OF'TAIL:='JOBENTRY':=(J,0,'NIL')
        'FI');


'C'KEEPS A CHECK OF ALL GAPS IN CORE'C'

'PROC'GAPLIST=('INT'X,Y,Z)'VOID':
       (SIZ:=X;BEG:=Y;FIN:=Z;
     'IF'GHEAD'IS'NULLUS
    'THEN'GHEAD:=GTAIL:='GAPENTRY':=
            (SIZ,BEG,FIN,'NIL')
    'ELSE'GPTR3:=GHEAD;CONTINUE:='TRUE';OK1:='FALSE';
LOOPEND:'WHILE'CONTINUE'DO'
            ('IF'BEGIN'OF'GPTR3=FIN+1
             'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
                   BEGIN'OF'GPTR3:=BEG;
                   SIZ:=SIZE'OF'GPTR3;
                   BEG:=BEGIN'OF'GPTR3;
                   FIN:=END'OF'GPTR3;
                  'IF'OK1
                   'THEN'GPTR2:=GHEAD;GPTR1:='NIL';
                         'WHILE''IF'BEG=BEGIN'OF'GPTR2'AND'END'OF'GPTR2#FIN
                                'THEN''IF'GPTR1'IS'NULLUS
                                       'THEN'GHEAD:=PTR'OF'GPTR2
                                       'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                       'FI'
                                'FI';
                                PTR'OF'GPTR2'ISNT'NULLUS
                               'DO'(GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
                   'FI';
                   OK1:='TRUE'
             'ELSF'END'OF'GPTR3=BEG-1
             'THEN'SIZE'OF'GPTR3'PLUS'SIZ;
                   END'OF'GPTR3:=FIN;
                   SIZ:=SIZE'OF'GPTR3;
                   BEG:=BEGIN'OF'GPTR3;
                   FIN:=END'OF'GPTR3;
                   'IF'OK1
                         'THEN'GPTR2:=GHEAD;GPTR1:='NIL';
                               'WHILE''IF'FIN=END'OF'GPTR2'AND'BEGIN'OF'GPTR2#BEG
                                      'THEN''IF'GPTR1'IS'NULLUS
                                             'THEN'GHEAD:=PTR'OF'GPTR2
                                             'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
                                             'FI'
                                      'FI'
                                      PTR'OF'GPTR2'ISNT'NULLUS
                               'DO'(GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2)
                         'FI';
                   OK1:='TRUE'
             'FI';
             'IF'PTR'OF'GPTR3'IS'NULLUS
             'THEN'CONTINUE:='FALSE';
                   'GOTO'LOOPEND
             'FI';
             GPTR3:=PTR'OF'GPTR3);
          'IF'OK1='FALSE'
          'THEN'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
                  (SIZ,BEG,FIN,'NIL')
          'FI'
     'FI' );
```

```
451
452   'C'RELOCATES JOBS IN CORE'C'
453
454
455   'PROC'RELOC=('INT'I)'VOID';
456      (CONTINUE:='FALSE';PTR1:=HEAD;JOBLOAD:=0;
457       'IF'PTR1'ISNT'NULL'THEN'
458       'WHILE'NEXT'OF'PTR1'ISNT'NULL'DO'
459         (JOBNO:=P'OF'PTR1;
460          JOB[JOBNO,2]:=JOBLOAD+1;
461          JOBLOAD'PLUS'JOB[JOBNO,1];
462          JOB[JOBNO,3]:=JOBLOAD;
463          PTR1:=NEXT'OF'PTR1);
464       'IF'NEXT'OF'PTR1'IS'NULL
465       'THEN'JOBNO:=P'OF'PTR1;
466             JOB[JOBNO,2]:=JOBLOAD+1;
467             JOBLOAD'PLUS'JOB[JOBNO,1];
468             JOB[JOBNO,3]:=JOBLOAD
469       'FI'
470       'FI';
471       'FOR'I'TO'N-1'DO'
472         ('IF'PR[I ]#0'AND''NOT'NEWJBOOL[I]
473          'THEN'JOBNO:=PR[I ];
474                JOB[JOBNO,2]:=JOBLOAD+1;
475                 JOBLOAD'PLUS'JOB[JOBNO,1];
476                JOB[JOBNO,3]:=JOBLOAD
477          'FI');
478       'FOR'JOBNO'TO'70'DO'
479           'IF'NOTROLLEDOUT[JOBNO]'OR'WAITBLOCKED[JOBNO]
480           'THEN'JOB[JOBNO,2]:=JOBLOAD+1;
481                 JOBLOAD'PLUS'JOB[JOBNO,1];
482                 JOB[JOBNO,3]:=JOBLOAD
483           'FI';
484        'FOR'CHA'TO'CHANO'DO'
485           CHANNEL[CHA]'PLUS'RELOCDEL;
486        'WHILE'INTRUPTIME'UF'IPTR2'PLUS'RELOCDEL;
487               NXT'OF'IPTR2'ISNT'NIL'DO'
488                 IPTR2:=NXT'OF'IPTR2;
489       PRTIME[I ]'PLUS'RELOCDEL;STIME[I]'PLUS'RELOCDEL;
490       REALTIME'PLUS'RELOCDEL;
491       'IF'JBLISTIME<REALTIME
492       'THEN'JBLISTIME:=REALTIME
493       'FI';
494            'FOR'I'TO'N'DO'
495         ('IF'PRTIME[I ]<REALTIME
496          'THEN'PRTIME[I ]:=REALTIME
497          'FI'));
498
499
500   'C'CHECKS TO SEE IF ENOUGH SPACE IN CORE FOR A JOB TO BE ROLLED IN'C'
501
502    'PROC'SPACE'INCORE=('INT'I)'BOOL';
503           ('IF'GHEAD'ISNT'NULLUS
504            'THEN'      GPTR1:='NIL';GPTR2:=GHEAD;
505             CONTINUE:='TRUE';GAPFOUND:='FALSE';
506             'WHILE'CONTINUE'AND'(PTR'OF'GPTR2'ISNT'NULLUS)'DO'
507               'IF'SIZE'OF'GPTR2>=JOBSIZE
508               'THEN'CONTINUE:='FALSE';GAPFOUND:='TRUE';
509                     'IF'GPTR1'IS'NULLUS
510                     'THEN'GHEAD:=PTR'OF'GPTR2
511                     'ELSE'PTR'OF'GPTR1:=PTR'OF'GPTR2
512                     'FI'
```

```
513             'ELSE'GPTR1:=GPTR2;GPTR2:=PTR'OF'GPTR2
514             'FI';
515             'IF'PTR'OF'GPTR2'IS'NULLUS
516             'THEN''IF'SIZE'OF'GPTR2>=JOBSIZE
517                     'THEN'GAPFOUND:='TRUE';
518                           'IF'GPTR1'IS'NULLUS
519                           'THEN'GHEAD:=GTAIL:='NIL'
520                           'ELSE'GTAIL:=GPTR1;PTR'OF'GTAIL:='NIL'
521                           'FI'
522                     'FI'
523             'FI'
524            'ELSE'GAPFOUND:='FALSE'
525           'FI';
526            PRTIME[I]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
527            GAPFOUND);
528
529   'C'INITIATES SELECTION OF AND ROLLING IN OF NEW JOBS'C'
530
531
532   'PROC'NEWJOB=('INT'I)'VOID':
533        (          OK:='TRUE';
534            'WHILE'TENTSTART#0'AND'OK'DO'
535              (PR[I]:=JOBNEW;JOBSIZGEN;
536            'IF'SPACEINCORE(I)'AND'TENTSTART>0
537              'THEN'FRAGLEVEL'MINUS'JOBSIZE;
538               JOBLOAD'PLUS'JOBSIZE;
539               JOB[JOBNEW,1]:=JOBSIZE;
540               JOB[JOBNEW,2]:=BEGIN'OF'GPTR2  ;
541               JOB[JOBNEW,3]:=JOBSIZE+BEGIN'OF'GPTR2-1;
542               JOB[JOBNEW,4]:=JOBTYPE;
543               'IF'JOB[JOBNEW,3]<END'OF'GPTR2
544               'THEN''IF'GHEAD'IS'NULLUS
545                     'THEN'GHEAD:=GTAIL:='GAPENTRY':=
546                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1,
547                            END'OF'GPTR2,'NIL')
548                     'ELSE'GTAIL:=PTR'OF'GTAIL:='GAPENTRY':=
549                           (SIZE'OF'GPTR2-JOBSIZE,JOB[JOBNEW,3]+1 ,
550                            END'OF'GPTR2,'NIL')
551                     'FI'
552               'FI';
553                ROLLIN(I);TENTSTART'MINUS'1;COREIMAGE'PLUS'1;PRTIME[I]'PLUS'LLSDEL;
554                      OSTIME[I]'PLUS'LLSDEL;
555               JOBNEW'PLUS'1
556             'ELSE'FRAGLEVEL>=JOBLOAD'AND'TENTSTART>0
557            'THEN'RELOC(I);PRINT(("RELOCATION NECESSARY",NEWLINE));
558               RELOCNO'PLUS'1;
559               FRAGLEVEL'MINUS'JOBSIZE;
560               JOBLOAD'PLUS'JOBSIZE;
561               JOB[JOBNEW,1]:=JOBSIZE;
562               JOB[JOBNEW,2]:=JOBLOAD+1-JOBSIZE;
563               JOB[JOBNEW,3]:=JOBLOAD;
564               JOB[JOBNEW,4]:=JOBTYPE;
565               'IF'FRAGLEVEL#FREECORE-JOBLOAD
566               'THEN'FRAGLEVEL:=FREECORE-JOBLOAD;
567                     PRINT("MISTAKE IN GAPTABLE")
568               'FI';
569                'IF'FRAGLEVEL>0
570                'THEN'              GHEAD:=GTAIL:='GAPENTRY':=
571               (FRAGLEVEL,JOBLOAD+1,FREECORE,'NIL')
572               'ELSE'GHEAD:=GTAIL:='NIL'
573                'FI';
574               ROLLIN(I);COREIMAGE'PLUS'1;TENTSTART'MINUS'1;PRTIME[I]'PLUS'LLSDEL;
```

```
575                         OSTIME[I]'PLUS'LLSDEL;
576                   JOBNEW'PLUS'1
577               'ELSE'LASTJOBIN:='FALSE';OK:='FALSE'
578             'FI'))};
579
580
581   'C'CHECKS IF THERE ARE ANY INTERRUPTS THAT HAVE YET TO BE DEALT WITH
582        AFTER PROCESSOR HAS JUST DEALT WITH ONE'C'
583
584   'PROC'INTRUPCHECK=('INT'I)'VOID':
585      ('IF'IHEAD'ISNT'NIL
586       'THEN''IF'INTRUPTIME'OF'IHEAD<=PRTIME[I ]
587             'THEN'PR[I]:=-1
588             'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
589             'FI'
590       'ELSE'NEWJBOOL[I]:='TRUE';JQLIST(I)
591       'FI');
592   'PROC'INTERUPT=('INT'I)'VOID':
593        ('IF'NOTENT[I]'THEN'PRTIME[I]:=REALTIME
594      'ELSE''IF'PRTIME[I ]>JBLISTIME
595                          'THEN'JBLISTIME:=PRTIME[I ]
596            'ELSE''IF'PR[I]#0'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I] 'FI';
597                          PRTIME[I ]:=JBLISTIME
598              'FI';
599             NOTENT[I]:='TRUE';
600             CONTINUE:='TRUE';QPTR1:='NIL';QPTR2:=QHEAD;
601             'WHILE'CONTINUE'AND'(QNEXT'OF'QPTR2'ISNT'NILL)
602             'DO'('IF'PRO'OF'QPTR2=I
603                  'THEN''IF'QPTR1'IS'NILL
604                        'THEN'QHEAD:=QNEXT'OF'QHEAD
605                        'ELSE'QNEXT'OF'QPTR1:=QNEXT'OF'QPTR2
606                        'FI';
607                        CONTINUE:='FALSE'
608                  'ELSE'QPTR1:=QPTR2;QPTR2:=QNEXT'OF'QPTR2
609                  'FI');
610                  'IF'QNEXT'OF'QPTR2'IS'NILL
611                  'THEN''IF'PRO'OF'QPTR2=I
612                        'THEN''IF'QPTR1'IS'NILL
613                              'THEN'QHEAD:='NIL'
614                              'ELSE'QNEXT'OF'QPTR1:='NIL'
615                              'FI'
616                        'FI'
617                  'FI'
618       'FI';
619       'IF'PR[I ]>0
620       'THEN''IF'NEWJBOOL[I]
621             'THEN'NEWJBOOL[I]:='FALSE'
622             'ELSE'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
623                         OSTIME[I]'PLUS'INSERTDEL
624             'FI'
625       'FI';
626   'IF'INTRUPTYP'OF'IHEAD
627          'THEN''IF'IO'OF'IHEAD
628           'THEN'PRTIME[I]'PLUS'SWAPDEL;OSTIME[I]'PLUS'SWAPDEL;
629                 JBLISTIME'PLUS'SWAPDEL;
630                 PR[I]:=JOBNO'OF'IHEAD;
631                  JOBLOCKED[PR[I]]'PLUS'PRTIME[I]-JOBLOCKSTART[PR[I]];
632                 WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
633           'ELSE'PR[I]:=JOBNO'OF'IHEAD;
634                  WAITBLOCKED[PR[I]]:='FALSE';INSERT(I)
635           'FI';
636        IHEAD:=NXT'OF'IHEAD
```

```
637    'ELSE'JOBNO:=JOBNO'OF'IHEAD;
638        NOTROLLEDOUT[JOBNO]:='FALSE';
639         FRAGLEVEL'PLUS'JOB[JOBNO,1];
640         JOBLOAD'MINUS'JOB[JOBNO,1];
641             COREIMAGE'MINUS'1;
642         GAPLIST(JOB[JOBNO,1],JOB[JOBNO,2],JOB[JOBNO,3]);
643        PRTIME[I ]'PLUS'GAPCHECKDEL;OSTIME[I]'PLUS'GAPCHECKDEL;
644        JOB[JOBNO,2]:=JOB[JOBNO,3];=0;OSTIME[I]'PLUS'INTRUPDEL;
645             PRTIME[I ]'PLUS'INTRUPDEL;
646            NEWJOB(I);IHEAD:=NXT'OF'IHEAD
647      'FI');
648
649
650    'C'WHEN A JOB TERMINATES THIS ROUTINE SETS UP THE ROLING OUT OF THAT JOB'C'
651
652    'PROC'ENDJOB=('INT'I)'VOID';
653       (NOTROLLEDOUT[PR[I ]]:='TRUE';
654        PRTIME[I ]'PLUS'CHANNELDEL;SYSTIME[PR[I]]'PLUS'CHANNELDEL;
655          OSTIME[I]'PLUS'CHANNELDEL;
656        IO[I]:='FALSE';
657        INTRUPTYPE:='FALSE';    IOLIST(I));
658
659    'C'NOTES SIZE OF QUEUE WAITING TO ENTER CRITICAL REGION AROUND READY QUEUE'C'
660
661
662     'PROC'JQSIZEPROBE='VOID';
663       ('IF'QHEAD'IS'NILL
664        'THEN'PRINT((NEWLINE,"JQSIZE IS=",JQSIZE,NEWLINE))
665        'ELSE'    QPTR2:=QHEAD;
666        'WHILE''IF'QTIME'OF'QPTR2<='REALTIME'THEN'JQSIZE'PLUS'1'FI';QNEXT'OF'QPTR2'ISNT'NILL'DO'
667           QPTR2:=QNEXT'OF'QPTR2 ;
668        PRINT((NEWLINE,"JQSIZE IS",JQSIZE ,NEWLINE));
669        PRINT(NEWLINE);
670        JQSIZE:=0
671        'FI');
672
673
674
675    'C'
676       NOTES SIZE OF ISO REQUEST QUEUE'C'
677
678
679    'PROC'IOQPROBE='VOID';
680          ('IF'IOHEAD'IS'NOTHING
681         'THEN'PRINT((NEWLINE,"IQSIZE IS",IOQSIZE,NEWLINE))
682         'ELSE'IOPTR2:=IOHEAD;
683        'WHILE''IF'IOTIM'OF'IOPTR2<='REALTIME'THEN'IOQSIZE'PLUS'1'FI';IONXT'OF'IOPTR2'ISNT'NOTHING'DO'
684          IOPTR2:=IONXT'OF'IOPTR2 ;
685         PRINT((NEWLINE,"IQSIZE IS",IOQSIZE ,NEWLINE));
686         PRINT(NEWLINE);
687         IOQSIZE:=0
688       'FI');
689
690
691    'C'GENERATES THE NEXT EVENT THAT IS TO OCCUR WITHIN EACH PARTICULARJOBTHAT
692          IS RUNNING'C'
693
694
695    'PROC'INSTRGEN=('INT'I,JOBTYPE)'VOID';
696    (PPTIME[I]:=REALTIME;
697       'IF'PR[I ]>0
698       'THEN'NO:='ENTIER'(RAND2*100000);
```

```
                    JBSIZE[PR[I] ]'PLUS'1.0;PRTIME[I]'PLUS'1.0;
          'IF'TOTAL>=EVENTPROBS[JOBTYPE,1];NO<TOTAL
          'THEN'TSEXPD[PR[I]]'PLUS'1;
                JQLIST(I)
          'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,2];NO<TOTAL
          'THEN'IOREQ[PR[I]]'PLUS'1;
                INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
          'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,3];NO<TOTAL
          'THEN'JOBCOUNT'PLUS'1;TERMINED[PR[I]]:='TRUE';PRINT("JOB",PR[I],"TERMED",NEWLINE);
                NEWJBOOL[I]:='TRUE';PRTIME[I]'PLUS'ACCOUNTDEL;HLS(I);ENDJOB(I);JQLIST(I)
          'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,4];NO<TOTAL
           'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
                OSTIME[I]'PLUS'CHAPCHDEL;
              CHAPCHNO'PLUS'1

           'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,5];NO<TOTAL
           'THEN'PRTIME[I]'PLUS'CHAPCHDEL;SYSTIME[PR[I]]'PLUS'CHAPCHDEL;
              CHAPTRANSNO'PLUS'1;OSTIME[I]'PLUS'CHAPCHDEL;
                INOUT(I);NEWJBOOL[I]:='TRUE';JQLIST(I)
          'ELSF'TOTAL'PLUS'EVENTPROBS[JOBTYPE,6];NO<TOTAL
           'THEN'INSERT(I);JBLISTIME'PLUS'INSERTDEL;PRTIME[I]'PLUS'INSERTDEL;
                 NEWJBOOL[I]:='TRUE';
                JOBSINWELL'PLUS'1;HLS(I);
                JQLIST(I)
          'FI'
     'ELSF'PR[I ]=0'THEN'NEWJBOOL[I]:='TRUE';JQLIST(I)
     'FI');


'C'
     DETERMINES TIME OF NEXT EVENT THROUGHOUT THE SYSTEM'C'

'PROC'NATEVENTIME='VOID';
   (TEMP:=SAMPLEVAR;
    'FOR'I'TO'N'DO'
       'IF'NOTENT[I]'AND'PR[I ]>0
       'THEN''IF'TEMP>PRTIME[I ]
             'THEN'TEMP:=PRTIME[I ]
             'FI'
       'FI';
    'IF'CHANNELTIME;TEMP>IOTIME
    'THEN'TEMP:=IOTIME
    'FI';

    'IF'TEMP>JBLISTIME
    'THEN''IF'HEAD'ISNT'NULL
          'THEN''IF'TIME'OF'HEAD<=JBLISTIME
                'THEN'TEMP:=JBLISTIME
                'FI'
          'FI';
         'FOR'I'TO'N'DO'
         'IF''NOT'NEWJBOOL[I]'AND''NOT'NOTENT[I]'THEN'TEMP:=JBLISTIME'FI'
    'FI';
    'IF'IHEAD'ISNT'NIL
    'THEN'
    'IF'TEMP>INTRUPTIME'OF'IHEAD
    'THEN'TEMP:=INTRUPTIME'OF'IHEAD
    'FI'
    'FI');
```

```
'C' **MAIN BODY**

     ORGANISES RUNNING OF JOBS,QUEUE AROUND CRITICAL REGION ,ISQ QUEUE,
     RESULTS TO BE PRINTED OUT DURING RUN,TIME-KEEPING,INTERRUPTS,RESULTS AT END OF RUN'C'


INIT:
'WHILE'JOBCOUNT# 12'DO'
  ('FOR'I'TO'N'DO'
   ('IF'PRTIME[I]<=REALTIME'AND'PR[I ]=0 'THEN'IDLETIME[I]'PLUS'1.0'FI'
    'IF'NOTENT[I]'AND'PR[I ]>=0'THEN'
     'IF'PRTIME[I ]<=  REALTIME'THEN'INSTRGEN(I,JOB[PR[I ],4])'FI'
     'FI');
   'IF'QHEAD'ISNT'NILL
   'THEN''IF'REALTIME>=QTIME'OF'QHEAD'AND'REALTIME>=JBLISTIME
        'THEN'I:=PRO'OF'QHEAD;
              'IF'PRTIME[I]>JBLISTIME
               'THEN'JBLISTIME:=PRTIME[I]
               'ELSE''IF'PR[I]#0
                     'THEN'PROJQWAIT[I]'PLUS'JBLISTIME-PRTIME[I]
                      'FI';
                     PRTIME[I]:=JBLISTIME
               'FI';
              NOTENT[I]:='TRUE';
              'IF'NEWJBOOL[I]
              'THEN'REALLOC(I)'NEWJBOOL[I]:='FALSE'
              'ELSE'INSERT(I)'REALLOC(I)
              'FI';
              'IF'QNEXT'OF'QHEAD'ISNT'NILL
              'THEN'QHEAD:=QNEXT'OF'QHEAD
              'ELSE'QHEAD:='NIL'
              'FI'
        'FI'
   'FI';

   'FOR'I'TO'CHANO'DO'
   ('IF'CHANNEL[I]<REALTIME
    'THEN'CHANNELIDLE[I]'PLUS'REALTIME-CHANNEL[I];
          CHANNEL[I]:=REALTIME
   'FI';
     'IF'IOHEAD'ISNT'NOTHING
     'THEN''IF'CHANNEL[I]<=REALTIME'AND'IOTIM'OF'IOHEAD<=REALTIME
          'THEN''IF'IO'OF'IOHEAD
                'THEN'IOTIM'OF'IOHEAD:=REALTIME+IODEL;
                      INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                 INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                     CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                'ELSE'IOTIM'OF'IOHEAD:=REALTIME+OLLINDEL;
                      INTRUPLIST(IOTIM'OF'IOHEAD,JOB'OF'IOHEAD,
                                 INTRUPTYPE'OF'IOHEAD,IO'OF'IOHEAD);
                     'IF''NOT'INTRUPTYPE'OF'IOHEAD
                      'THEN'JOBFINISH[JOB'OF'IOHEAD];=IOTIM'OF'IOHEAD
                      'FI';

                     CHANNEL[I]:=IOTIM'OF'IOHEAD;IOHEAD:=IONXT'OF'IOHEAD
                'FI'
          'FI'
   'FI');
   'IF'REALTIME>=SAMPLEVAR
   'THEN'SAMPLEVAR'PLUS'50.0 'UNUSECORE'PLUS'FRAGLEVEL'FRAGCOUNT'PLUS'1;
```

```
                USERCORE'PLUS'JOBLOAD;USERCORECOUNT'PLUS'1;
                SYSCORE'PLUS'MEMSIZE-FREECORE;SYSCORECOUNT'PLUS'1
        'FI';
        'IF'REALTIME>=SAMPLEPRINT
        'THEN'SAMPLEPRINT'PLUS'50000.0;
              OSTOT:=0.0;IDLETOT:=0.0;
              'FOR'I'TO'N'DO'
               COSTOT'PLUS'(OSTIME[I]-OSTIMESAMPLE[I])/50000*100;
              IDLETOT'PLUS'(IDLETIME[I]-IDLESAMPLE[I])/50000*100;
                  OSTIMESAMPLE[I]:=OSTIME[I];IDLESAMPLE[I]:=IDLETIME[I];
              PRINT(NEWLINE);
              PRINT("    TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE",
                NEWLINE);
                OUTF(STANDOUT,$<8.1>,<2.1>3X,<2.1>7X,<2.1>4X,<3>4X,<3>4X,<3>L$,
              (REALTIME,OSTOT/N,IDLETOT/N,100-(OSTOT/N+IDLETOT/N),JOBLOAD,
                    FRAGLEVEL,MEMSIZE-FREECORE));PRINT(NEWLINE);
               PRINT(("CUREIMAGES TENTSTARTED JOBSINWELL",NEWLINE));
                PRINT((CUREIMAGE,TENTSTART,JOBSINWELL,NEWLINE,NEWLINE))
          'FI';
       REALTIME'PLUS'1.0;
      NXTEVENTIME;
      'IF'REALTIME<TEMP
      'THEN''FOR'I'TO'N'DO'
             'IF'PR[I ]=0
             'THEN'IDLETIME[I]'PLUS'TEMP-REALTIME
              'FI';
            REALTIME:=TEMP
      'FI';
      'IF'IHEAD'ISNT'NIL
      'THEN''IF'REALTIME>='INTRUPTIME'OF'IHEAD'AND'PRTIME[N ]<=REALTIME
            'THEN'INTERUPT(N);INTRUPCHECK(N)
           'FI'
      'FI';
      PRINT(NEWPAGE);
      AVGUNUSED:=UNUSECORE'/'FRAGCOUNT;USERCOREAVG:=USERCORE'/'USERCORECOUNT;
      SYSCOREAVG:=SYSCORE'/'SYSCORECOUNT;PERCENT:=AVGUNUSED*100'/'MEMSIZE;
      PRINT(NEWLINE);
      OUTF(STANDOUT,$"REALTIME IS -"<8.3>2L$,(REALTIME));
     PRINT(("TOTAL NO OF CHAPCH IS -",CHAPCHNO,NEWLINE,
            "TOTAL NO OF CHAPTRANS IS -",CHAPTRANSNO,NEWLINE));
OUTF(STANDOUT,$"TOTAL CHAPCH+TRANS PER SEC ="<5.1>L$,
     ((CHAPCHNO+CHAPTRANSNO)/REALTIME*1000));
OUTF(STANDOUT,$"PERCENT REQ TRANS ="<2.1>L$,
     (CHAPTRANSNO/CHAPCHNO*100));
     PRINT(("PROCESSOR USAGE",NEWLINE,"---------------",NEWLINE));
     PRINT(("PRO   PROIDLETIME %TIME PRO IDLE WAIT IN JOB Q %TIME IN JOB Q    OSTIME     % TIME IN OS",NEWLINE));
     'FOR'I'TO'N'DO'
      (OUTF(STANDOUT,$<1>  ,<8.3>8X,<2.1>3X,<8.3>6X,<2.1>8X,<8.3>4X,<2.1>L$,
           (I,IDLETIME[I],IDLETIME[I] *100/ REALTIME ,PROJQWAIT[I],
                PROJQWAIT[I] *100/ REALTIME ,OSTIME[I],OSTIME[I]*100/REALTIME));
       PROIDLESUM'PLUS'IDLETIME[I];PROJQWAITSUM'PLUS'PROJQWAIT[I]);
     OUTF(STANDOUT,$"AVG % TIME PROS IDLE IS"<2.1>"%"L$,
         (PROIDLESUM *100/ N / REALTIME ));
     OUTF(STANDOUT,$"AVG % TIME PROS IN JOB Q IS"<2.1>"%"L$,
         (PROJQWAITSUM *100/ N / JOBLISTIME ));
     PRINT(("CHANNEL USAGE",NEWLINE,"-------------",NEWLINE));
     PRINT(("CHANNEL TIME CHANNEL IDLE % TIME CHANNEL IDLE",NEWLINE));
     'FOR'I'TO'CHANO'DO'
      (OUTF(STANDOUT,$<1>3X,<8.3>8X,<2.1>"%"L$,(I,CHANNELIDLE[I],
            CHANNELIDLE[I] *100/ TOTIME ));
       CHANNELIDLESUM'PLUS'CHANNELIDLE[I]);
     OUTF(STANDOUT,$"AVG % TIME CHANNELS IDLE IS"<2.1>"%"L$,
```

```
              (CHANNELIDLESUM *100/ IOTIME  / CHANO));
   'FOR'I'TO'JOBNEW-1'DO'IOSUM'PLUS'IOREQ[I];
      OUTF(STANDOUT,$"AVG FILESTORE I/O PER SEC ="<3.1>L$,
             (IOSUM/REALTIME*1000));
        PRINT(("MEMORY USAGE",NEWLINE,"------------",NEWLINE));
        OUTF(STANDOUT,$"NO OF MEMORY RELOCATIONS="<3>L,
                     "AVG AMOUNT OF MEMORY UNUSED"<3>"K"L,
                     "AVG % OF MEMORY UNUSED"<2>"%"L ,
                     "AVG AMOUNT OF MEM USED FOR CORE IMAGES"<3>"K"L,
                       "AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS"<3>"K"L$,
            (RELOCNO,AVGUNUSED,PERCENT,USERCOREAVG,SYSCOREAVG));
        PRINT(("JOB DESCRIPTIONS",NEWLINE,"----------------",NEWLINE));
        PRINT(("JOBNO JOBTYPE JOBTIME(MSECS) T/S'S    IOREQS    BLOCKTIME  SYSTEM TIME  TERMED",
   NEWLINE));
        'FOR'I'TO'JOBNEW-1'DO'
         (OUTF(STANDOUT, $<3>3X,<2>2X,<8.3>3X ,<4>3X ,<5>3X ,<8.3> ,<8.3>$;
              (I,JOB[I,4],JBSIZE[I],TSEXPD[I] ,IOREQ[I],JBBLOCKED[I],SYSTIME[I]));
        PRINT(("  ",TERMINED[I]));
          JBBLOCKSUM'PLUS'JBBLOCKED[I] ;
         'IF'WAITBLOCKED[I]'AND'IOREQ[I]>0
         'THEN'JBLOCKSUM'PLUS'REALTIME-JBBLOCKSTART[I];
               PRINT(("   -STILL BLOCKED",NEWLINE))
         'ELSE'PRINT(NEWLINE)
         'FI');
       PRINT(NEWPAGE);
        PRINT(("JOBNO TIME JOB INITIATED   TIME JOB FINISHED     TIME IN SYSTEM";
                NEWLINE));
        'FOR'I'TO'JOBNEW-1'DO'
         'IF''NOT'('NOT'(JOBFINISH[I]<0.0)'AND''NOT'(0.0<JOBFINISH[I]))
         'THEN'OUTF(STANDOUT,$<3>3X,<8.3>8X,<8.3>7X,<8.3>L$,
              (I,JOBSTART[I],JOBFINISH[I],JOBFINISH[I]-JOBSTART[I]))
          'ELSE'OUTF(STANDOUT,$<3>3X,<8.3>3X"         --               --"L$,
                    (I,JOBSTART[I]))
              'FI'
   'END'
   'FINISH'
0.24 ;HALTED :    COMPILED TIMESIM
20.21.19 FREE *DA11,156 TRANSFERS
20.21.20 FREE *LPO ,923 TRANSFERS
20.2?.21 0.25 CORE GIVEN 35840,CLOCKED 0.19
DISPLAY : LD
26.38 :DELETED : OK
21.10.02 FREE *FHO ,25 TRANSFERS
21.10.02 FREE *DA12,111 TRANSFERS
21.10.02 FREE *DA11,74 TRANSFERS
21.10.02 FREE *TRO ,98 TRANSFERS
21.10.03 FREE *LPO ,422 TRANSFERS
21.10.03 FREE *DA15,7 TRANSFERS
21.10.03 26.30 DELETED,CLOCKED 26.31
PARAMETER NUMBER 1,2 UNACCESSED
DISPLAY: UAALGOL68R: NORMAL EXIT
21.10.05 26.39 FINISHED : 2 LISTFILES
21.10.05 JOBTIME USED 1600 : MAXIMUM CORE USED 37888
21.10.05 JOB UNITS 3520
```

```
†YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY†
†YYYYYYY                                YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY†
†YYYYYYYY        NUMBER OF PAGES   16    YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY†
†YYYYYYYY                                YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY†
†YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY†
```

```
PAUL RUN
----------

SYSTEM SPECIFICATIONS

FREE CORE =       +80
MEMORY SIZE =     +172
NO OF PROCESSORS =        +1
NO OF CHANNELS =        +3
I|O DELAY =   70.0
REALLOCATION DELAY =    1.0
DELAY FOR INSERTING ENTRY INTO READY QUEUE =    1.0
ROLL OUT DELAY = 600.0
ROLL IN DELAY = 600.0
RELOCATION DELAY =    5.0
DELAY FOR INSERTING ENTRY INTO CHANNEL QUEUE =    1.0
DELAY FOR SEARCHING FOR SPACE IN MEMORY =    2.0
INTERRUPT DELAY =    2.5
DELAY FOR SWAPPING ENTRY BETWEEN READY AND BLOCKED QUEUES =    2.0
HIGH LEVEL SCHEDULING DELAY =    1.0
LOW LEVEL SCHEDULING DELAY =    1.5
ACCOUNTING DELAY =    3.0
SIZE OF SEGMENTS =       +30
NO OF SEGMENTS =        +3
CHAPTER CHANGE DELAY =    1.0
FREE POOL OF MEMORY =        +5
MEMORY NEEDED FOR ESSENTIAL FUNCTION =        +3

WORKLOAD SPECIFICATIONS

SEED1 =       +11
SEED2 =        +7
NO OF JOBTYPES =        +8
% OF TYPE   1        +30
% OF TYPE        +2       +30
% OF TYPE        +3       +15
% OF TYPE        +4       +15
% OF TYPE        +5        +4
% OF TYPE        +6        +4
% OF TYPE        +7        +1
% OF TYPE        +8        +1
NO OF JOB SIZES =        +7
      +5% OF JOBS ARE        +6K
     +19% OF JOBS ARE       +10K
     +29% OF JOBS ARE       +14K
     +23% OF JOBS ARE       +18K
     +15% OF JOBS ARE       +22K
      +8% OF JOBS ARE       +26K
      +1% OF JOBS ARE       +30K
NO OF EVENTS =        +6

PROBS FOR JOBTYPE        +1
   +1000     +8000        +3    +80000     +7000        +1
PROBS FOR JOBTYPE        +2
   +1000     +4000        +3    +80000     +7000        +1
PROBS FOR JOBTYPE        +3
   +1000     +8000        +2    +80000     +7000        +1
PROBS FOR JOBTYPE        +4
   +1000     +4000        +2    +80000     +7000        +1
PROBS FOR JOBTYPE        +5
   +1000     +8000        +1    +80000     +7000        +1
PROBS FOR JOBTYPE        +6
   +1000     +4000        +1    +80000     +7000        +1
PROBS FOR JOBTYPE        +7
```

```
PROBS FOR JOB TYPE      +8
   +1000    +4000       +0   +80000    +7000       +1
```

```
      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
     50000.0 48.0     2.7        49.3      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +5

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    100000.0 48.4     2.5        49.1      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +5

JOB       +2TERMED
JOB       +1TERMED

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    150000.0 46.8     3.0        50.2      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +3

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    200000.0 47.5     3.5        49.2      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +3

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    250000.0 48.4     3.0        48.7      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +3

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    300000.0 48.1     2.8        49.1      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +5       +3

JOB       +3TERMED

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    350000.0 47.5     2.9        49.8      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +3

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    400000.0 48.8     2.8        48.4      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5       +4       +3

JOB       +4TERMED
```

```
      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    450000.0  48.4       3.0       48.6       78       2       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    500000.0  48.8       2.9       48.3       78       2       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +5        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    550000.0  49.1       2.5       48.3       78       2       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +5        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    600000.0  48.0       2.8       49.2       78       2       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +5        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    650000.0  48.1       3.2       48.8       78       2       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +5        +2

JOB       +7TERMED

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    700000.0  48.1       3.0       48.9       74       6       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    750000.0  48.5       2.3       49.1       74       6       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    800000.0  47.8       2.6       49.5       74       6       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    850000.0  48.7       2.8       48.5       74       6       92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +2


      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
```

```
      900000.0 47.9      3.0        49.1       74        6       92

COREIMAGES TENTSTARTED JOBSINWELL
        +5        +4        +2

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
    950000.0 48.6      2.3        49.0       74        6       92

COREIMAGES TENTSTARTED JOBSINWELL
        +5        +4        +2

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1000000.0 49.3      2.3        48.4       74        6       92

COREIMAGES TENTSTARTED JOBSINWELL
        +5        +4        +2

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1050000.0 48.5      2.6        48.9       74        6       92

COREIMAGES TENTSTARTED JOBSINWELL
        +5        +4        +2

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1100000.0 48.5      2.7        49.0       74        6       92

COREIMAGES TENTSTARTED JOBSINWELL
        +5        +4        +2

JOB       +8TERMED

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1150000.0 45.4      4.9        49.7       60       20       92

COREIMAGES TENTSTARTED JOBSINWELL
        +4        +5        +1

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1200000.0 44.5      5.3        50.1       60       20       92

COREIMAGES TENTSTARTED JOBSINWELL
        +4        +5        +1

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1250000.0 44.4      5.1        50.5       60       20       92

COREIMAGES TENTSTARTED JOBSINWELL
        +4        +5        +1

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1300000.0 44.4      5.7        49.8       60       20       92

COREIMAGES TENTSTARTED JOBSINWELL
        +4        +5        +1

JOB       +10TERMED

      TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
   1350000.0 45.1      4.6        50.3       74        6       92
```

```
COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +1

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1400000.0 49.2     2.6        48.3      74       6      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +1

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1450000.0 48.3     2.4        49.3      74       6      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +1

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1500000.0 48.8     2.5        48.6      74       6      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +1

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1550000.0 48.9     2.2        48.8      74       6      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +1

JOB      +12TERMED

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1600000.0 49.4     2.1        48.5      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +0

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1650000.0 49.0     2.4        48.6      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +0

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1700000.0 48.6     2.5        48.9      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +3        +0

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1750000.0 49.2     2.4        48.4      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
      +5        +4        +0

JOB      +13TERMED
JOB       +6TERMED

   TIME  %OSTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1800000.0 46.8     3.8        49.4      78       2      92
```

```
COREIMAGES TENTSTARTED JOBSINWELL
     +5        +2         +0

   TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1850000.0 48.5      2.8        49.0      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
     +5        +2         +0

   TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1900000.0 49.0      2.6        48.4      78       2      92

COREIMAGES TENTSTARTED JOBSINWELL
     +5        +2         +0

JOB       +9TERMED

   TIME  %USTIME  %IDLETIME  %OBJTIME  OBJCORE  UNUSE  SYSTEMCORE
1950000.0 46.5      4.6        48.9      64       16     92

COREIMAGES TENTSTARTED JOBSINWELL
     +4        +2         +0

JOB       +5TERMED
```

REALTIME IS = 1986320.000

TOTAL NO OF CHAPCH IS = +440498
TOTAL NO OF CHAPTRANS IS = +38417
TOTAL CHAPCH+TRANS PER SEC = 241.1
PERCENT REQ TRANS = 8.7
PROCESSOR USAGE

PRO	PROIDLETIME	%TIME PRO IDLE	WAIT IN JOB Q	%TIME IN JOB Q	OSTIME	% TIME IN OS
1	61777.000	3.1	0.000	0.0	949968.500	47.8

AVG % TIME PROS IDLE IS 3.1%
AVG % TIME PROS IN JOB Q IS 0.0%
CHANNEL USAGE

CHANNEL	TIME CHANNEL IDLE	% TIME CHANNEL IDLE
1	195248.000	9.8%
2	257889.000	13.0%
3	370529.000	18.6%

AVG % TIME CHANNELS IDLE IS 13.8%
AVG FILESTORE I/O PER SEC = 17.5
MEMORY USAGE

NO OF MEMORY RELOCATIONS= 0
AVG AMOUNT OF MEMORY UNUSED 6K
AVG % OF MEMORY UNUSED 3%
AVG AMOUNT OF MEM USED FOR CORE IMAGES 73K
AVG AMOUNT OF MEM USED FOR SYSTEM FUNCTIONS 92K
JOB DESCRIPTIONS

JORNO	JOBTYPE	JOBTIME(MSECS)	T/S*S	IOREQS	BLOCKTIME	SYSTEM TIME	TERMED	
1	3	7516.000	72	598	91341.000	9301.000	T	
2	4	7037.000	64	264	62785.000	8174.000	T	
3	1	17527.000	163	1421	217307.000	21624.000	T	
4	1	22266.000	225	1732	274846.000	27523.000	T	
5	1	102540.000	1006	8127	1250935.000	126923.000	T	
6	2	113811.000	1188	4701	1005313.000	131917.000	T	
7	3	27995.000	305	2208	345504.000	34601.000	T	
8	4	50057.000	520	1982	445650.000	58135.000	T	
9	1	77619.000	801	6036	926859.000	93581.000	T	
10	1	33766.000	308	2742	404384.000	41557.000	T	
11	1	33515.000	360	2706	411443.000	41418.000	F	
12	2	17463.000	190	679	153325.000	20270.000	T	
13	2	12144.000	131	494	110913.000	14083.000	T	
14	2	13506.000	150	543	112150.000	15507.000	F	-STILL BLOCKED
15	4	13476.000	147	535	113169.500	15502.000	F	

JOBNO	TIME JOB INITIATED	TIME JOB FINISHED	TIME IN SYSTEM
1	0.000	146032.000	146032.000
2	0.000	106390.000	106390.000
3	0.000	344103.000	344103.000
4	0.000	437032.000	437032.000
5	0.000	--	--
6	106398.500	1797170.000	1690771.500
7	146041.500	697472.000	551430.500
8	344111.500	1105044.000	760932.500
9	437040.500	1913360.000	1476319.500
10	697478.500	1330883.000	633404.500
11	1330865.500	--	--
12	1330868.000	1593910.000	263042.000
13	1593919.500	1782278.000	188358.500
14	1797178.500	--	--
15	1797185.000	--	--

NUMBER OF PAGES 10

BIBLIOGRAPHY

(A1) Abate J, Dubner H and Weinberg "Queuing Analysis of IBM 2314 Disk Storage Facility" J.ACM Vol 15 No.4 1968

(A2) Abate J, Dubner H "Optimizing the Performance of a Drum-like Storage" IEEE Trans Computers C-18, 11, pp 992-997, 1969

(A3) Abernathy D.M, et al "Survey of Design Goals for Operating Systems" ACM SIGOPS Vol 7 Nos. 3, 4, Vol 8 No. 1, 1973-4

(A4) ACM/SIGME Symp. on Computer Performance Evaluation Feb.1973

(A5) ACM/SIGMETRICS Newsletter "Performance Evaluation Bibliography" Performance Evaluation Review 2, 2, pp 37-49, June 1973

(A6) Adams R.T., Caldwell W.H.G., Carlton D.E. "Evaluation of Time-sharing Systems - Benchmarks" Georgia Inst. of Tech. report GITIS-69-22, 1969, PB 189621

(A7) Agajanian A.H. "A Bibliography on System Performance Evaluation" Computer 1975 pp 63-74

(A8) Agrawala A.K., Larsen R.L. "Experience with the Control Server Model on a lightly Loaded System" 4th Proc. of Symp. on Simulation of Computer Systems, NBS, Boulder Colo., SIGSIM/ACM, 8285.167, Vol 7, No.4, July 1976 pp 103

(A9) Akkoyunlue E., Bernstein A., Schantz R. "Interprocess Communication Facilities for Network Operating Systems" Computer, June 1974

(A10) American National Standard "Vocabulary for Information Processing" X3, 12, 1970

(A11) Anderson H., Sargent R. "Modelling, Evaluation and Performance Measurements of Time-sharing Computer Systems" Computing Reviews 13, 12, pp 603-608, Dec.1972

(A12) Anderson J.P. et al "D825 - A Multiple Computer System for Command and Control" Proc. AFIPS FJCC pp 86-96, 1962

(A13) Anderson J.P. "Program Structures for Parallel Processing" CACM 8, 12, pp 786-788, Dec. 1965

(A14) Anderson J.W. "Primitive Process Level Modelling and Simulation of a Multiprocessor Computer System" Computer Science Dept. TR-32, Ph.D. Dissertation, Univ. of Texas al Austin, May 1972

(A15) Apple C.T. "The Program Monitor - a Device for Program Performance Measurement" Proc. ACM 20th Nat.Conf. pp 66-75, 1965

(A16) Arden B.W., Boettner "Measurement and Performance of a Multiprogramming System" 2nd ACM Symp. on Operating System Principles, Princeton Univ. pp 130-146 Oct.1969

(A17) Arndt F.R., Oliver G.M. "Hardware Monitoring of Real-time Computer System Performance" Computer, Vol. 5 No.4., July/Aug. 1972 pp 25-29

(A18) Arnold J.S. et al "Design of a Tightly Coupled Multiprocessing Multiprogramming System" IBM Sys. J. Vol. 13. No.1, 1974

(A19) Aschenbrenner R.A. et al "The Neurotron Monitor System" AFIPS Conf. Proc. Vol. 39 pp 31-37, 1972, Las Vegas, Nevada

(A20) Auerbach, "Auerbach Guide to International Computer Systems Architecture" Philadelphia P.A. 1976

(B1) Baer J.L. "A Survey of some Theoretical Aspects of Multiprocessing" Computing Surveys Vol. 5 No. 1, March 1973

(B2) Baer J.L., Bovet D.P. "Compilation of Arithmetic Expressions for Parallel Computations" Proc. I.F.I.P. Congress, Booklet B, 4-10, 1968

(B3) Bahr D. "Computer Performance: Principles and Techniques" European Conf. on Computer Performance Evaluation, Eurocamp, London, Sept.1976

(B4) Bairstow J.N. "A Review of System Evaluation Packages" Computer Decisions, p 20, June 1970

(B5) Baker F.T. "Chief Programmer Team" Datamation, Dec. 1973

(B6) Baker F.T. "Chief Programmer Team Management of Production Programming" IBM Sys.J. Vol. 11, No.1, 1972

(B7) Baldwin F.R. et al "A Multiprocessing Approach to a Large Computer System" IBM Sys. J. Vol.1, Sept. 1962 pp 64 - 76

(B8) Ball J.R. et al "On the Use of the Soloman Parallel Processing Computer" Proc. AFIPS FJCC pp 137 - 146, 1962, Spartan Books N.Y.

(B9) Balzar R.M. "An Overview of ISPL Computer System Design" CACM pp.117, 1973

(B10) Banks R. "Assessing Computer System Design from a Performance Viewpoint" - Computer Performance Methods of Assessment" B.C.S. Conf.Papers, Univ. of Surrey, Sept.1972

(B11) Bard Y. "Performance Criteria and Measurement for a Time-sharing System" IBM Sys. J. Vol. 10 No. 3 pp 193 - 216, 1971

(B12) Barnes G.R. et al "The Illiac IV Computer" IEEE Trans. Computers 17, pp 246 - 257, 1968

(B13) Baskin H.B. et al "A Modular Computer Sharing System" .. CACM, Oct.1969

(B14) Baskin H.B. et al "PRIME - A Modular Architecture for Terminal Oriented Systems" Proc. AFIPS SJCC Vol.40 pp 431 - 437, 1972

(B15) Bell T.E. "Computer Performance Variability" ACM/NBS Computer Performance Evaluation Workshop, San Diego p 109, 1973 (NBS Special Pub.406, NBS Washington D.C. 1975)

(B16) Bell T.E. "Computer Performance Analysis: Measurement Objectives and Tools" Santa Monica, Rand, Feb. 1971 (R-584-NASA/PR)

(B17) Berners-Lee C.M. "A Performance Methodology for System Planning", On line Internat. Conf. on Computer Performance Evaluation, Brunel Univ., 1973

(B18) Berners-Lee C.M. "Four Years Experience with Performance Methodology for System Planning" European Conf. on Computer Performance Evaluation, Eurocamp, London, Sept.1976

(B19) Berners-Lee C.M. "Three Analytic Models of Batch Processing Systems" B.C.S. Conf. on Computer Performance, Univ. of Surrey, Sept. 1972

(B20) Betourne C. et al "System Design and Implementation Using Parallel Processors" Proc. IFIP Congress pp 345 - 353, 1971

(B21) Bibliography on Simulation, Rep. No. 320-0923-0, IBM Corp. White Plains, New York, 1966

(B22) Bhandarkar D.P. "Analysis of Memory Interference in Multiprocessors" IEEE-TC Vol. C-24 No. 9 pp 897-908, Sept.1975

(B23) Blaavw G.A. "The Structure of the System/360 Part V - Multisystem Organisation" IBM Sys. J. 3, 1964 pp 181 - 195

(B24) Blakeney G.R., et al "Design Characteristics of the 9020 System" IBM System J, 6, 1967 pp 90-94

(B25) Boehm B.W. "Studies in Measuring, Evaluating and Simulating Computer Systems, " Santa Monica, Rand., Sept. 1970 (R - 520 - N.A.S.A.)

(B26) Boehm B.W., Bell T.E. "Workshop Summery" ACM/NBS Computer Performance, Evaluation Workshop San Diego 1973 P.1. (NBS Special Publication 406, NBS Washington D.C. 1975)

(B27) Boehm B.W., Bell T.E. "Issues in Computer Performance Evaluation: Some Concensus, Some Divergence" NBS/ACM Workshop Summery, Performance Evaluation Reviews ACM/SIGMETRICS

(B28) Bonner A.J. "Using System Monitor Output to Improve Performance" IBM System Journal, Vol 8 No.4 1969 pp 290 - 298

(B29) Bordsen D.T. "Univac 1108 Hardware Instrumentation System" Proc. ACM/SIQOPS Workshop on Systems Performance Evaluation, Harvard Univ. April 1971 pp 1 - 28

(B30) Borovits I., Ein-Dor P. "Cost/Utilization: A Measure of System Performance" CACM March 1977, Vol.20 No.3

(B31) Bredt T.H. "Analysis of Parallel Systems" IEEE Trans. Computers Vol. 1-20 No. 11 pp 1403 - 1407, Nov. 1971

(B32) Bredt T.H. et al "Analysis and Synthesis of Control Mechanisms for Parallel Processes". In "Parallel Processor Systems, Technologies and Applications" Hobbs L.C. et al Spartan Books N.Y. 1970 pp 287 - 296

(B33) Bredt T., Saxena A.R. "Hierarchial Design Methods for Operating Systems" Proc COMPCON 1974 Fall (Sept. 1974)

(B34) Brice R.S., Sherman S.W. "Empirical Comparison of Partitioned and Non-Partitioned Buffer Management In Virtual Memory Systems" European Conf. on Computer Performance Evaluation

(B35) Bright H.S."A Philco Multiprocessing System" AFIPS Fall Joint Conf. 1964, pp 97 - 141, Spartan Books

(B36) Brinch Hansen P. "The Nucleus of a Multiprogramming System" CACM 13, 4, 1970 pp 238 - 242

(B37) Brinch Hansen P. "Structured Multiprogramming" CACM 15, 7, 1972 pp 574 - 578

(B38) Brinch Hansen P. "A Programming Methodology for Operating Systems Design" Proc. IFIP Congress 1974

(B39) Brinch Hansen P. "A Comparison of two Synchronizing Concepts" Acta Informatica 1972 pp 190 - 199

(B40) Brinch Hansen P. "Determining a Computing Central Environment" CACM July 1965 p 463

(B41) Brinch Hansen P. "The Purpose of Concurrent Pascal" Proc.Internat. Conf. A Reliable Software (April 1975)

(B42) Brinch Hansen P. "Operating Systems Principles" Prentice Hall 1973

(B43) Brinch Hansen P. Deamy P. "A Structured Operating System" Calif. Instit. of Technology, Passadena, Calif. May 1974

(B44) Brown G.G. "Analysis of IBM 300's Under Operating System" BCS Conf. on Computer Performance Univ. of Surrey Sept. 1972

(B45) Brown P.J., Saunders R.C. "A Comparison Between Implementation of an Identical Programme on Several Different Computers" BCS Conf. on Computer Performance Univ. of Surrey

(B46) Brown J.C. "Performance Factors for University Computer Facilities" ACM/NBS Computer Performance Evaluation Workshop, San Diego 1973 p 39 WBS special publication 406. Washington D.C. 1975

(B47) Brown J.C. "An Overview of Performance Evaluation" Second International Conf. on Software Engineering Oct. 1976 ACM/IEEE/NBS San Diego, Calif.

(B48) Buchholz W. "A Synthetic Job For Measuring System Performance" IBM System J. 8, 4, 1969 pp 309-331

(B49) Buchholz W. (ED) "Planning A Computer System: Project Sketch" McGraw Hill 1962 p 136

(B50) Burris H.R. "A Simulation Method for Multi Level Data Security Analysis" 4th Proc. of Symp. A. Simulation of Computer Systems. NBS Boulder Colo Sigsim/ACM 8285, 167 Vol. 7 No. 4, p.53

(B51) Burroughs Corp. Description Bulletin "The B5000 Concept"

(B52) B8500 Simulator for B5500, Programmers Reference Manual, Burroughs Corp. 1966

(B53) Bussell B. Koster R.A. "Instrumenting Computer Systems and their Programs" Proc. AFIPS FJCC 1970 pp 525 - 534

(B54) Buzen J.P. "Queueing Network Models of Multiprogramming" Ph.D. Diss. Harvard Univ. Cambridge, Mass. August 1971

(C1) Calingeart P. "System Performance Evaluation: Survey and Appraisal" CACM 10 No. 1 Jan. 1967 pp 12-18

(C2) Campbell D.J., Heffner W.J., "Measurement and Analysis of Large Operating Systems During System Development" AFIPS FJCC proc. 1968 Vol. 30 part 1 pp 903-914

(C3) Canning R.E. "Equipment Selection" Data Processing Digest June 1966 pp 1-9

(C4) Cantrell H.N., Ellison A.L. "Multiprogramming System Performance, Measurement and Analysis" AFIPS SJCC proc 1968 Vol. 29 pp 213-221

(C5) Cerf V.G. "Measurement of Recursive Processes" Computer Science Dept. Techn. rep. No. UCLA - ENG - 70 - 43. Univ. of Calif., Los Angeles, May 1970

(C6) Cerf V.G. et al "Formal Properties of a Graph Model of Computation" Computer Science Dept. Tech. rep No. UCLA - ENG -7178, Dec. 1971

(C7) Cheatham T., et al "RADC Program Transferability Study" Report RADC - TR - 63 - 431 Nov. 1968 CFSTIAD 678 589

(C8) Chen. Y.E., Epley D.L. "Memory Requirements In a Multiprocessor Environment" J.ACM 19, 1, Jan.1972 pp 57-61

(C9) Chew R.L. "Note on Timing Simulation of a Large Asynchronous Computer" Comput. J. Vol. 7 No.2 July 1964 pp 122-123

(C10) Chu W.W. "A Study of Asynchronous Time Division Multiplexing for Time Sharing Computers" AFIPS conf. proc. Vol. 39 pp 669-678, Fall Joint Conf. Las Vegas, Nevada, Nov.1971

(C11) Cockrum J.S., Crockett E.D. "Interpreting the Results of a Hardware System Monitor" AFIPS SJCC 1971 pp 23-38

(C12) Coffman E.G. et al "Analysis of Scanning Policies for Reducing Seek Times" S.I.A.M. J. on Computing, 1972, 1, p 269

(C13) Coffman E.G. et al "Waiting Time Distributions for Processor Sharing Systems" JACM Vol. 17, No. 1 pp 123-130, June 1970

(C14) Cohen E. "Symmetric Multi-Mini-Processors: A Better Way To Go?" Comput. Decis pp 16-20, Jan. 1973

(C15) Cohen L.J. "S3, The System and Software Simulator" Digest 2nd. Conf. on Application of Simulation, Dec. 1968, New York

(C16) Cole G.D. "Performance Measurements on the ARPA Computer Network" Proc. 2nd Symp. Opt Data Comms, 1971 pp 39-45

(C17) Colin A. J.T. "Introduction to Operating Systems" Elsevier Monographs

(C18) Comtre Corps., Sayer. A.P. Ed "Operating Systems Survey" Brincerton Averbach 1971

(C19) Control Data Corps "CDC-SCOPE User's Guide "Software Documentation Sunny Vale, Calif.

(C20) Control Data Corps. "Control Data STAR Computer System" Hardware Manual, 602 5600 Revision 01 1971

(C21) Conway M.E. "A Multiprocessor System Design" AFIPS Conf. P100 Vol.24 1963, pp 139-146

(C22) Cooke M.A. "Measuring a V.S. System" European Conf. on C.P.E., Eurocomp, London, Sept.1976 p 29

(C23) Corbato F.J., Vissostsky V.A. "Introduction and Overview of the Multics System" Proc. AFIPS Joint Conf. 1965 pp 185-196

(C24) Cox P.R. "General System Organisation of Multi-Processor Configurations" Software 70, Sheffield, England pp 33-40, 1970

(C25) Critchlow A.J. "Generalized Multiprocessing and Multiprogramming Systems" AFIPS Conf. Proc Vol.24 1963 pp 107-126

(C26) Curtain W.A. "Multiple Computer Systems" Advances in Computers Vol. 4, Alt. F and Rubinoff M. (Eds) 1963 pp 245-303

(C27) Cussens C.J., Broadribb J.P. "A Configuration Modelling System" On-line Conf. on Computer System Evaluation Brunel Univ. Sept.1973

(D1) Dahl O.J. Nygaard K. "Simula-An Algol Based Simulation Language" CACM 9, 9, 1966 pp 670-678

(D2) Daly D. et al "Measurement of Computer Installation Effectiveness" Sigcosim Nesltr April 1971

(D3) Davis R.L. et al "A Building Block Approach To Multiprocessing" Proc. SJCC pp 685-703, 1972

(D4) Deniston W.R. "SIPE: A TSS/360 Software Measurement Technique" Proc. 24 ACM Nat. Conf. 1969 pp 229-245

(D5) Denning P.J. "Resource Allocation In Multiprocessing Computer Systems" Ph.D. Thesis. Dept. Electrical Engineering. MIT, Cambridge, Mass., June 1968

(D6) Denning P.J. "The Working Set Model For Program Behaviour" CACM Vol. 5 No.11 pp 323-333 May 1968

(D7) Denning P.J., Muntz R.R. "Queuing Theoretic Models" ACM/NBS. Computer Performance Evaluation, Workshop, San Diego, 1973 p 119 NBS Special Publication 406 NBS Washington D.C. 1975

(D8) Dennis J.B. "Programming Generability, Parallelism and Computer Architecture" Proc. IFIP. Congress 68 Booklet C, Software 2 pp C1-C7, North Holland Publishing Co., Edinburgh, England, August 1968

(D9) Dennis J.B. Van Horn E.C. "Programming Semantics for Multiprogrammed Computation" CACM 9, 3 March 1966 pp 143-155

(D10) Deutsch P., Grant C.A. "A Flexible Measurement Tool For Software Systems" Proc. IFIP Congress, 71 Ljubljana, Vol. 7A-3, on Computer Software pp TS-3-1 - TA-3-6

(D11) Dijkstra E.W. "The Structure of the T.H.E. Multiprogramming System" ACM Symp. on Operating System Principles 1967. Gothenburg, Tenn.

(D12) Dijkstra E.W. "Cooperating Sequential Processes" Programming Languages. Genuys (Ed) Academic Press. N.Y. 1968 pp 43-112

(D13) Dijkstra E.W. "Hierarchial Ordering of Sequential Processes" Acta Information 1, 2, 1971 pp 115-138

(D14) Dijkstra E.W. "A Constructive Approach to the Problem of Program Correctness" BIT 8 pp 174-186, 1968

(D15) Dijkstra E.W. "Notes on Structured Programming" T.H.E. report 2nd Edition 1970

(D16) Dinardo G.P. "Computer System Performance Factors at Mellan Bank" ACM/NBS Computer Performance Evaluation Workshop San Diego 1973, p27 NBS Special Publication 406 NBS Washington P.C. 1975

(D17) Doherty W.J. "Scheduling TSS/360 For Responsiveness" Proc.AFIPS FJCC 1970 pp 97-111

(D18) Drey fus P. "France's Gamma 60" Datamation 4, p 34 1958

(D19) Dreyfus P. "Programming on a Concurrent Digital Computer" Notes of Univ. of Michigan 1961. Engineering Summer Conf. on Theory of Computing Machine Design

(D20) Droughton E, et al "Programming Considerations for Parallel Computers" Courant Institute of Math. Sciences, report IMM 362

(D21) Drummond M.E. Jnr. "Evaluation and Measurement Technique for Digital Computer Systems" Prentice Hall, Englewood Cliffs 1973

(D22) Drummond M.E. Jnr. "A Perspective On System Performance Evaluation" IBM Systems J. Vol. 4 1962 pp 252-63

(D23) Dumas R.K. "The Effects of Program Segmentation on Job Completion Times In a Multiprocessor Computing System" Dig. of 2nd Conf. on Application of Simulation; SHARE/ACM/IEEE/1 CL, Dec. 1968, New York

(E1) Eckert J.P. et al "Design of Univac Lare System 1" Proc. Eastern Joint Conf. 1959 pp 59-65

(E2) Eckstein R.T. "Getting Started In Computer Performance Evaluation" 11th Meet of Computer Performance Evaluation Users Group Oklahoma City, Sept. 1975 No.6413 35F PB252174

(E3) EDP Analyser "Savings from Performance Monitoring" Sept. 1972

(E4) Edwards B.F. "Systems Performance in Bureau Computing" BCS Conf. on Computer Performance, Univ. of Surrey, Sept. 1972 p 78

(E5) Emshoff, Sisson "The Design and Use of Computer Simulation Models" MacMillan 1970

(E6) England D.M. "Critical Requirements For Multiprocessor System Design" Info Software, Infotech, Maidenhead, Berks., England 1974

(E7) Enslow P.H. Jn. ED. "Multiprocessors and Parallel Processing" Comtre Corp. Wiley Interscience Sept. 1974, Index 0-471-16735-5

(E8) Enslow P.H. Jn. ED "Multiprocessor Organisation A Survey" Computing Surveys Vol. 9 No. 1 March 1977

(E9) Estrin G. et al "Modelling, Measurement and Computer Power" AFIPS SJCC 1972 pp 725-738

(E10) Estrin G. et al "Snuper Computer - A Computer in Instrumentation Automation" AFIPS SJCC 1967 Vol. 30 pp 645-656

(E11) Everett R.R. et al "Sage; A Data Processing System For Air Defence" FJCC pp 148-155, 1957

(F1) Fabry D.S. "Dynamic Verification of Operating System Decisions" CACM Vol. 16 No.11 Nov. 1973 pp 659-688

(F2) Farber D.J. "The System Architecture of a Distributed Computer System - Reliability Features" Technical Report, Univ. of Calif. Irvine 1970

(F3) Farber D.J. "An Overview of Distributed Processing Aims" Proc. 8th Annual I.E.E.E. Computer Soc, Internat. Conf. Feb. 1974

(F4) Farber D.J. "Software Considerations in Distributed Architecture" Computer Vol. 7 No. 3 pp 31-35, March 1974

(F5) Farber D.J. "Distributed Data Bases - An Exploration" Tech. Report, Faine, Farber and Gordon Inc. Passadena Calif. 1974

(F6) Farber D.J. "The Status of the Distributed Computer System" Information and Computer Science Dept. Univ. of Calif. Irvine 1975

(F7) Farber D.J., et al "The Distributed Computer System" Proc. 7th Annual I.E.E.E. Computer Soc. Internat. Conf. pp 31-34, Feb.1973

(F8) Farber D.J., Heinrich F.R. "The Structure of the Distributed Computer System" - the File System "Proc. Internat. Conf. on Computer Communications pp 364-370, Oct.1972

(F9) Farber D.J., Larson K.C. "The Structure of the Distributed Computer System" - Software Symp. on Computer Communications Networks and Teletraffic, Polytechnic Institute of Brooklyn, April 1972

(F10) Farber D.J., Larson K.C. "The Structure of the Distributed Computer System - the Communications System" Proc. Symp. on Computer Communications Networks and Teletraffic, Polytechnic Inst. of Brooklyn, April 1972

(F11) Farber D.J. "A Ring Network" Datamation Feb. 1975

(F12) Feeley J.M. "A Computer Performance Monitor and Markov Analysis for Multiprocessor System Evaluation", Statistical Computer Performance Evaluation, Freiberger (Ed), Academic Press, N.Y. 1972 pp 165-225

(F13) Fenichel, Grossman "An Analytic Model of Multiprogrammed Computing" AFIPS S.J.C.C. 1969

(F14) Fernandez E.B. "Restructuring and Scheduling Parallel Computations" Proc. 5th Annual Asilomar Conf. on Circuits and Systems, Pacific Grove, Calif, Nov.1971

(F15) Fine G.H., McIsaac P.V. "Simulation of a Time-sharing System" Management Science Vol. 12 No.6 pp B180-B194, Feb.1966

(F16) Firestone R.M. "Parallel Programming: Operational Model and Detection of Parallelism" Ph.D. Thesis, New York Univ., May 1971

(F17) Foley "A Markovian Model of the University of Michigan Executive System" CACM Vol. 10 No. 9 1967

(F18) Fox D., Kessler J.L. "Experiments in Software Modelling" AFIPS Conf. Proc. Vol. 31 1967 FJCC pp 429-436

(F19) Frank "Analysis and Optimization of Disc Storage Devices for Time-Sharing Computer Systems" JACM Vol. 16 No.14 1969

(F20) Freeman D.N. "IBM and Multiprocessing" Datamation pp 92-109, March 1976

(F21) Freiberger W. (Ed) "Statistical Computer Performance Evaluation" Academic Press 1972

(F22) Fuller "An Optimal Drum Scheduling Algorithm" IEEE Trans. on Elec. Computers, Vol. 21 No.11 1972

(F23) Fuller S.H. et all "Computer Modules: An Architecture for Large Digital Modules" Dept. of Comp. Sci. and Elec. Eng., Carnigie - Mellon Univ. Oct. 1973 pp 231-238

(G1) Gaver D.P. "Probability Models for Multiprogramming Computer Systems" JACM Vol. 14 pp 623-638 July 1967

(G2) Gehringer E.F., Schwetman H.D. "Run-time Characteristics of a Simulation Model" 4th Proc. of Symp. on Simulation of Computer Systems, NBS, Boulder Colo, SIGSIM/ACM, Vol. 7 No. 4., July 1976 pp 121

(G3) Gibson C.T. "Time-sharing in IBM Systems/360 Model 67" Proc. AFIPS 1966 Joint Conf. Spartan Books pp 61-78

(G4) Gibson J.C. "The Gibson Mix" Report TR 00.2043. IBM Systems Devel. Div. Poughkeepside, New York, 1970

(G5) Gilmore P.A. "Structuring of Parallel Algorithms" JACM 15, 2, April 1968 pp 176-192

(G6) Glading D.G. "Performance Measurements" On-line Internat. Conf. on Computer System Evaluation, Brunel Univ. Sept.1973

(G7) Gomaa H., Lehman M.M. "Performance Analysis of an Interactive Computing System in a Controlled Environment" On-line Internat. Conf. on Computer System Evaluation, Brunel Univ., Sept.1973

(G8) Conzales M.J. "A Decentralized Parallel Processor System" Ph.D. Thesis Dept. Elec. Eng. Univ. of Texas at Austin, Dec.1971

(G9) Gonzales M.J., Ramamoorthy C.V. "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs" AFIPS FJCC 1969

(G10) Goos G. "Some Basic Principles in Structuring Operating Systems" Operating System Techniques 1972

(G11) Goos G., Hartmanis J. (Ed) "Operating Systems" Proc. Internat. Conf. Rocquencourt, April 1974

(G12) Gordon G. "System Simulation" Prentice Hall, Englewood Cliffs N.J.1969

(G13) Gordon W.J., Newell G.F. "Closed Queueing Systems with Exponential Servers" ORSA Journal Vol. 15 No. 2 pp 254-265, March 1967

(G14) Gostelow K. "Flower of Control, Resource Allocation and the Proper Termination of Programs" Computer Sci.Dept. Tech. Report No. UCLA-ENG-7179, Univ. of Calif., Los Angeles, Dec. 1971

(G15) Gotlieb C.C., MacEwan G.H. "System Evaluation Tools" Software Engineering Techniques (Buxton and Randell (Eds) pp 93-99 Nato Sci. Affairs Div., Brussells, 1970

(G16) Gotlieb C.C., Metzger J.K. "Trace Driven Analysis of a Batch Processing System" ACM Symp. on the Simulation of Computer Systems 1973, p 215

(G17) Gould I.H. "A Multi-level Digital Simulator" Univ. of Aston

(G18) Graham R.L. "Bounds on Multiprocessing Anomalies and Related Packing Algorithms" Bell Telephone Lab. Inc., Murray Hill, Proc. AFIPS Conf. pp 205-217, 1972

(G19) Graham R.M. et al "A Software Design and Evaluation System" CACM 1973 p 110

(G20) Greenbourn H.J. "A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System" M.I.T. Dept. Engineering M.Sc. Thesis, MAC-TR-58, Oct. 1968

(G21) Gregory J., McReynolds R. "The SOLOMON Computer" Proc. AFIPS, FJCC 1962, Spartan Books N.Y. pp 97-107

(G22) Grenander V., Tsao R.F. "Quantative Methods for Evaluating Computer System Performance: a Review and Proposals" In "Statistical Computer System Performance Evaluation" Academic Press pp 3-24, 1972

(G23) Grochow J.M. "Real-time Graphic Display of Time-sharing System Operating Characteristics" Proc. AFIPS FJCC 1969 pp 374-379

(G24) Grochow J.M. "Measuring and Monitoring a Multiple-access Computer System" Proc. FJCC pp 379-386, 1969

(H1) Habermann A.N. "Prevention of System Deadlocks" CACM Vol. 12 No. 7 pp 373-377, July 1969

(H2) Hall G., Wisman J. (Eds) "Computer Measurement and Evaluation: Selected Papers from the SHARE Project" SHARE Inc. 1973

(H3) Hansmann F. et al "Modelling for Computer Centre Planning" IBM Sys.J. Vol. 10 No. 4 pp 305-324, 1971

(H4) Hart L.E. "The User's Guide to Evaluation Products" Datamation Vol.16 No.17 1970 pp 32-35

(H5) Hastings T. et al "Conversational System Performance and Measurement" DECUS Conf. Proc. Fall 1969, Nevada pp 191-201

(H6) Heart F.E. et al "A New Mini-computer/Multiprocessor for the ARPA Network" Proc. AFIPS NCC Vol. 42 1973 pp 529-537

(H7) Heinrich F. "Systems Architecture of the Distributed Computer System - the Distributed File System" Tech. Report, Univ. of Calif, Irvine

(H8) Hellerman H., Conroy T.F. "Computer System Performance" McGraw Hill Inc. 1975

(H9) Herman D.J. "SCER: A Computer Evaluation Tool" Datamation Vol. 13, No.2, Feb. 1967 pp 26-28

(H10) Herman D.J., Ihrer F.C. "The Use of a Computer to Evaluate Computers" Proc. SJCC 1964, Spartan Books, Washington D.C. pp 383-395

(H11) Herscovitch H., Schreider T.H. "GPSS III - an Expanded General Purpose Systems Simulator" IBM Sys. J. 4, 3, 1965 pp 174-183

(H12) Hoare C.A.R. "Theory of Parallel Programming" Operating System Techniques, Academic Press 1972

(H13) Hoare C.A.R. "Monitors: An Operating System Structuring Concept" CACM 17, 10 Oct.1974 pp 549-557

(H14) Holland F.C., Merikallic R.A. "Simulation of a Multiprocessor System Using GPSS" IEEE Trans. Sys. Sci. Cyber. 4, 4, pp 395-400, 1968

(H15) Holt A.W., Commoner F. "Events and Conditions" Parts 1-3 Applied Data Research Inc. 450 Seventh Ave., New York, 10001, 1969

(H16) Holt A.W. "Information System Theory Project" Clearing House AD 676-972, Sept. 1968

(H17) Holt A.W. et al "Final Report on the Information System Theory Project" Rome Air Development Centre, Applied Data Research Inc. Contract No. AF30 (602)-4211, 1968

(H18) Hooley J.L. "Interactive Computer Simulation as an Aid to Systems Design and Cost Effectiveness Analysis" Dig. 2nd Conf. on Applications of Simulation, Dec. 1968, New York

(H19) Horning J.J., Randell B. "Process Structuring" Univ. of Newcastle upon Tyne 1972

(H20) Howard P.C. "Optimizing Performance in a Multiprogramming System" Datamation Jan 1969, pp 65-67

(H21) Hughes J. "Performance Evaluation Techniques and System Reliability - A Practical Approach" ACM/NBS Performance Evaluation Workshop, San Diego 1973; NBS Special Pub. 406, NBS Washington D.C. 1975 p 87-97

(H22) Hughes J, Cranshaw D. "On Using a Hardware Monitor as an Intelligent Peripheral" El Segundo, C.A., Xerox Corp. 1973

(H23) Hughes K. "SMS-A Methodology for Evaluation" On-line International Conf. on Computer System Evaluation, Brunel Univ. Sept. 1973

(H24) Hughes P.H. "General Constraints on the Performance of Multiprogramming Computer Systems" On-line Internat. Conf. on Computer System Evaluation, Brunel Univ., Sept.1973

(H25) Hughes P.H. "The Interpretation of Performance Measurements on a Multiprogramming Computer System" On-line Internat. Conf. on Computer System Evaluation, Brunel Univ., Sept.1973

(H26) Hughes P.H. "Developing a Reliable Benchmark for Performance Evaluation" Nord Data 72 Conf. Helsinki 1972, Vol. II pp 1259-1284

(H27) Husband M.A. et al "The MU5 Computer Monitoring System" European Conf. on Computer Performance Evaluation, Eurocomp, London, Sept. 1976

(H28) Hutchinson G.K. "A Computer Centre Simulation Project" CACM Vol. 8 No.9 Sept. 1965 pp 559-568

(H29) Hutchinson G.K., Maguire J.N. "Computer Systems Design and Analysis through Simulation" Proc. 1965 FJCC, Spartan Books, Washington D.C. pp 161-167

(H30) Huesmann L.R., Goldberg R.P. "Evaluating Computer Systems through Simulation" Comput. J. 10, 2. 1967 p 150

(I1) I.B.M. "General Purpose System Simulator III User's Manual" form H20-0163 1965

(I2) I.B.M. "IBM 7090/7040 Direct Couple Operating System: Operating Guide" IBM System Reference Library. C28-6384

(I3) I.B.M. "IBM 7090/7040 Direct Couple Operating System: Programmer's Guide" ibid, C28-6382

(I4) I.B.M. "IBM 7090/7040 Direct Couple Operating System: System Programmers Guide" ibid, 228-6383

(I5) I.C.L. 1900 Series "Direct Access Manual" 2nd Edition

(I6) ICL 1900 Series "George 3 and 4 Operation Management"

(I7) I.C.L. 1900 Series "Statistical Analysis System, Mark 2" 1975 p 113

(I8) Info Process Soc. Japan"An Analytic Model For Bus-Connected Multiprocessing Systems and Some Results of Analysis" Vol. 17 No. 5 pp 394-401 1976

(I9) Infotech "Multiprocessor System" State of the Art Report, Infotech, Maidenhead, Berks, England

(I10) Infotech "Computer System Measurement" State of the Art Report, Infotech, Maidenhead, Berks, England

(I11) Infotech "Software Engineering" Infotech State of the Art Report No.11 Maidenhead, Berks., England

(J1) Jackson J.R. "University of San Diego Management Science" No. 1 Oct. 1963

(J2) Jefferey S., Chantker A.F., "Computer Performance Analysis, Facts, Figures and Fancies" Nat. Bureau of Standards

(J3) Jensen E.D. "A Distributed Function Computer for Relative Control" Proc. 2nd Annual Symp. on Computer Architecture, June 1975

(J4) Johnson P.F. Jnr "The Pounds and Pence of Measurement" BCS Conf. on Computer Performance, Univ. of Surrey, Sept.1972 p112

(J5) Johnson R.R. "Needed: A Measure for Measure" Datamatic Vol.16 No.17 Dec. 1970 pp 22-30

(J6) Jordan J.W. "Task Scheduling For a Real Time Multiprocessor" Electronics Research Centre, Cambridge, Mass. NASA, TN-D-5786

(J7) Joslin E.O. "Computer Selection" Addison-Wesley Reading. Mass 1968

(J8) Juliuson J.E., Mowle F.J. "Multiple Micro Processors with Common Main and Control Memories". IEEE Trans. C-22 No.11 Nov.1973

(K1) Karp R.M., Miller R.E. "Parallel Program Schemata" Journal of Computer and System Sciences Vol. 3 No.4 pp 147-195 May 1969

(K2) Karush D.A. "Two Approaches for Measuring the Performance of Time Sharing Systems" Symp. on Operating System Principles. Proc. Princeston Univ. Oct. 1969 N.Y. ACM Sigops 1971

(K3) Kasper H., Miller D. "Jobs and Jobstream Modelling in the T.R.W. Timesharing System Simulation" Symp. on the Simulation of Computer Systems 11, June 1974 pp 94-121

(K4) Katz J.H. "Simulation of a Multiprocessor Computer System" Proc. AFIPS SJCC Vol. 28 pp 127-139 1966

(K5) Katz J.H. "Optimizing Bit-Time Computer Simulation" CACM Vol. 6 No.11 Nov.1963 pp 679-685

(K6) Katzan H. "Operating Systems: A Pragmatic Approach" Van Nostrand Reinhold, New York 1973

(K7) Keith J. "An Analysis by Software Methods of Processor Time Use" BCS Conf. on Computer Performance, Univ. of Surrey, Sept. 1972 p 32

(K8) Kimbleton S.R. "The Role of Computer System Models in Performance Evaluation" CACM 15, 7 July 1972 pp 686 - 590

(K9) Kimbleton S.R. "An Analytic Framework For Computer Sizing and Tuning" ACM/NBS Computer Performance Workshop, San Diego 1973

(K10) Kiviat P.J. et al "The Simscript II Programming Language" Prentice Hall, Englewood Cliffs, N.J. 1968

(K11) Kiviat P.J., Kolence K.W. "Software Unit Profiles and Kiviat Figures" Performance Evaluation Review, Quarterly Newsletter of SIGMETRICS/ACM 1973 Vol. 2 No. 3 pp 31-36

(K12) Klar "A Counting Monitor For Measurement of Dynamic Programming Behaviour In Memory" Proc. Internat. Computing Symp. German Chapter of ACM 1970 pp 127-141

(K13) Kleinrock L. "Time Shared Systems: A Theoretical Treatment" Proc. Computers and Communications Conf., Sept. 1969

(K14) Kleinrock L. "Certain Analytic Results For Time Shared Processors" IFIP 68, 1968

(K15) Kleinrock L. "Sequential Processing Machines Analyzed With a Queuing Theory Model" JACM Vol. 13 No. 2 1966

(K16) Kleinrock L. "Analytic and Simulation Methods in Computer Network Design" Proc. AFIPS SJCC Vol. 36 pp 569-579 1970

(K17) Kleinrock, Muntz "Processor Sharing Queuing Models of Mixed Scheduling Disciplines For Time Shared Systems JACM Vol. 19 No. 3 1972

(K18) Knight L. "The Measurement and Prediction of the Reliability of Computer Systems" ICL Bulletin Vol. 12 No. 8/9 1976

(K19) Knuth D.E. "SOL - A Symbolic Language For General Purpose System Simulation" IEEE Trans EC-13 1964 pp 401-408

(K20) KnuthD.E. "A Formal Description of SOL" IEEE Trans EC 13 1964 pp 409-414

(K21) Kolence K.W. "A Software View of Measurement Tools" Datamatic Vol. 17 No. 1 pp 32-38

(K22) Kransnow, Merakallio "The Past, Present and Future of General Simulation Language" Management Science Vol. 11 No. 2 Nov.1964

(K23) Kurtzberg J.M., Villani R.D. "A Balanced Pipelining Approach to Multiprocessing on an Instruction Stream Level" IEEE Trans. C22, Feb. 1973 pp 143-153

(L1) Lampson B.W. "A Scheduling Philosphy for Multiprocessing System" CACM 11, 5, 1968 pp 347-360

(L2) Laueson S. "A large Semaphore Based Operating System" CACM 1975 pp 377-389

(L3) Laughlin G.W. "Reducing Run Ratio of a Multiprocessor Software System Simulator" IEEE Trans Computers pp 115-134 1975

(L4) Laver M. "User's Influence on Computer System Design" Datamation. Oct. 1969 pp 107-110, 115 - 116

(L5) Lehman M.M. "Principles of Computer Usage and Control" On line Internat. Conf. on Computer System Evaluation Sept. 1973 Brunel Univ.

(L6) Lehman M. "A Survey of Problems and Preliminary Results Concerning Parallel Processing and Parallel Processors" Proc. IEEE 54 Dec. 1966 pp 1889-1907

(L7) Lehman M., Rosenfeld J.L. "Performance of a Simulated Multiprogramming System" Proc. AFIPS 1968 FJCC Vol. 32 pp 1431-1442

(L8) Leiner A.L. et al "Pilot - A New Multiple Computer System" JACM 6, 3, July 1959 pp 313-335

(L9) Levy H.O., Cann R.B. "A Simulation Program For Reliability Prediction of Fault Tolerant Systems" Internat. Symp. on Fault Tolerant Computing Paris France, June 1975

(L10) Lewis, Shedler "A Cyclic Queue Model of System Overhead in Multiprogrammed Computer Systems" JACM Vol. 18 No. 2 1971

(L11) Liskov B.H. "The Design of the Venus Operating System" CACM 15, 3 March 1972 pp 144-149

(L12) Lucas H. "Performance Evaluation and the Management of Information Services" Data Base, Spring 1972 pp 1-8

(L13) Lucas H.J. JR. "Performance Evaluation and Monitoring" ACM Computing Surveys" Vol. 3 No. 3 pp 79-92, Sept.1971

(M1) MacDougall M.H. "Computer System Simulation: an Introduction" ACM Computing Surveys Vol. 2 No. 3 pp 191-209, Sept.1970

(M2) MacDougall M.H., McAlpine J.S. "Computer System Simulation with ASPOL" Proc. Symp. on the Simulation of Computer Systems, Gaithersbourg, Md.June 1973 pp 93-103

(M3) MacGowan J.M. Jr. "Univac 1108 Instrumentation" NATO Conf. on Software Engineering, Rome Oct. 1969

(M4) Madnick S.E. "Multiprocessor Software Lockout" Cambridge Scientific Centre IBM Tech. Report no 320-2027 April 1968

(M5) Madnick S.E., Donovan J.J. "Operating Systems" McGrawhill 1974

(M6) Maher R.J. "Problems of Storage Allocation in a Multiprogrammed System" CACM Vol. 4. No.10 pp 421-422, Oct.1961

(M7) Markowitz H.M. et al "Simscript: A Simulation Programming Language" Prentice Hall, Englewood Cliffs 1963

(M8) Martin D.F. "The Automatic Assignment and Sequencing of Computation on Parallel Processor Systems" Ph.D. Diss. and Computer Science Dept. Tech. Rep. No. ULLA-ENG-66-4 Univ. of Calif, Los.Angeles 1966

(M9) Martin D.F., Estrin G. "Path Length Computations on Graph Models of Computations" IEEE Trans Computers Vol. C18 No. 6 pp 530-536 June 1969

(M10) Martin F.F. "Computer Modelling and Simulation" Wiley 1968

(M11) Martin J.T. "Design of Relative Computer Systems" Prentice Hall 1967

(M12) Martin J.T. "Systems Analysis for Data Transmission" Prentice Hall 1972

(M13) McIsaac P.V. "Time Sharing Job Descriptions For Simulation" System Development Corp. Document TM-2713 Nov.1965

(M14) McClure R.M. "A Programming Language for Simulating Digital Systems" J. ACM Vol. 12 No. 1 June 1965 pp 14-22

(M15) McConachie M.A., Newman I.A. "Establishing Criteria For the Assessment of Performance" On-line Conf. On Computer System Evaluation, Brunel Univ. Sept. 1973

(M16) McKinney J.M. "A Survey of Analytical Time Sharing Models" ACM Computing Surveys 1, 2, June 1969 pp 105-116

(M17) McLain D.M. "The Effectiveness of Low Level Scheduling Strategies in Unpaged Batch Systems" European Conf. on C.P.E., Eurocomp. London Sept. 1976 p 225

(M18) Mealy G.H. "Operating Systems", Programming Systems and Language. Saul (Ed)

(M19) Meilander W.C. "Staran, an Associate Approach to Multiprocessor Architecture" Info Tech. Report "Multiprocessor System - State of the Art" 1976/77 Maidenhead, Berks

(M20) Merikallio R.A., Holland F.C. "Simulation Design of a Multiprocessing System" Proc. AFIPS. FJCC 1968 pp 1399-1410

(M21) Merill H.E.B. "A Technique For Comparative Analysis of Kiviat Graphs" Perf. Evaluation Review, Quaterly Newsletter. SIGMETRICS/ACM 1974 Vol. 3. No. 1 pp 34-39

(M22) Mills H. "Top Down Programming In Large Systems" "Debugging Techniques In Large Systems" R. Rustin (Ed) Prentice Hall 1971

(M23) Mills H. "Chief Programmer Principles and Procedures" Report in FSC 71-S108 IBM Corps. Federal Systems Division, Maryland

(M24) Min Tech. Memo 737 "Comparison of Computer Speeds Using Mixes of Instructions", May 1969

(M25) Morrison R.L. "Computer System Performance Factors, their Changing Requirements". ACM/NBS Computer Performance Evaluation Workshop, San Diego 1973 p 53 NBS Special Pub. 406, Washington D.C. 1975

(M26) Mulroy J.J. "Computer Performance Analysis" Infotch SOA Reports V.8, Maidenhead, Berks. England 1972 pp 400-418

(M27) Murphy R.W. "The System Logic and Usage Recorder" AFIPS Vol. 35 FJCC 1969 pp 219-229

(N1) Nato Sci Comm. Report "Software Engineering" Germany Oct.1966 Nour, Randell (Eds)

(N2) Naylor T.H. et al "Computer Simulation Techniques" Wiley 1966

(N3) Nemeth A.G., Rovner P.D. "User Program Measurement in a Time Shared Environment" CACM Vol. 14 No. 10 pp 661-666 1971

(N4) Nielson "Analysis of General Purpose Time Sharing Systems" Ph.D. Thesis Stanford Computation Centre, Stanford Univ. 1966

(N5) Nielson "An Approach to Simulation of Time Sharing Systems" AFIPS FJCC 1967

(N6) Nielson N.R. "Computer Simulation of Computer System Performance" Proc. 1967 ACM Nat. Conf. pp 581-590

(N7) Noe J.D. "University Education In Computer Measurement and Evaluation" ACM/NBS Computer Performance Evaluation Workshop San Diego 1973 p 159 NBS Special Pub. 406. Washington D.C. 1975

(N8) Noe J.D., Nutt G.J. "Multiprogramming and Multiprocessing System Description" Univ. of Washington, Seattle, National Science Foundation No. LJ 28781

(N9) Noetzel A.S., Herring LAA. "Experience with Trace Driven Modelling" 4th Proc. of Symp. on Simulation of Computer Systems NBS Boulder Colo. SIGSIM/ACM 8285 167, Vol. 7 No. 4 July 1976 p 111

(01) Oconnor T.J. "Analysis of A Computer Time Sharing System - a Simulation Study" Unpublished Doctoral Dissertation Stanford Univ. June 1965

(02) Opler A. "Fourth Generation Software" Datamation Vol. 13 No. 1 Jan. 1967 pp 22-24

(03) Opler A. "Procedure - Oriented Language Statements to Facilitate Parallel Processing" CACM 8, 5 May 1965 pp 306-307

(04) Oren T.I. "General System Theories and Simulation of Large Scale Systems". Computer Science Conf. 1975 Washington D.C. Feb. 1975 p15

(05) Ornstein S.M. et al "Pluribus - A Reliable Multiprocessor" Proc. AFIPS NCC Vol. 44 1975 pp 551-559

(06) Osterberg R.D., Hibling F.J. "Performance Assessment Techniques a User's Viewpoint" BCS Conf. on Computer Performance, Univ. of Surrey Sept. 1972 pp 162-178

(P1) Parnas D.L., Darringer J.A. "SODAS and a Methodology for System Design" AFIPS Conf. proc Vol. 31 1967 FJCC pp 449-474

(P2) Parnas D.L. "On the Criteria to the Used in Decomposing Systems Into Modules" CACM 15, 12, Dec.1972 pp 1053-1058

(P3) Parnas D.L. "More on Simulation Language and Design Methodology for Computer Systems" AFIPS SJCC 1969

(P4) Parnas D.L. "A Technique For Module Specification With Examples" CACM 15, 1972 pp 330-336

(P5) Part of Post Office Invitation to Tender "Performance Curves for Computing Equipment" 1961

(P6) Petri C.A. "Kommunikation Mit Automaten" Schriften des Rheinsch - West Fabischen Institutes Fur Instrumentelle Mathematike Bonn Univ. 1962

(P7) Pinkerton T.B. "Performance Monitoring and System Evaluation" Nato Conf. Report on Software Engineering Germany Oct. 1968, Nour, Randell (Eds)

(P8) Pinkerton T.B. "Performance Measuring In a Time Sharing System" CACM Nov. 1969 pp 600-610

(P9) Pliener A.M. "Simulation For Performance Evaluation" On-Line Internat. Conf. On Computer System Evaluation. Brunel Univ. Sept.1973

(P10) Pooch U.W. "A Dynamic Clustering Strategy in a Demand Programming Environment" 4th Proc. of Symp. on Simulation of Computer Systems, NBS Boulder Colo, SIGSIM/ACM 8285, 167 Vol. 7 No.4 July 1976 p 11

(P11) Porter R.E. "The RW-400 - A New Polymorphic Data System" Datamation 6 p 8 1960

(P12) Presser L. "Multiprogramming Co-Ordination" Computing Surveys Vol. 7 No.1 March 1975

(P13) Pyle I.C. "Some Techniques in Multicomputer System Software Design" Software Practice and Experience 2, 1, pp 43-54 1972

(R1) Raichelson E., Collins G. "A Method For Comparing the Internal Operating Speeds of Computers" CACM 7, 5 May 1966 pp 309-336

(R2) Ramamoorthy C.V., Gonzalez M.J. "Optional Scheduling Strategies in a Multiprocessor System" IEEE Trans on Elec. Computers Vol. 21 No. 2 1972

(R3) Ramamoothy C.V., Gonzalez M.J. "A Survey of Techniques for Recognizing Parallel Processable Streams in Computer Programs" FJCC proc. 35 1969

(R4) Randell B. "A Note on Storage Fragmentation and Program Segmentation" CACM Vol. 12. No. 7 pp 365 - 369 1969

(R5) Rasch "A Queuing Theory Study of Round Robin Scheduling of Time-Shared Computer Systems" J ACM Vol. 17. No. 1 1970

(R6) Raynor R.J., Gwynn J.M. "Minimization of Supervisor Conflict for Multiprocessor or Computer Systems" 4th proc. of Symp. on Simulation of Computer Systems SIGSIM/ACM 8285-167, Vol. 7 No. 4 July 1976 p61

(R7) Reitman J. "Computer Simulation Applications" Wiley 1971

(R8) Robey K.G. "Computer Performance Analysis in Practice" BSC Conf. on Computer Performance, Univ. of Surrey Sept. 1972 p 191

(R9) Robinson L. "Computer System Performance Evaluation Bibliography" IBM Corp. Nov. 1972

(R10) Robinson L., Levitt K.N. et al "On Attaining Reliable Software for a Science Operating System" 1975 International Conf. on Reliable Software, Los Angeles, Calif. Sigplan Notices, June 1975

(R11) Roek D.J., Emerson W.C. "A Hardware Instrumentation Approach to Evaluation of a Large Scale System" Proc. 24th Nat. Conf. ACM 1969 pp 351-367

(R12) Rosenfeld J.L. "A Case Study of Programming For Parallel Processors" CACM 12, 12, Dec. 1969 pp 645-655

(R13) Rosenfeld J.L., VillaniR.D. "Micro-Multiprocessing: An Approach to Multiprocessing at the Level of Very Small Tasks" IEEE Trans C-22 pp 143-153, Feb. 1973

(S1) Saltzer J.H., Gintell J.W. "The Instrumentation of Multics" ACM 2nd Symp. on Operating System Principles, Princerton Univ. Oct. 1969 pp 167-174

(S2) Saltzer J.H. "Traffic Control in a Multiplexed Computer System" Ph.D. Diss. Massachusetts Institute of Technology 1966

(S3) Saxena A.R., Bredt T.H., "A Structured Specification of Hierarchial Operating Systems" Record of 1975 Internat. Conf. on Reliable Software Los Angeles, April 1975

(S4) Scherr A.L. "Time Sharing Measurement" Datamation Vol. 12, No. 4 April 1966 pp 22-26

(S5) Scherr A.L. "An Analysis of Time - Shared Computer Systems" Mass. Institute of Technology MAC-TR-18, project MAC June 1965

(S6) Schulman F.D. "Hardware Measurement Device For IBM System/360 Timesharing Evaluation". Proc. 22nd ACM Nat. Conf. 1967 pp 103-109

(S7) Schwetman H.D. "State of the Art: Experiment Design and Data Analysis" ACM/NBS Computer Perf. Evaluation Workshop, San Diego 1973 p103 NBS Special pub. 406, NBS Washington D.C. 1975

(S8) Seals E. "Computer Performance Analysis: Industry Needs" ACM/NBS Computer Perf. Evaluation Workshop, San Diego 1973 p 33, NBS Special pub. 406 Washington D.C. 1975

(S9) Sedgwick R, et al "SPY. A Program to Monitor OS/360" Proc. AFIPS FJCC 1970 pp 119-128

(S10) Sekino A "Performance Evaluation of Multi Programmed Time-Shared Computer Systems" MIT Project MAC Report TR-103 Cambridge, Mass. 1972

(S11) Sennett C.T. et al "George 3 Performance Measurement" Royal Radar Establishment Malvern

(S12) Sharpe W.F. "The Economics of Computers" Columbia Univ. press, New York 1969

(S13) Shaw A.C. "The Logical Design of Operating Systems" Prentice Hall 1974

(S14) Shedler G.S., Yang S.C. "Simulation of a Model of Paging System Performance" IBM System J. 10, 2, 1971 pp 113-128

(S15) Sherlock J.F. "The Simulation of a Multi Computer System" IEEE Trans Computers Vol. C-19, pp 114-117 Nov.1970

(S16) Sherman S.W., Brown J.C. "Trace Driven Modelling: Review and Overview" ACM Symp. on Simulation of Computer Systems 1973 p 201

(S17) Sherman S.W., Brice R.S. "I/O Buffer Performance in a Virtual Memory System" Proc. Symp. on Simulation of Computer Systems. Boulder Colarado Aug. 1976

(S18) Shrimpton W. "A George 3 Case Study With Hardware Monitoring" On-Line Internat. Conf. on Computer System Evaluation Brunnel Univ. Sept. 1973

(S19) Slotnick D.L. et al "The SOLOMON Computer" Proc. AFIPS Fall. JCC 1962 Spartan Books N.Y. pp 97-107

(S20) Smith E.C. Jnr "Simulation in System Engineering" IBM Systems J. Vol. 1 Sept. 1962 pp 33-50

(S21) Smith J. "Computer Simulation Models" Griffin 1968

(S22) Smith M.J. "A Review and Comparison of Certain Methods of Computer Performance Evaluation" Computer Bulletin May 1968 pp 13-18

(S23) Spiegel M.G. "Evaluation of a Large Scale Data Retrieved System STAIRS/AQUARIUS" European Conf. on CPE, Eurocomp. London Sept. 1976 p 189

(S24) Spier M.J. et al "An Experimental Implementation of the Kernel" Domain Architecture" 4th Symp. on Operating Systems Principles New York 1973, p 8-21

(S25) Stangh, Southgate P. "Performance Evaluation of Third Generation Computing Systems" Datamation Vol. 15 pp 181-190 Nov. 1969

(S26) Stanga D.C. "Univac 1108 Multiprocessing System" Proc. AFIPS 1967 Spring Joint Conf. pp 67-74

(S27) Statland M. "Methods for Evaluating Computer Systems Performance" Computers and Automation Vol. 13 No. 2 Feb. 1964 pp 18-23

(S28) Stevens D.F. "System Evaluation of the Control Data 6600" IFIP Congress 68, Software 2 August 1968 pp C34 - C38

(S29) Stevens D.F. "The User/Manufacturer Interface" Computers and Automation, Sept. 1970 pp 25-27

(S30) Streeter D.N. "Centralization or Dispersion of Computing Facilities" IBM Systems J. Vol. 12. No. 3 pp 283-301

(S31) Streeter D.N. "Cost Benefit Evaluation of Scientific Computing Services" IBM System J. 1972 pp 219-233

(S32) Sutherland J.W. "The Configurations: Today and Tomorrow" Computer Decisions, Feb. 1971

(S33) Svobodova L., Mattson R. "The Role of Simulation in Performance Measurement and Evaluation" Intenat. Symp. - ACM - SIGMETRICS - IFIP Working group in Computer Performance Modelling, Measurement and Evaluation. March 1976, Harvard Univ. p126

(T1) Teichrow D., Lubin J.F. "Computer Simulation - Discussion of the Techniques and Comparison of Languages" CACM Vol. 9 No. 10 Oct. 1966

(T2) Tecry T. J., Pinkerton T.B. "A Comparative Analysis of Disk Scheduling Policies" Proc. 3rd Symp. on Operating System Principles pp 114-121 Stanford Univ. 1971

(T3) Terman F.E. "A Study of Interleaved Memory Systems by Trace Driven Simulation" 4th Proc. of Symp. on Simulation of Computer Systems NBS, Boulder Colo. SIGSIM/ACM 8285. 167, Vol. 7 No.4 July 1976 p.3

(T4) Thomas J.R. "Introduction to Monitoring and Modelling" On-line Conf. on Computer System Evaluation Brunel Univ. Sept. 1973

(T5) Thorton J.E. "Parallel Operation in the Control Data 6600" Proc. AFIPS 1964 Fall joint Conf. pp 33-40 Spartan Books

(T6) Timmreck E.M. "Computer Selection Methodology" Computing Surveys ACM Vol. 5 1973 pp 199-222

(T7) Tocher K.D. "The Art of Simulation" E.U.P. 1963

(T8) Tryggestad T.N. "An Introduction to the Application of Simulation in Computer Performance Evaluation" 11th meet. Computer Performance Evaluation User's Group. Oklahoma City, Sept.1975

(T9) Tsichritzis D.C., Bernstein P.A. "Operating Systems" Academic Press

(U1) Univ. of Aston - Users Hand Book on Operating Systems, Nov.1976

(U2) Univ. of Aston "George 3 Performance Statistics" 1976/77

(U3) U.S. General Accounting Office "Opportunity for Greater Efficiency and Savings through the Use of Evaluation Techniques in the Federated Government's Computer Operations" Report No. B-115369, Aug. 1972

(V1) Volansky S.A. "Graph Model Analysis and Implementation of Computational Sequences" Ph.D. Diss and Computer Science Dept. Report UCLA-ENG-70-48. Univ. of California, Los Angeles 1970

(V2) Von Almen K.W. "The Simulation of Computer Systems" On line Conf. on Computer System Evaluation, Brunel Univ., Sept. 1973

(W1) Warner C.D. "The Hardware Monitor - An Overview" On line Conf. on Computer System Evaluation, Brunel Univ. Sept. 1973

(W2) Warner C.D. "A Case Study For Hardware Monitors" On line Conf. on Computer System Evaluation, Brunel Univ. Sept. 1973

(W3) Warner C.D. "System Performance and Evaluation:, Past, Present and Future" B.C.S. Conf. on Computer Performance, Univ. of Surrey, Sept. 1972 p 271

(W4) Waters S.J. "Objective of Computer System Design"

(W5) Watson W.J. "The Texas Instrument Advanced Scientific Computer" COMPCON 1972 Digest pp 291-294

(W6) Wecker S. "Investigations of Multiprocessor Mini Computer Systems" Digital Equipment Corps. Mass. Aug. 1973

(W7) Wecker S. "A Building Block Approach to Multi Function. Multiple Processor Operating Systems" AIAA. Computer Network System Conf. Huntsville, Alabama, April 1973

(W8) Whitby - Strevens C. "Research Proposal"Dept. Computer Science Univ. of Warwick, Dec. 1975

(W9) Whitby - Strevens C. "Current Research in Operating System and Computer Networks". Univ. of Warwick Computer Centre Report No. 10, July 1975

(W10) Wichman B.A. "Some Statistics for Algol Programs" National Physics Lab., Central Computer Unit Report No. 11 1970

(W11) Wichman B.A. Private Communication 23rd Nov. 1976.

(W12) Wichman B.A. "Five Algol Computers" Computer J. 15, 1, Feb.1972 pp 8-12

(W13) Wielgosz J. "A System for Simulation of Hardware to Software Allocation and Performance Evaluation" Ph.D. Thesis, Institute of Computer Science, Univ. of London 1974-75

(W14) Wilkes M.V. "A Model For Core Space Allocation in a Time Sharing System" Proc. AFIPS SJCC 1969 pp 265-271

(W15) Williams R.K. "System 250 - Basic Concepts" Proc. Conf. on Computer Systems and Technology 1972 IERE London England

(W16) Wilson D.E. "The PEPE Support Software System" COMPLON 1972 IEEE Computer Soc. Internat. Conf. pp 61-64

(W17) Wiltsher C.D. "Performance Improvement in a Multi Access System" BCS Conf. on Computer Performance Univ. of Surrey, Sept. 1972 p 281

(W18) Winograd J, et al "Simulation Studies of a Virtual Memory Time Shared Demand Programming Operating System" 3rd ACM Symp. on Operating System Principles Stanford Univ. pp 149-155, Oct. 1971

(W19) Wirth N. "A Note on Program Structures For Parallel Processing" CACM 9, 5 May 1966 pp 320-321

(W20) Wirth N. "On Multiprogramming, Machine Coding and Computer Organisation" CACM 12, 9, Sept.1969 pp 489-498

(W21) Wisniewska A.Z. "ICL Real Time Simulation Model" B.C.S. Conf. on Computer Performance, Univ. of Surrey, Sept. 1972 p 289

(W22) Witt B.I. "M65: An Experiment in OS/360 Multiprocessing" Information System Symposium Sept. 1968 Washington D.C.

(W23) Wood P.E. Jr. "Interconnection of Processors and Memory in the Multiprocessor System" ERC Memo KC-T-041, Feb. 1968

(W24) Wulf W. A., Bell C.G. " M.M.P. - Multi - Mini - Processor" Proc. AFIPS FJCC Vol. 41 1972 pp 765-773

(W25) Wulf W.A., et al "HYDRA: The Kernel of a Multi-Processor Operating System" CACM Vol. 17 No. 6 June 1974

(Z1) Ziegler "Towards a Formal Theory of Modelling and Simulation: Structure Preserving Morphisms" J ACM Vol. 19 No. 4 1972

(Z2) Zurcher F.W., Randell B "Iterative Mult-level Modelling - a Methodology for Computer System Design" Proc. IFIP Conf. 1968