Short title for lettering...

MULTIPLE REGIONS FOR THE MCGILL-RAX TIME-SHARING SYSTEM

MULTIPLE USER REGIONS FOR THE MCGILL-RAX TIME-SHARING SYSTEM---JUSTIFICATIONS AND METHODS OF IMPLEMENTATION

Presented as a requirement for the degree of
MASTER OF SCIENCE (COMPUTER SCIENCE)
by

Roy Watt Miller
McGill University
Montreal, CANADA
September, 1971

A B S T R A C T

This paper sketches the development of a time-sharing system known as MCGILL-RAX which is based on an earlier IBM RAX System.

The reader is introduced to time-sharing concepts and the internal operation of the RAX System. Arguments are presented in support of the best method, in this author's judgment, of adding multiple user regions to the system. The required modifications are outlined and an implementation order is suggested.

Measurements performed on the current version of the system are presented as part of the justifications towards making such modifications.

This paper also includes a description of how the seemingly unrelated DOS/OS Compatibility Feature can be used to advantage in an implementation of the multiple user region scheme.

MULTIPLE USER REGIONS
for the
MCGILL-RAX TIME-SHARING SYSTEM
JUSTIFICATIONS and METHODS of IMPLEMENTATION

by

Roy Watt Miller

COMPUTER SCIENCE
MCGILL UNIVERSITY
MONTREAL, CANADA

SEPTEMBER, 1971

PREFACE

Fundamental to the design of a time-sharing system are the techniques used to manage the user regions. Indeed, time-sharing systems are generally classified by the techniques used.

This paper addresses the problem of restructuring RAX to support more than one user region. Decisions must be made as to the proper technique to use and whether the result would justify the work involved. Such decisions are usually left to management and senior systems personnel. For this reason this thesis is written so that it may be of use to those who do not have a detailed knowledge of time-sharing systems. The relevant keywords and concepts are introduced informally in the text. The reader is referred to any of a multitude of "computer dictionaries" for their formal definitions.

Detailed measurements were made of the system's internal performance in relation to the various user requests. The previously unmeasured figures are presented as part of this paper.

Since RAX operates only on IBM Computers, this report restricts its study to these machines. Some of the

techniques and discussions presented here could equally apply to other makes of computers.

The author wishes to express his thanks to Professor W.D. Thorpe, Director and Professor A.M. Valenti, Associate Director of the McGill Computing Centre, for their encouragement during the preparation of this work. Special thanks are due them for *guiding* rather than *directing* the development of MCGILL-RAX. The developers of MCGILL-RAX, themselves major users of the system, were given a free hand in implementing changes which they thought were necessary. This appears to have contributed significantly to the success of the system.

The author wishes to acknowledge the support and encouragement of the Computing Centre staff as a whole. Specific thanks are due Mr. Alan Greenberg who worked together with the author during the past four years developing the MCGILL-RAX Operating System.

Thanks are also due to the seven other MCGILL-RAX installations in North America and Europe. Their support of the system and interest in further extensions were in part the motivation for writing this paper.

The author appreciates the assistance of Mr. Peter Mann, IBM's local representative, who attempted to resolve the author's endless questions. The privilege and experience of working with the IBM Cambridge Scientific Centre and the CP/67 Development Group is acknowledged. Working together with these groups gave the author valuable insight into the design of time-sharing systems on machines with relocation hardware.

The author wishes to thank Miss Linda Doke for her assistance in typing parts of this paper.

TABLE OF CONTENTS

Chapter

1. INTRODUCTION 1

2. HISTORY OF RAX 5

3. RAX SYSTEM INTERNAL OPERATION 8

4. USER REGION MANAGEMENT TECHNIQUES 16

5. CHOICE OF A MULTIPLE USER REGION MANAGEMENT TECHNIQUE FOR RAX 22

6. RELOCATION THROUGH USE OF THE DOS/OS COMPATIBILITY FEATURE 26

7. MODIFICATIONS TO SUPPORT MULTIPLE REGIONS ON RAX 30

8. IMPLEMENTATION SCHEDULE 39

9. JUSTIFICATION CRITERIA 42

10. MEASUREMENT OF MCGILL'S RAX INSTALLATION . . . 51

11. JUSTIFICATION CRITERIA APPLIED TO MCGILL'S INSTALLATION 55

12. CONCLUSIONS 64

APPENDIX A 66

APPENDIX B 72

BIBLIOGRAPHY 83

CHAPTER 1

INTRODUCTION

The evolution of computers has brought with it significant strides in the computer's calculation speed and logical ability. Computing centres have tended to take advantage of these changes by utilizing faster computers to reduce the cost per unit of work. Unfortunately, these large scale systems do not permit the user to "interact" with the computer hardware at a basic level. At typical computer rentals of 10 to 20 dollars a minute, the computer centre can ill-afford to have the computer idle for even a few seconds. Whereas the user originally loaded his own problem into the computer and controlled its solution, this mode of operation is no longer feasible on a cost basis.

To solve a typical problem, the user must organize a deck of cards (his *program*) which completely defines the computations and logical steps to achieve the results, take it to the computing centre, and return perhaps two days later for the results. Thus a calculation which might take 30 minutes at a desk calculator and one second on the computer, could effectively take two days to solve using the above *batch* method.

The typical turnaround time available with these batch systems prevents its use for the class of jobs which require fast turnaround in order to be effective. This class includes applications such as on-line program debugging and information retrieval.

Time-sharing† was developed to solve these dilemmas. Through typewriter-like devices called *terminals*, a user can be directly connected to a computer to allow him to enter his problem and wait for its solution. Perhaps he need only type in the required data to a previously written program.

To make this scheme effective, it must be possible for one user to start and perhaps finish a job before another user's job is ended. If this were not the case, then one user might be able to submit a 10 minute job which would require that the others wait 10 minutes for their turn.

To accomplish time-sharing, the technique is to run a job for a short period of time, or *time-slice*, with a

---

†*Resource-sharing* is perhaps a more descriptive term, but "time-sharing" is the accepted name for this type of operation.

duration not more than several seconds, and then in turn process other jobs during their time-slices and then resume execution of the first. A technique of temporarily storing a partially executed job is also required. This can be done by *checkpointing* (copying) part or all of the job to an external storage device until the *scheduler* selects it for service again.

A specialized supervisory program is required to run a computer system in time-sharing mode. Some of the functions of this program are to:

- schedule the time and perhaps length of each time-slice,
- allow users to modify, control and inspect their job while it is running,
- ensure that one user cannot destroy or otherwise adversely affect the running or the results of another,
- provide adequate diagnostic messages if the user attempts to do something incorrectly,
- ensure that only authorized persons use the system,
- provide adequate accounting of service performed for each user,
- operate in a manner to minimize human intervention.

As an example, some time-sharing supervisors are designed to handle several programs internally before having

to resort to checkpointing. Such a system is said to have *multiple user regions*.

In this paper, the time-sharing supervisor program being discussed is called *MCGILL-RAX*. At the present, it is a *single user region* system. This paper discusses the types of changes required to support multiple user regions and the criteria that might be used to justify such a modification.

The next two chapters are presented to acquaint the reader with the development of RAX and its internal structure.

CHAPTER 2

HISTORY OF RAX

In April of 1964 IBM announced the SYSTEM/360 in an attempt to satisfy all computing needs with one product line.

One of the first installations of these new computers was at the Lockheed Aircraft Plant in Marietta, Georgia. Working with IBM field personnel, they developed by July 1965, a system known as *Remote Access Computing System (RACS)*. It has been claimed that "it was the first time-sharing system built around an IBM SYSTEM/360 installed and working in an industrial environment"[14].

RACS used some of the programs developed for the IBM system called Basic Programming Support (BPS). The BPS-TAPE System was designed to handle jobs read in on cards and to use magnetic tape for work space. RACS added the facility to read input from a disk file created from terminal input and to use disk for work space. The initial RACS system was little more than a remote job entry facility since it lacked the ability to retain (save) programs and to interact with the program while it was running. It did however have time-sliced execution.

In July 1967 IBM took over the responsibility of maintaining the system and changed its name to *RAX*. By this time RAX had the facility of saving programs and the ability to read data from the terminals during execution. These features, coupled with the fact it was the only IBM time-sharing system suitable to run on all announced SYSTEM/360 computers, gave it great marketing potential.

Significant changes to RAX were made at ITT DATA SERVICES. In May 1968, they started running a time-sharing service called "RTS" using RAX as its base. By 1969 they had added many enhancements including the support of languages such as BASIC, FORTRAN-G, COBOL. Their modifications were proprietary due to the nature of their operation.

The University of Rhode Island and Bell Aerosystems Company contributed modifications to RAX which were partially incorporated into later versions of the system.

McGill University Computing Centre started using RAX in October 1967 on an IBM Model 50. At that time the Centre had no experience with either IBM SYSTEM/360 or time-sharing. Two recent additions to the McGill staff were assigned the task of modifying and supporting the system. They were Mr. Alan Greenberg and the author of this paper.

McGill's modifications included supporting Teletype terminals and the Teletype-compatible terminals which soon became popular. The latter had the additional ability to run at speeds typically up to six times that of the Teletype terminals. Additional security was implemented through passwords, fetch protection, private and execute-only files. The data transfer rate of checkpointing was first doubled and then doubled again. MCGILL-RAX was changed to support the IBM O/S FORTRAN-G, COBOL, ASSEMBLER-F language compilers. In April 1971, with the availability of the System/370 Model 155, McGill became one of the first to run time-sharing on this new line of machines.

CHAPTER 3

RAX SYSTEM INTERNAL OPERATION

a) RAX-Hardware Interface

At the centre of a RAX installation is the *central processing unit (CPU)* (see figure 1). The CPU has the arithmetic, logical and control circuitry to perform calculations and direct the system as a whole. The supervisor (RAX in this case) resides in *main storage*†. Parts of main storage are allocated to contain the user region and other parts are set aside for disk and terminal buffers.

RAX issues commands that initiate *input/output (I/O)* with the terminals. The CPU control circuitry in response executes a series of operations that start communications to the channels and then to the specific control unit. The control unit in response may originate signals to be sent through the communications lines to the terminal.

---

† Main storage is sometimes referred to as *core storage* or simply *core*.

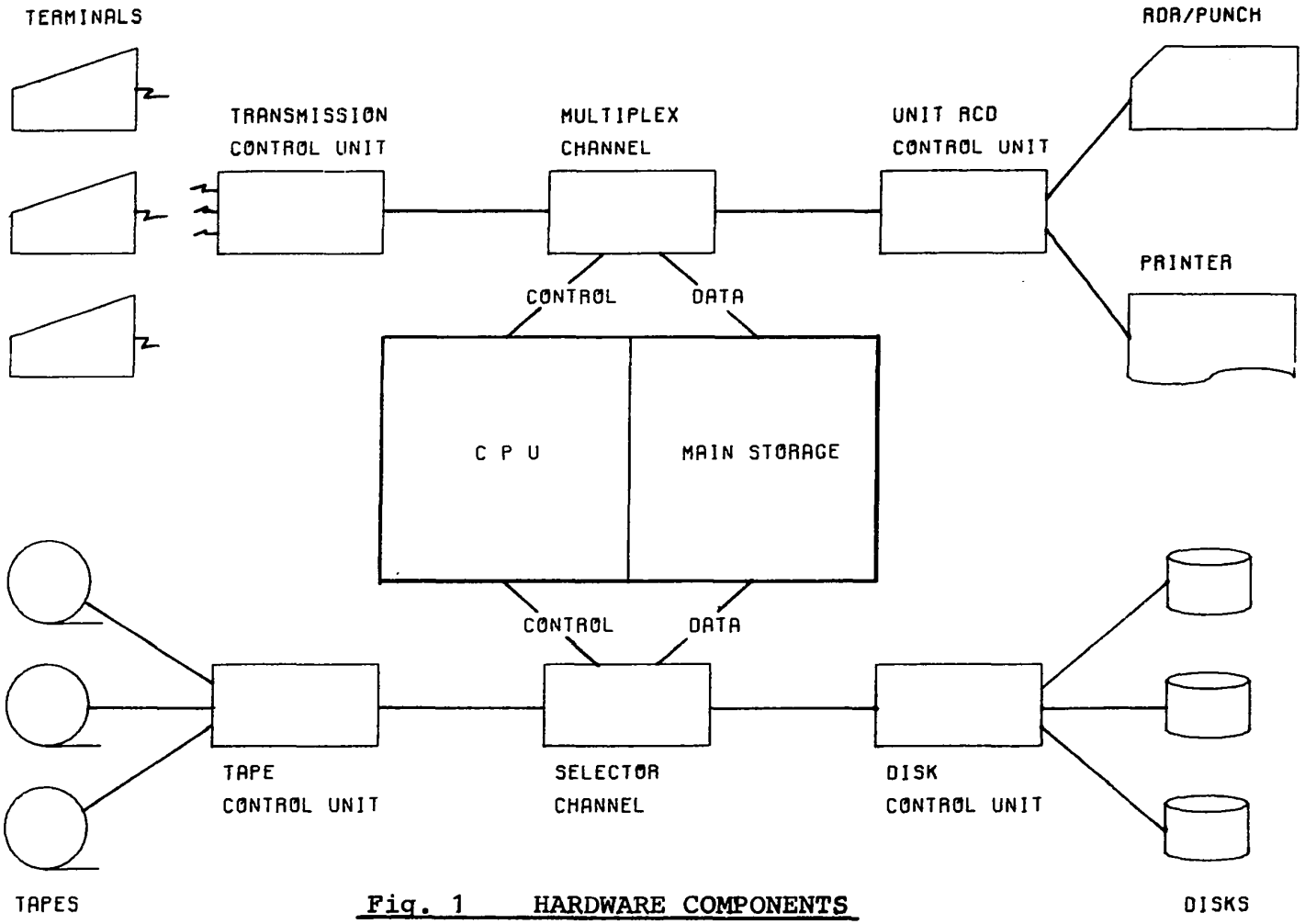


Fig. 1 HARDWARE COMPONENTS

The same sequence is used to control the other I/O devices such as the disk storage units. The details of the signal flow is normally of little importance to the supervisor program which need only be concerned with the general characteristics of the addressed device.

A general knowledge of the characteristics of a disk storage unit are required in this study. Three operations are required to access the data from these devices. First, the access mechanism containing the read/write heads must be mechanically positioned. This is known as the *seek* operation. Second, the *search* operation is started and generally requires a time-lapse while the required data rotates to a read/write head. Finally, the *data transfer* takes place.

It is important in the system design to recognize that different components of the system have vastly different transfer rates. Typical data rates are given below:

TABLE 1

DEVICE	DATA RATE (chars/sec)
user typing	1 or 2
terminal typing	10 to 60
disk storage (2314 type)	312,000
main storage (Mod 65)	10,600,000

b) RAX Data Flow

Figure 2 illustrates the basic data flow in RAX. As the user types characters on his terminal they are transferred into main storage (path 5). After several lines are typed, the block of lines (buffer) is transferred to an input area allocated on disk(4). Conversely to write to the terminal, a block is read from the output area on disk and is sent line by line to the terminal. RAX is informed by the hardware when each line of input or output is completed. It need not be concerned with each character as it is sent over the data paths. The amount of time required to satisfy these terminal I/O requests is minimal in a RAX system. These operations are performed on a demand basis and proceed asynchronously with the rest of the operations in the system.

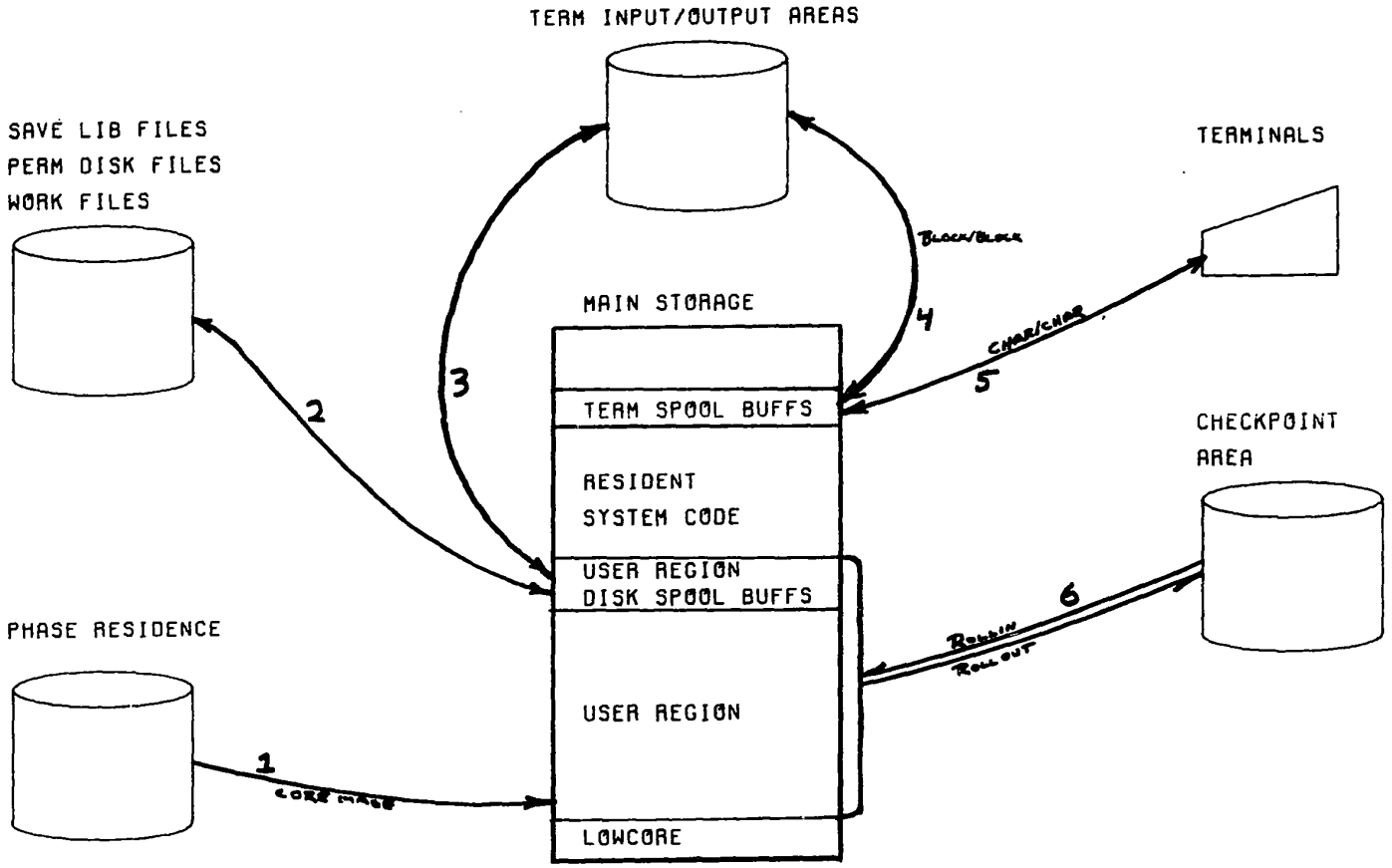


Fig 2. RAX SYSTEM DATA FLOW

To submit a job, the user types in the program from his terminal or indicates by a command line that the program has already been written and is to be found on RAX's save library. When the specification of the problem is complete, the user requests that it be run. At this point all the lines he has typed have been transferred to disk. His job request is put into a queue for the user region. When the job scheduler selects it to be run, an initialization phase is read in *(loaded)* from an area on disk(1). This transient system code determines if a *compiler* must first be read in to translate the program into a series of machine instructions. If this were the case, then it would load the appropriate compiler to service the request(1). The compiler reads in the program from the disks(2,3) and perhaps produces some output to be sent to the terminal via the disk(3). If the compiler has not detected any errors in the program then the generated machine instructions will be loaded into main storage and the user program will be allowed to execute.

If the program executes for more than four seconds it may be stopped and another job started. The system is required to preserve the state of the job until it can be run again. RAX writes the user region in its entirety on disk, thus freeing the user region for another job. This operation is called *rollout* or taking a *checkpoint*(6). When

the job's turn comes up again it is read from the checkpoint device(6) and its execution resumed.

If at any time during this process there are output lines written to disk, they will be printed by the terminal support whose operation continues regardless of the activity in the user region.

Due to the great difference in speeds between the terminal and the CPU, it is possible that the user sees uninterrupted printing of results and is thus unaware that it is being produced in discrete portions.

It is usual for a job running in the user region to request I/O from disk and then be unable to proceed until the information is completely read from disk. In this condition the program must wait for the completion of the I/O and is said to be in *wait state*. A program which requests a read from the terminal is not put into wait state, for this would require that the CPU be idle for at least several seconds, perhaps minutes. Instead RAX will prematurely end the job's time-slice and will schedule another user's program to run in the interim.

A user request for activity on the save library, such as to save a program, is handled in much the same way.

The transient phase which is read in will contain the code to handle the save request in the same fashion that a compiler is loaded to process a job. These types of requests are of short duration and are never time-sliced.

c) User Region Scheduler

This scheduler receives requests for the user region and records the time and type of request. Records of jobs in the checkpointed state are also maintained. The scheduling philosophy is to service short utility requests first, such as sign-on, save, etc. It will then service the user who has the least amount of output waiting to be printed and who has been waiting for CPU service for the longest time.

CHAPTER 4

USER REGION MANAGEMENT TECHNIQUES

The number of user regions and the techniques for managing them form a integral part of any time-sharing system. This chapter discusses five management techniques that could be used.

The simplest technique involves the single user region. This scheme's main disadvantage is its excessive unproductive wait time. MCGILL-RAX rarely is in I/O wait state for longer than 0.06 seconds. Individually, these portions of time are too short to justify checkpointing the user's program, which typically takes 0.6 seconds. However, these short time quanta may add up to as much as four hours a day. Additionally up to two hours are accumulated daily waiting for the checkpoint operation to be complete.

Adding other regions to this scheme would allow the supervisor to switch control momentarily to process another region's job when one goes into wait state. Such a modification increases the complexity of the system as discussed below.

Suppose Job A and B are running in a two region system. Now Job C is to run. Clearly Job A or B must be checkpointed to allow room for this new job. Let us suppose Job A is the one to be checkpointed. At some later time Job A will have to be read in again into one of the user regions. If the computer is any of the IBM SYSTEM/360 series (except the Model 67) the job will have to be returned to the same region it previously occupied. This is a requirement because the program will probably rely on absolute main storage addresses. This restriction could result in a situation where one region is empty while other jobs are waiting to run in the other region. Thus, in an effort to make use of wait time, it might be that the system still has significant amounts of unproductive wait time; and in addition, it has a block of main storage idle.

One way to solve this problem is to ensure that the programs do not have specific address dependencies or perhaps, if they must, then to make sure all addresses are in well-defined locations so that the user region scheduler might change these addresses before dispatching the program in another region. Certain high-level language compilers such as FORTRAN and PL/I have been written to run in such an environment and to produce object code adhering to these specifications. Writing programs in assembler language for these systems would be beyond the capability of almost all

users. The wide variety of application programs that have been developed over the years to run on batch systems could not be easily adapted to run in this environment since many of them are written, at least partially, in assembler language.

The computer hardware could be changed to assist in translating addresses so that the program appears to be always operating in the same user region. Such hardware exists on a SYSTEM/360 Model 67 and certain models of the SYSTEM/370 line.

The hardware on the Model 67 additionally provides the facility of mapping the user program into sections (called *pages*) which may occupy non-contiguous core locations. The programming system maintains a correspondence table (*page table*) which is used by the hardware to determine the physical main storage locations represented by the *virtual* addresses generated by the user program. It is also possible to indicate that certain pages are not in main storage. References that map to these entries will cause an interruption. This can be used to inform the supervisor of the need to obtain the required page from an auxiliary storage device such as a disk or drum. It is also possible to setup a page table which represents a range of virtual main storage addresses which

exceeds the size limitation of the machine's physical main storage. These features allow a time-sharing system to be designed to run without fixed user regions and to permit the user program to be of almost unlimited size. The supervisor would attempt to write only the most inactive pages to the auxiliary storage. Depending on how much real core is available and the number of jobs running, it may occur that only a small portion of the job is in real core. If the program refers to a section of the program which has been paged-out, the supervisor will delay execution until the required page is read in.

The current implementation of relocation does not permit the I/O to use the address translation. Instead it requires that the supervisor provide the real core addresses when performing the I/O and to ensure that the core pages involved remain there until the I/O is complete.

The five techniques discussed above are reviewed in Table 2.

In summary, a single user region can make no use of wait time. To utilize this wait time more user regions are necessary. Thus at the expense of main storage the unproductive wait time may be reduced. To make more efficient use of main storage, a paging system might be

Table 2 USER REGION MANAGEMENT TECHNIQUES

	DESCRIPTION	NOTES	SPECIAL† USER SOFTWARE	SPECIAL MACHINE HARDWARE	EXAMPLES
A	Single user region	all job requests use same region	no	no	RAX
B	Multiple user -user region dependent	checkpointed job must return to the same region it used before	no	no	TSO (current version)††
C	Multiple user region-software assisted		yes	no	CALL/360 *APL*\360
D	Multiple user hardware assisted		no	yes-only simple relocation needed	
E	Variable user region-core on demand		no	yes-full relocation required	CP/67 TSS

†The word "special" here means non-standard as compared with batch systems

††It is rumored that this system is designed for category D or E.

implemented. Unfortunately if such a system is not closely controlled it could mean that a job which needs a large amount of core but does little I/O could become a job which requires the system to do a lot of I/O on its behalf to bring in pages. This would increase the unproductive wait time for the system as a whole. Thus, this sophisticated time-sharing technique could turn out to be even less efficient than a single user region system. Ideally faster I/O devices would make this technique more attractive. Note, however, that as the speed of I/O approaches something infinitely fast, the need for anything but a single user region approaches zero!

CHAPTER 5

CHOICE OF A MULTIPLE REGION MANAGEMENT TECHNIQUE FOR RAX

This chapter concerns the problem of choosing the best multiple user region management technique for a RAX System. A discussion of the merits of such a change is found in Chapter 11.

Modifying an existing system places certain constraints on the "best" choice to ensure that such a change does not adversely affect the current users. Specifically:

a) no user programs should have to be rewritten,

b) no facilities should be removed or curtailed. For example, the user should not have a smaller amount of storage in which to run his program,

c) the system should not be made inherently less reliable,

d) the cost of running a program should not increase without some compensating return justifiable to the user.

Referring to Table 2, we can at once eliminate choice A as it is the single user region scheme used at present. Choice C can also be eliminated as it does not satisfy the above criterion a or b.

Method B would seem at first to be the easiest choice. Unfortunately, RAX due to its BPS background, presents two major obstacles in the implementation of this scheme. The first being that the BPS Compilers used in RAX can only run in the first 65,936 (64K) bytes of core. This would restrict their operation to the first user region. This restriction could be eliminated by removing the BPS Compilers and substituting ones which did not have this restriction. This change would have to be done with great care in order not to affect the current users.

To further confound the issue, RAX has countless dependencies on the assumption that the user region always starts at the same location. The major one is that all phases are in non-relocatable (core-image) format.

Method D does not have the restrictions of B since a relocation scheme could be set up so that all the user regions appear to have the same fixed starting address. The RAX supervisor would then be responsible for determining the real core address from the virtual one that the job running in relocation mode would provide. It should be noted that running in "relocation mode" would slow the execution of

instructions by perhaps 10 per cent†.

Method E would have approximately the same relocation degradation of 10 per cent and in addition some due to paging. Jobs might get faster response in a paging environment but it must be stressed that the CPU times for these jobs would never be better than those under methods A,B or D. The evaluation of this method lies in the determination of whether a paging or a swapping scheme would be the more expedient. The answer is dependent on the characteristics of the jobs run on such a system. Many investigations have been made to determine whether paging is the better scheme[18]. In general, the conclusion is that programs may be written which are much faster in a paging scheme, but if care is not taken, could be considerably slower. To make a study of whether the typical RAX program would be best swapped or paged would require a study of core utilization during a time-slice. These figures theoretically could be obtained through trace programs but to do a thorough study would require tracing through all the user region requests for an entire day--an unthinkable task. Thus, for practical reasons such a study requires the hardware to determine what section of the user region has

---

† It could be possible to run one of the regions without relocation, though this would impact the efforts to achieve consistency in CPU accounting data.

been referenced or modified during a time-slice. This hardware is presently limited to an IBM SYSTEM/360 Model 67.

Having discussed the various alternatives, a choice must now be made. In the author's opinion, Method D is easier to implement than Method B. It is almost impossible to replace existing compilers without impacting some users, since each compiler has its own peculiarities and limitations. Method D puts all the changes in the system. This would appear to make it less likely to adversely affect the user. Method E is attractive but its effectiveness is difficult to measure. It also requires a completely new scheduling and core management system whereas Method D does not. Choosing Method D would not preclude the later change to Method E after hardware measurements have determined its effect. The reader should note that it is not necessary to page or swap exclusively and so it might be useful to design a system which would do either depending on some criteria. Method D has the advantage that it requires only a very simple relocation scheme such as is available with the *DOS/OS* Compatibility Feature. It should be noted that this feature is not marketed for use in this manner, though it is sufficient for the requirements of Method D. The next chapter describes how this feature could be used on RAX. This feature is available on the IBM SYSTEM/370 Models 135, 145 and 155, allowing an implementation over the wide range of CPU powers represented by these machines.

CHAPTER 6

RELOCATION THROUGH USE OF THE DOS/OS COMPATIBILITY FEATURE

This chapter discusses the DOS/OS Compatibility Feature and how it might be used to achieve a simplified relocation scheme for RAX.

DESCRIPTION

The DOS/OS Compatibility Feature is a conversion aid implemented on the IBM SYSTEM/370 to assist current users of the DOS Operating System to switch to the OS Operating System. This feature permits an unmodified DOS System to run under the control of OS. To the OS supervisor, the DOS system appears to be a problem program. To the DOS supervisor, it appears that it has the complete control over the machine. Two basic facilities were required to achieve this goal. One was that the DOS supervisor must not be able to control the machine's status or the I/O devices directly, though it must appear that it is doing so. In other words, it must not be able to perform the "privileged" instructions directly. Secondly, the main storage addresses must appear to start from location zero as far as the DOS supervisor was concerned, but not to be

assigned these addresses in reality as it would conflict with OS's use of them.

The feature was implemented by two additional instructions for the hardware and a software interface routine.

In operation, the DOS system is run in a user region of OS together with the software interface routine. The *EXECUTE LOCAL* instruction is provided to enable the DOS system to run in *local execute mode* such that all privileged instructions will be intercepted and control given to the interface program. This routine can then perform the equivalent functions through the facilities available to the OS problem programs.

When in local execute mode, the hardware will transparently add a constant to all generated main storage references. This constant is provided by the interface routine when it issues the EXECUTE LOCAL instruction. Its effect will be to map the addresses into the main storage area set aside in the OS user region to contain the simulated system's main storage. Thus this facility adds the capability of "relocating" a block of main storage.

An additional instruction, *ADJUST CCW STRING*, is provided to aid in the translation of the channel programs from the "virtual" addresses developed in local execute mode to the true addresses required by the channels.

The techniques used in the DOS/OS Compatibility Feature are not unique to this feature and bear resemblance to some portions of the CP/67 Operating System[25]. What is of interest, is that it is now possible to accomplish some form of relocation without using the IBM Model 67.

The performance degradation using the feature is due to two distinct causes. One is the extra cycles that the hardware goes through to add on the relocation constant when it is in local execute mode. The second is due to the requirement of trapping the status control and I/O control instructions issued by the DOS supervisor and their simulation through the facilities available to the OS problem programs. The former may typically cause an overall 10% degradation, while the latter may be expected to degrade up to 300% depending on the programs run.

RAX USAGE

As discussed in the previous chapter, RAX requires only the facility to dynamically set up an addressing scheme such that all the user regions appear to start at the same location. Thus the severe degradation that might result from running the DOS system under the Compatibility Feature would not be present if it is used merely to provide some form of relocation for RAX. Furthermore, it is possible to utilize this feature in a way such that the added system coding could be easily adapted for use in another relocation environment. This would ease the conversion should it be required to support the IBM Model 67 or other similar machines with relocation.

CHAPTER 7

MODIFICATIONS TO SUPPORT MULTIPLE USER REGIONS ON RAX

The parts of RAX that would require modifications to support multiple user regions are explored specifically in this chapter. The presentation here is designed to indicate the magnitude of the entire task rather than the design of each section of the code. The reader is referred to the Bibliography if he requires material which would be of assistance in the design of these changes.

a) Wait Loops

If there is only one user region in the system there can be only one user job running at any time. There will be times that the job will have to wait for the completion of its I/O. It could do this in one of two ways. It could merely go into an instruction loop waiting for the completion of the event to be posted by the I/O scheduler. Alternately, it could inform the supervisor of the need to wait until the completion of the event. This latter technique is the one that is generally used in multiple region systems. RAX uses the first and simpler method. This means that RAX does not go into a true wait state pending the completion of the I/O. As a result, it is

difficult to separate the "wait time" from the CPU time. It also makes it impossible for the system to make use of this time if it had multiple user regions. Thus it is necessary to change RAX to remove these loops and replace them with calls to the supervisor. Once this change has been made, the supervisor can record information on the reason and length of time for the wait conditions. These figures are also required for the statistical recorder described later. Removing these instruction loops requires modifications to about half of the modules in the system and the additional supervisor code to handle the wait requests.

b) Accounting Routines

Currently RAX job charges are based on the elapsed time the job was in the user region. This scheme is obviously limited to a one user region system. To charge equitably for jobs in a multiple user region system requires a more complex scheme.

It is reasonable to require that the redesign of an accounting system for RAX be done in such a way as to attain charge consistency. This is an important rule that must be kept in mind when designing accounting systems. It may be stated as follows; assuming that the charge rates remain

constant-- *then if a user runs the same job twice, he should be charged the same for both.*

It is very difficult to explain to a user why "the computer worked faster" for one run than the other. Many present accounting systems are lacking in this respect, mainly due to the interference of one job's I/O with another, which can increase the I/O charge for both.

On multiprogrammed systems, job accounting is often done by dividing the charge into *CPU time* and *I/O time*. CPU time can easily be measured. I/O time can be approximated by job wait time. The multiple user region system would attempt to process a second job when one goes into wait state. This second job may use the same devices that the first job did and thus interfere with its I/O requests. Attempts have been made to remedy this problem by separating I/O charge into seek and data transfer time. The most significant interference factor is usually disk arm contention, which causes a multiprogrammed job to spend more time seeking than would be required if it were running by itself. The obvious solution here is to charge I/O time as made up of data transfer time and simulated seek time. The seek time would be calculated by reference to the time it would have taken had the disk's access mechanism not been moved by the other job. The above scheme penalizes the job

which does excessive seeking such as the job which copies from one part of the disk to another part of the same disk. It also charges the user more if he makes several small requests for data as opposed to requesting a large block of data at one time. Accounting systems are usually designed with this philosophy in mind.

On RAX however, almost all the blocking and the choice of where the data is located is beyond the control of the user. This relieves the user of the necessity of becoming familiar with disk packs, seek times and channel programs. Unfortunately, as a consequence, the accounting system for RAX should not penalize the job for badly managing its I/O. Thus an accounting system for RAX can only charge for CPU time, data transfer time and some fixed charge per I/O request. It would be up to the systems support staff to ensure that the user data sets are arranged in an order which minimizes the seek time. With such a scheme the job charge would be reproducible.

The implementation of this charge scheme requires that RAX's internal accounting system be rewritten to maintain separate I/O and CPU charges. As a consequence, RAX's disk I/O scheduler would have to be modified to be able to identify the requests from the user regions and to maintain the I/O charges accordingly.

Changing the charging structure also involves changing the installation's billing program and informing the users of the new charge scheme. For some installations this might be a non-trivial task.

c) Contiguous Core

When hardware relocation is in effect, each user region can appear identical with respect to addressing. In fact, it should be impossible for the program to determine its real location in main storage. This is of course, the whole significance of using relocation. The user region must contain all the information that is necessary to the program running in it. It is therefore required to gather all information specific to a user region in a contiguous area of core storage.

Parameters such as time and date are currently maintained by the system outside of the users region and can be directly addressed from a user program. The users' programs obtain this information via routines provided to the user and maintained by the system staff. It should therefore only require that these be altered to request the information via some other technique, such as a Supervisor Call Instruction (SVC).

d) Fetch Protect

To ensure that the user does not accidentally or otherwise refer to locations outside his region, all such core should be *fetch protected*. This protection is provided as part of the CPU's hardware and is designed to prohibit access to any selected areas of main storage (in increments of 2K).

e) Core Limits

At the present time RAX's execution region is about 110K. In the future, it would be advisable to maintain the actual core size used in case it proves justifiable to have other user regions of smaller size. All unused core should be fetch protected to prevent the possibility of some errors only occurring when the job is run in a smaller region.

f) SVC Save Areas

The system may put a job into a wait state pending the completion of the I/O. Typically this wait is done in an SVC handler. In order to make use of this wait time another user region must now be able to be dispatched. However RAX's SVC handlers are *serially re-usable*, that is, one usage must be complete before another is allowed to start.

To make use of this wait time it is necessary that these routines be made *re-entrant* so that another request may be started before the first is completed. In order to maintain the required status of the SVC request it is necessary to have a save area outside the SVC routines, perhaps one associated with each user region.

g) Enqueue, Dequeue

There will be occasions when the system must ensure that only one program is using a resource or modifing a disk record at any given time. Such a condition could occur when a program is being added to the save library. This involves reading a record from disk, updating the record and rewriting it to disk. Clearly a second program must be prevented from performing the same operation until the first is completed. Currently such a condition cannot occur since only one save job can be in operation at a time and it completes the entire save operation before another one can start. In a multiple user region system we would require an enqueuing facility to ensure that the system integrity is preserved.

h) Time-Slice Scheduler

Modifications would have to be made to the current RAX scheduler in order that it could handle more than one region. Additional scheduling parameters would have to be inspected so that the user regions could be used in the optimum fashion.

i) User Region Dispatcher

This would be a new system routine whose responsibility would be to ensure that the user region jobs do not destructively interfere and that the wait times are effectively used. To simplify its design perhaps the technique is to run Job B only when Job A goes into wait state. The identity of Job A and B could be changed dynamically depending on some observed characteristics of the jobs. Attaching a fixed priority to the jobs could not, in general, maximize throughput and response to the user.

j) I/O Dispatcher

This routine would have a similar function to the user region dispatcher except its responsibility would be to the I/O channels.

k) Statistical Recorder

Counters and timers would have to be added to measure wait times and keep records of the types and characteristics of the job requests. Such statistics would be required to justify a change to the multiple user region scheme and to determine its effect after implementation. It would also be indispensable for use in design of the new schedulers.

l) Data Set Reorganization

Based on the channel utilization statistics, certain data sets would have to be moved to different disk modules to reduce the seek time. It would be imperative that this be done if the charge scheme was chosen as recommended above, which would have the effect of making seek time part of system overhead.

CHAPTER 8

IMPLEMENTATION SCHEDULE

This chapter discusses the order in which the modifications outlined in the previous chapter should be made. Estimates of the amount of time required for these changes are not given here as they vary somewhat depending on the installation's experience, manpower, etc. Chapter 11 discusses the time in relation to McGill's installation.

The implementation order should be determined with several considerations in mind. They are that:

a) the users must not be adversely affected at any point,

b) it must be possible to use the measured statistics before and after the change to determine whether, in fact, the implementation had the desired effect,

c) be able to "undo" the changes if they do not initially work or if they have some unexpected side-effect,

d) do as much as possible with as little equipment outlay as possible.

A four phase implementation schedule is suggested below.

Phase 1 - Preparation

Replace all wait loops in RAX with calls to a wait routine.

•Write programs to retrieve the statistics produced by the system and to determine the feasibility of the entire project.

•Arrange the user region so it occupies contiguous core locations and fetch protect all other core from the user programs.

•Write and implement a new accounting system based on CPU charge and a measure of I/O charge. This should be done in parallel with the present accounting system. This change should be done in conjunction with the changes to the time-slice scheduler and the implementation of the user region scheduler.

•Rewrite many, if not all, of the SVC routines which handle the I/O requests to make them re-entrant and put in enqueues and dequeues where necessary.

Phase 2 - Initial Use of Relocation Hardware

•The system should now use the relocation hardware with only one user region. Should any problem develop it should be possible to immediately discontinue this mode.

•Ensure that the new accounting system is functioning smoothly.

Phase 3 - Full Multiple User Region Implementation

•Allow more than one job to run at a time and closely observe the interference between the user regions.

Phase 4 - Optimization

•Based on the statistics gathered up to this point, rearrange the data sets to minimize the seek time and job interference.

•Perhaps make certain phases resident in the unused core.

•Determine the best number of user regions and their optimum size.

A major portion of Phase 1 has been completed. Typical measurements of wait times and other statistics are discussed in Chapter 10.

CHAPTER 9

JUSTIFICATION CRITERIA

In this section an attempt will be made to establish the criteria required to justify the multiple user region support to RAX. In effect, we are attempting to balance the additional cost with an increase in performance. How much importance is attached to the need for a "balance" is installation dependent. One installation may feel that an increased performance is a justifiable cause for an increase in expense. A commercial time-sharing company, on the other hand, would be more interested in a "balance" that would work out to cost them less per unit of chargeable work. This chapter enumerates the criteria which should be considered in such an evaluation. The actual weighing of these factors is left to the installation. Chapter 11, which analyzes McGill's installation, may be used as a guide by others in their evaluation.

COST CONSIDERATIONS

Equipment Cost

The equipment cost is composed of the relocation feature and that of the additional main storage. The extra

storage is required to contain the added user regions and the enlargements of the RAX Supervisor to control them. Additional main storage is typically available in sections of no less than 128K. It is imperative to note that the cost of an additional 128K plus one character is the same as the cost of 256K.

Optimizing the system for the multiple user regions might additionally precipitate the need for more channels and/or additional I/O devices.

The above costs are summarized in Appendix A.

Better Use of Equipment

Considerations must be made as to whether the additional equipment cost might be better used for other functions.

As an example, the 128K needed for a second user region could also be used to hold compilers and other system phases. As described in Chapter 3, these routines are currently read in from disk when they are needed. The unproductive wait time associated with this operation alone accounts for about forty minutes a day at McGill (see Chapter 10). A substantial saving of time could thus be

realized by keeping these routines in main storage at the expense of the additional cost of the core.

Manpower Cost

An additional cost that must be considered is that of the manpower to accomplish this change. An installation might be faced with three alternatives. If the systems support staff is capable of doing the task, then it must be considered whether their time might be better spent in another way. If the system staff is not to make the change then the cost is that of hiring additional help, recognizing that a major portion of their efforts would be directed towards familiarization with RAX rather than with the problem at hand. If the system changes had already been made then it might be best to purchase the change rather than undertake the job of doing it again. This last choice does not present itself at the present since the changes have not been made by anyone else.

PERFORMANCE CONSIDERATIONS

The measurement of performance in a time-sharing system can be divided into two sections. One is the utilization of the computer system, the other the response as seen by the terminal user.

Computer Systems Performance

The question that is of interest here is: "Can the job's unproductive wait time, the swapping time and other overhead be reduced?".

Performance to the User

The user sees the system's efficiency through the response he gets at the terminal. Therefore, when implementing systems changes, their effect on terminal operation should be evaluated. Particularly important is the response time for short requests. Studies have been made[13] that indicate that a user can perceive differences of a tenth of a second in responses in the order of two seconds. For thirty second responses, only differences of several seconds are perceptible.

The response of the terminal itself is also of importance. Some terminals require about one second to unlock their keyboard. Thus if the system reduces its response time to something below this, the difference would be imperceptible to the user.

The performance to a user also includes turnaround time for jobs that have few or no interactions from the terminal. A significant factor is how many minutes of CPU time is demanded by the user per hour that he is connected to the computer. Note that this sets an upper limit to the number of terminals that might be supported. Suppose that the average terminal user requested one minute of CPU time per hour, then it is clear that only sixty such users (at most) could be supported over a period of an hour.

The effect of improving the response time to a terminal user will tend to keep the user busier and hence happier. Beyond a certain point however, the increase in performance would have little or no effect. At this point additional users are needed to justify the system. Whether in fact more users are available is a question that would have to be answered by each installation.

Performance of the User

A study of the effect of response time was made by Robert Miller, a psychologist, working for IBM. His paper[13] emphasizes that the ideal response time is a function of the type of activity requested and cannot, as is often

attempted, be described as a single value of, say, two seconds.

This study demonstrates the complexity of human reaction to varying response time. His findings show that there is not a linear decrease in human efficiency with increasing response times. A ten second response, in certain cases, was found to be no better than one of one minute. This was found to be due to a requirement to respond within four seconds, (two was optimum), in order to preserve the continuity of the user's thought processes. Conversely, when typing in information the user is best not informed of an error until at least two seconds has elapsed.

A popular demonstration of the use of time-sharing is to allow the user to type in his program and have the computer check each line as entered. The system must be able to accept lines as fast as the user can type them in since "obviously" (to the user anyway), the computer is faster than he is. This *line by line syntax checker* should take into account the fact that there is a human requirement for a delay of two seconds if an error is detected. There is also a need for consistency of response time. This is the type of dilemma that should be considered when making system changes. It also illustrates the complexity of designing systems in which human interaction is such an integral part.

Robert Miller also suggests that if no response is expected within several seconds, the system should supply one. In MCGILL-RAX, for example, a message "*IN PROGRESS" is issued after a request to run a job. There is no system requirement to send this message; it is simply there to give the user a sense of partial psychological closure. A close parallel exists when we place a telephone call. The only absolute requirement is to be connected to the required party, though the telephone company decides to artificially induce the sound of a telephone ringing to satisfy the human need.

A study[12] has been made to demonstrate that giving users too good a response may increase their time to solve a problem. Suggestions have been that responses be slowed down for certain activities, thus creating a *lock-out effect*. Unfortunately, no universal criteria can be established for this. Indeed, some commercial time-sharing firms might be quite happy to give the user an excellent response so that he might use the service inefficiently and thus increase his computer charges.

In summary, the effect of improving response will generally allow a user to be more effective. It could allow him to perform more interactive functions. It might reduce

his solution time and hence allow him to be more productive. It could increase his effectiveness by helping him find the errors through debugging aids. This might also have a negative effect in that he would rely on the computer to find his error where a simple glance at his program might determine the cause.

SYSTEMS RELIABILITY AND AVAILABILITY

A batch system which is down for a period of time, let us say an hour, may not have any adverse effect on the usage. The cost of an operator for the additional hour would be small in proportion to the machine cost.

The effect of a time-sharing system being down for one hour is entirely different. The users would notice immediately, often before the operating staff does, that the system is not working. Since time-sharing primarily exists to satisfy the user's immediate requests, the revenue lost by the machine being unavailable is generally irrecoverable. To ask all the users to remain for one hour to recover the lost time is unthinkable. To a commercial time-sharing service this down time may have the side effect that users will go to a competitor who may be just another phone number away.

To the average user the time waiting for the system to come back up is generally time lost. The user generally hopes it will be working in a few minutes and that there it is not much point working on something else in the meantime.

Thus in evaluating a major system change, it is necessary to consider whether such a change would make the system measurably more unreliable. Since the system is not available until it is completely started, modifications that would significantly increase the restarting time would be undesirable.

ADDITIONAL CAPABILITIES

The change to a multiple user region system permits additional facilities to be implemented. For example, a facility could be added to allow a single job to run in two or more concatenated regions, thus permitting the running of programs which exceed current RAX size limitations.

CHAPTER 10

MEASUREMENT OF MCGILL'S RAX INSTALLATION

This chapter discusses the measurements made at McGill University to assess the effect that multiple user regions might have on the system. Of primary interest was the usage of the present user region and the amount of wait time occurring for typical user requests.

In order to measure this time, a substantial portion of Phase 1 (see Chapter 8) was completed by the author. Specifically the instruction loops were replaced with calls to a wait routine. All uses of the user region were recorded in a table in main storage together with various parameters about the usage. Since this table had a limitation of one thousand entries, it was necessary to periodically inspect and summarize the contents of the table. A facility was added to the system to allow a job to request its next time-slice after a specified period of elapsed time.

A program was written to summarize the user request table and certain other parameters and counters of interest in less than one second. The program was activated every

ten minutes. This period was chosen so that it would be short enough to measure the system status on a time basis and long enough so that it would not impact the system's performance.

The results of these measurements are summarized in Appendix B, and described briefly below.

Table 6 and Figure 4 show the proportion of time the user region spends performing its major functions. Notice that the user region is active about 430 minutes a day. The region is in job wait state for about 52% of this time while only 27% of this time, or 115 minutes, was actually used processing instructions.

The next graphs represent a more detailed picture of the activity in the user region. The length of time of each request is shown in Figure 5. Notice that most requests are either for less than 0.2 seconds or go for the full time-slice of 4.0 seconds.

Table 6 showed that the average job was in wait state 2/3 of the time. It is of interest to see whether the amount of I/O wait bears any relation to the length of the user region request. Figure 6 shows that the requests tend to be slightly less I/O bound as they approach the four

second time-slice limit. The plot between four and ten seconds represents the small portion of jobs that are not time-sliced and which can run until they are finished. Such requests are generally due to a long BPS compile job. Figure 5 showed that these requests were few in number.

Table 7 and Figure 7 show the requests for the user region by the job type. Notice that the utility requests which make up 23% of the requests, total only 14% of the time. Notice that a sign-on takes 0.2 seconds to process, but it may force the system to do an additional two checkpoints totaling 0.6 seconds to rollout the current job and then restart it afterward.

When adding another user region, it is of interest to know whether it needs to be as large as the first. Figure 8 shows the proportion of the requests that could be run as a function of the user region size. Notice that about 50% of the requests could run in about 60K or less of main storage.

Figure 9 is similar to the previous plot though it shows the proportion of time a given user region size is needed. Notice that a user region of about 70K could satisfy the total user requirements 43% of the time.

Figure 10 shows the systems activity during a typical day. It is presented here to show that the usage is dependent on the time of day. It can be seen, for example, that most users have a coffee break at 10:30 and go for lunch at 1:00 pm. Batch jobs on MCGILL-RAX can be scheduled to run by the system only when the user region would otherwise be idle. Thus if a batch stream is run during the day, the system utilization would not show such marked time dependencies.

Figure 11 shows the system usage rate per active user. It can be seen here that the typical usage rate is roughly one minute per terminal hour. This limits the number of similar users that can be supported simultaneously to about sixty. The CPU rate by itself is about 0.4 minutes per terminal hour which sets the theoretical maximum of 150 users, assuming other variables are held constant.

CHAPTER 11

JUSTIFICATION CRITERIA APPLIED TO MCGILL'S INSTALLATION

Chapter 9 outlined the criteria that might be used to justify adding multiple user region support to RAX. This chapter will discuss the criteria in reference to McGill's own installation using some of the measurements presented in the last chapter.

COST EVALUATION

EQUIPMENT COST

Currently MCGILL-RAX uses almost all of 256K of main storage available. The unused part is earmarked for future changes other than those required by the multiple user region concept. Therefore, main storage must be added for the new regions. The smallest addition available for our machine is 128K. The author estimates that the additional code, pointers and buffers required to support the multiple user regions would take about 20K. An additional 10K should be left available to provide a margin of safety. Thus a 128K addition to our machine would not be large enough to hold an additional user region of 110K plus the added system code of 20K (minimum) to support the change. An addition of 256K to the current system would be

just large enough to support two additional 110K regions.

Notice that from this investigation that roughly one-half our jobs could be run in a 70K region. If this was the size chosen then we would be left with 28K spare. This space could be used to hold the most frequently used phases that would, otherwise have to be read in from disk†. This latter configuration would appear to be the best initial choice since it involves the least amount of additional equipment.

It would cost about $1,800 per month to add 128k of main storage to our Model 155. The DOS/OS Compatibility Feature which rents for roughly $300 per month would also be needed--for a total of $2,100 per month. This would mean that we would have to be able to justify an additional charge of $100 per working day. At typical commercial charge rates of $10 per elasped minute, it would require an additional chargeable time of 10 minutes per day.

---

†It is estimated that about one half of the 40 minutes a day that is spent loading phases can be saved by taking advantage of this space.

It is interesting to compare this price with that of a similar change on the SYSTEM/360 line. It would be necessary to get a Model 67 which is roughly the same internal speed as the Model 65 and the SYSTEM/370 Model 155. The required change would cost about $14,710, or in other words, require an additional 74 minutes of chargeable time a day (though additional relocation features are available with this hardware).

Manpower Cost

The author estimates about two man-months to complete Phase 1 of the implementation schedule. This estimate assumes the full-time dedication of a systems programmer experienced in making modifications to MCGILL-RAX.

After the completion of Phase 1, the installation could start to benefit from the modifications. The work involved in Phases 2 through 4 could be handled as part of the installation support effort.

The manpower cost of doing the change at McGill would be minimal since the present system staff is capable of doing the job. Considerations must be given as to whether their time might be better spent on another project.

We must also consider the urgency of making these changes. Figure 11 showed that if the usage rate continues at its present level, then, about sixty terminals can be supported in the present configuration. If roughly half of the unproductive wait time could be utilized then over one hundred terminals could be supported. Notice that if a faster CPU were used then not much more throughput could be realized than with our present configuration.

It is also a consideration whether the facilities offered by MCGILL-RAX would satisfy the requirements of the immediate future or whether we must scrap the system and use another one entirely. This latter choice would make the entire change to a multiple user region system a venture of limited value. At the present time we can see no other time-sharing system working on an IBM computer that can satisfy our present time-sharing requirements without substantial increase in cost. Functions not currently supported by MCGILL-RAX could be added with less work than that in implementing and supporting a new system. This can be partially attributed to the fact that supporting another system would inevitably require changes to the supervisor and terminal handlers to satisfy a community with previous time-sharing experience. This is due to the fact that time-sharing users become "attached" to the convenience features of a system. A switch of systems is not sufficient

justification to the user of why the computer, for example, is now incapable of printing his output with tabs or allowing him to skip several lines of output by hitting the attention key.

EFFECTS ON SYSTEMS PERFORMANCE

As mentioned above, it would require an additional 10 minutes of chargeable time to justify the cost of the additional equipment. This time would come out of the 300 minutes of wait time†. It would seem unlikely that the addition of multiple regions could not achieve this meager increase.

It is a momentous task to calculate accurately the amount of wait time that could be utilized by a second region. It would require a detailed simulation study based on the characteristics of the jobs' I/O requests, such as the locations of the data on disk, which disk it used, the amount of data transferred, the time of the I/O requests, etc. Unfortunately the types of jobs the user submits and the rates at which he submits them are dependent on the response he gets and the time of day. Therefore the results

---

†Alternately a batch load could be run to use the 14 hours a day of idle time.

of this study would only be of significant value if the net result of the change is transparent to the user. If this were so, then probably the change could not be justified. Thus the element of dealing with human reactions makes the task almost impossible. Furthermore, the constant series of changes being made to MCGILL-RAX means that the characteristics of the system could likely change in such a way as to make the simulation results of dubious value. It is the feeling of the author that the time involved, possibly several man-months, could be better used in implementing the features rather than simulating their effect. The following case demonstrates the effect users can have on a time-sharing system.

One MCGILL-RAX installation, who was running on a SYSTEM/360 Model 40, planned to run on a Model 67 using a system called CP/67 and hoped for a performance similar to a Model 50. Due to a delay in the generation of their CP/67 System, it was decided to run the machine in Model 65 mode for a few weeks in the interim. During those few weeks the characteristics of the users' jobs changed so drastically as their usage expanded to use the faster system. When the switch was finally made to run RAX under CP/67, their users were not satisfied with the response as it was slower than what they had been getting with the machine in "65 mode". The decision was then made to run permanently without CP/67.

Several months later they had to get an additional Model 65 to handle the ever increasing load. From this experience it would appear that time-sharing users should never be given a facility that will have to be withdrawn or decreased later as they tend to fully utilize additional resources as they are added rather than grow into them at a linear pace.

For the above reasons no attempt should be made to attain full utilization of our system overnight. It is desirable however, to ensure that there is sufficient capacity for the near future to allow for the continued growth so that more users may be supported while maintaining suitable response time for all.

There are, of course, further enhancements which would enable the multiple user region to function better. For example, one could checkpoint the regions on a channel which is not used by the user regions in their I/O. Such a change might require additional I/O and control units.

PERFORMANCE TO THE USER

The implementation of multiple user regions might slow down the execution of a job by some 10%. However, due to the fact that the user requests are usually at least 50% I/O bound and that there is now a good chance that the

request will be started sooner, the net result should be an improved response.

SYSTEMS RELIABILITY AND AVAILABILITY

The multiple user region support would not increase the time to start or restart the system. It would not be inherently less reliable. Systems that use a paging scheme generally have a built-in reliability problem--that the system is programmed to abort when there is no more space in main storage to hold a critical page.

ADDITIONAL CAPABILITIES

The discussion so far has been concerned with the addition of another user region. However, with only a slight additional modification, a scheme could be implemented to dynamically allow the available user region of about 180K to be divided on a demand basis into several small regions or perhaps one large one. This availability would allow larger programs to run than are now possible and also permit smaller programs to get better service. The core fragmentation problem usually associated with this scheme would not be experienced due to the use of the relocation feature.

Additionally we could have subsystems which do not require disk I/O to the same extent that the usual RAX job does. These systems could communicate directly with the terminal, thus not using the disk as an intermediate device. These tasks would be capable of running while the disk channels were being fully utilized by the other regions or by the system for checkpointing. Candidates for this type of operation are "desk calculator" programs, or even a complete APL System.

In summary, the additional cost of adding multiple user regions to MCGILL-RAX at McGill is of minor consequence. The major consideration is whether the time spent making these modifications can be justified. The author thinks so.

CHAPTER 12

CONCLUSIONS

The SYSTEM/360 Model 67 was heralded as IBM's "time-sharing computer"[7]. The high cost of its additional main storage and channels together with the price of the relocation feature resulted in a number of time-sharing systems being designed to run without the use of relocation. The SYSTEM/370 line of computers now provides some form of relocation over a wide range of computing power. This availability, reinforced by its now nominal cost, makes the support of relocation of interest to a wide range of programming systems. Though the imminent announcement of the availability of full relocation on these machines has not been made at this time, this paper suggests that the relocation available through the use of the DOS/OS Compatibility Feature is sufficient for some purposes.

MCGILL-RAX is an ideal system to support these new features since it already has the capability of running on a range of computer models and it has been shown that it can be run relatively economically on these machines.

If the multiple user region support were not added to our system, we would not be able to continue to provide

sufficient service to our users. A faster machine could not be justified due to the relatively low CPU utilization at present. Having two identical systems would be most inconvenient to the systems and operations support staff.

It must therefore be concluded that the addition of multiple user regions to MCGILL-RAX is justified and that this modification should be made in the immediate future.

APPENDIX A

IBM EQUIPMENT RENTAL CHARGES

The rentals given in this section are based on IBM Canada's "Monthy Availability Charge" which includes maintenance.

All charges are computed without certain discounts which do not apply to all installations. Such discounts are for long term rentals of certain I/O units, educational allowance and duty and Federal Sales Tax exemptions.

CPU'S PRICED

System/360 Model	50	
	65	
	67	(Simplex)
System/370 Model	145	(without IFA)
	155	

Costs calculated based on the following I/O configuration which is typical for a large MCGILL-RAX installation.

2 Selector channels (or Equivalent)

1 Multiplexor channel

1 2703 Transmission Control Unit with 60 lines (30 2741 break, 30 TTY lines)

6 2319 Direct Access Storage Facilities (configured into two 9 drive (8 usable) units on separate channels)

2 2314 Control Units for the above

1 2540 Card Reader and Punch

1 1403 Printer

1 2821 Control Unit for the 2540 and 1403 devices

The rentals are summarized in the following graph and tables.

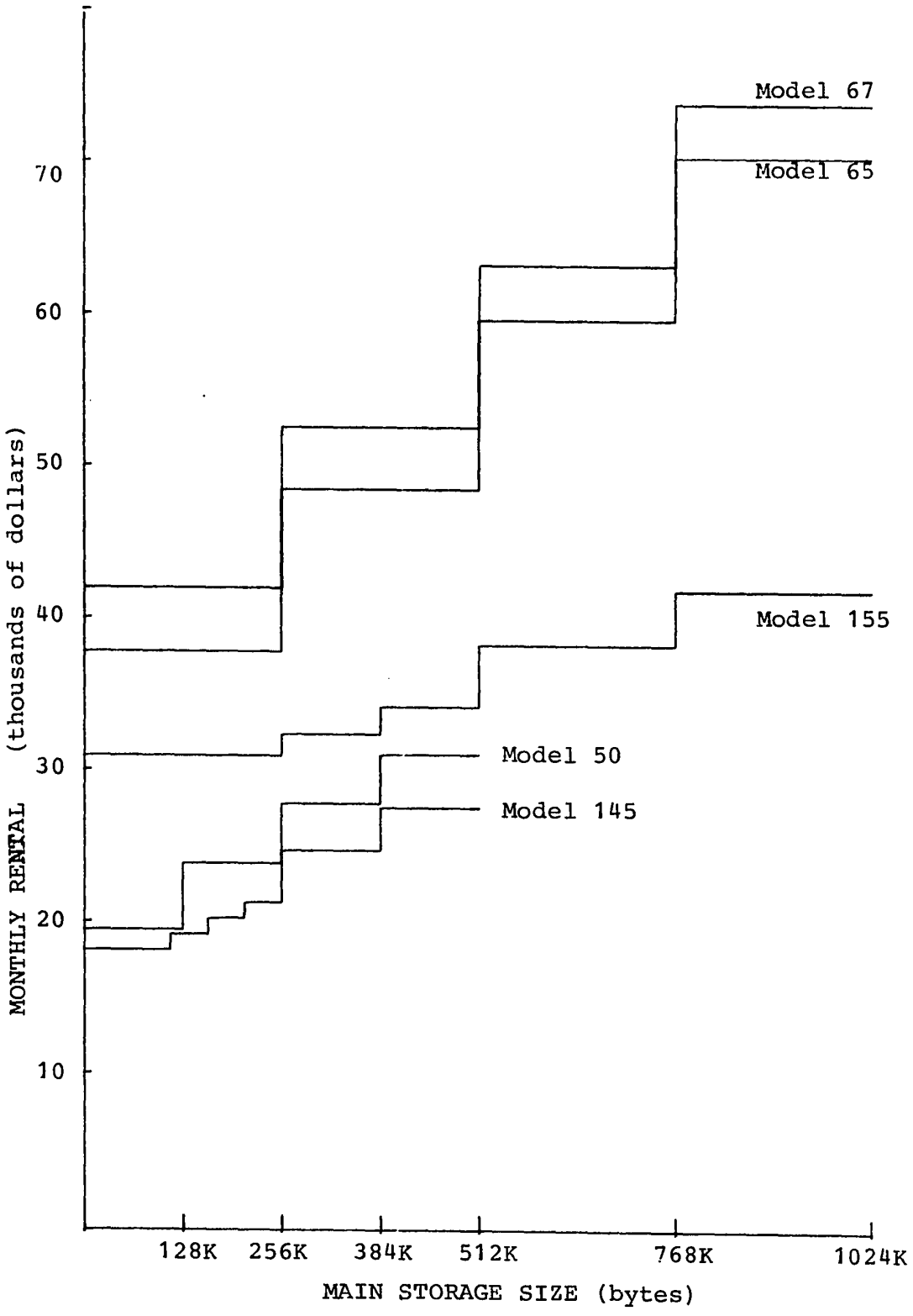


Figure 3 CONFIGURATION COSTS

ADDITIONAL COST OF RELOCATION FEATURE

	Block† Relocation	Page Relocation
System/360 Model 50	-	-
65	-	-
67	-	$3,495 to $4,130 ††
System/370 Model 145	$0	-
155	$295	-

† Such as with the DOS/OS COMPATIBILITY FEATURE

††Calculated from difference in price of Models 65 and 67 and depends slightly on the amount of main storage.

Table 3

COST OF ADDITIONAL SELECTOR CHANNELS

COST OF 3rd and 4th Selector Channel†

	3rd	4th
System/360 Model 50	$ 810	---
65	$1,040	$2,430
67	$1,040	$2,430
System/370 Model 145	$ 265	$ 265
155	$ 443	$ 413

†assuming that channels 1 and 2 are installed.

Table 4

COST OF ADDITIONAL MAIN STORAGE

	256K to 384K	384K to 512K
System/360 Model 50	$ 3,950	$ 3,220
65	$10,630†	$ 0
67	$10,630†	$ 0
System/370 Model 145	$ 3,424	$ 2,805
155	$ 1,800	$ 1,800

†Since 128K increments are not available, this reflects a 256k increase.

Table 5

APPENDIX B

Detailed measurements of the MCGILL-RAX System operating at McGill University were made during the week of July 5, 1971 (Monday through Friday). Specific emphasis was placed on recording the activity in the user region. Each request for the region was broken down into its CPU time, core requirements and its function. Supplemental measurements were made such as the number of active terminals and checkpoints. The results of these measurements are summarized in the following pages. Chapter 10 presents a description of the relevance of these graphs and tables.

Table 6 SYSTEM USAGE †

	number of check-points	number of region requests	user region usage (mins)		
			check-pointing	chargeable job time	
				wait	cpu
Monday	17,024	16,437	89	234	124
Tuesday	15,885	16,136	84	199	101
Wednesday	15,964	16,349	82	244	120
Thursday	18,639	18,087	97	242	128
Friday	20,486	17,193	107	211	101
Totals	87,998	84,202	459	1130	574
Average	17,600	16,840	92	226	115

†excluding Idle time

USER REGION USAGE

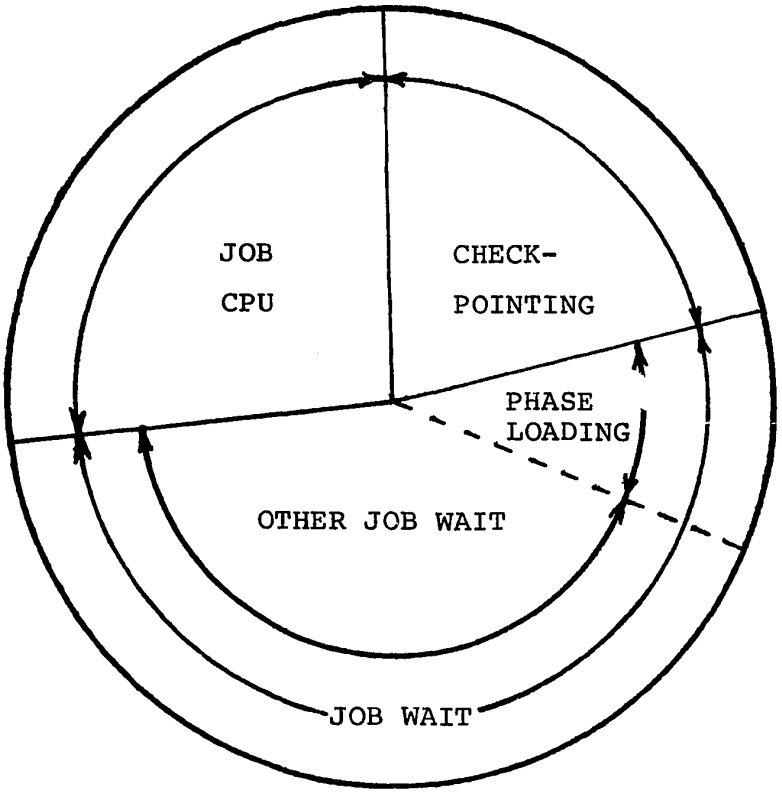


Figure 4

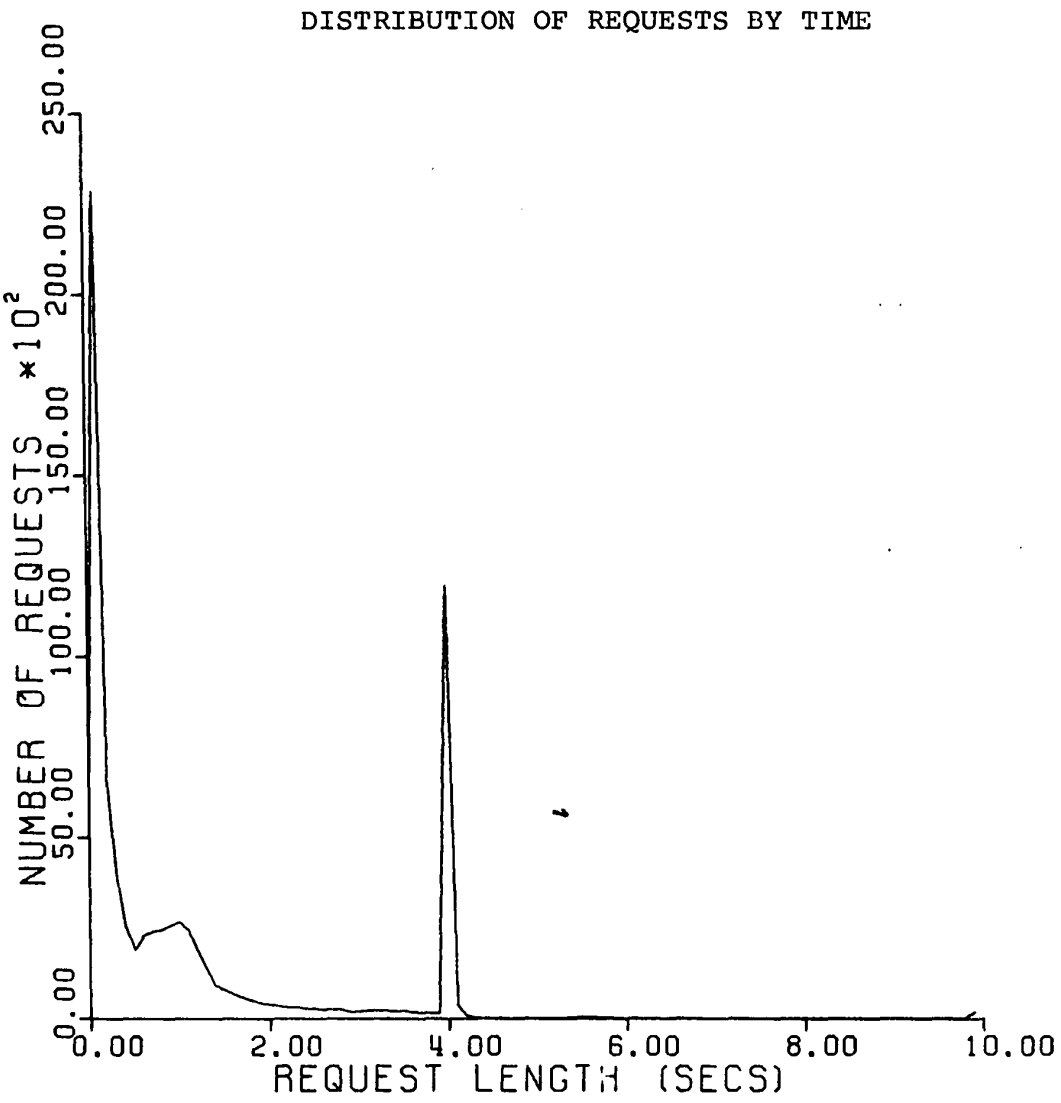

DISTRIBUTION OF REQUESTS BY TIME
NUMBER OF REQUESTS *10²
0.00
50.00
100.00
150.00
200.00
250.00
0.00
2.00
4.00
6.00
8.00
10.00
REQUEST LENGTH (SECS)

Figure 5

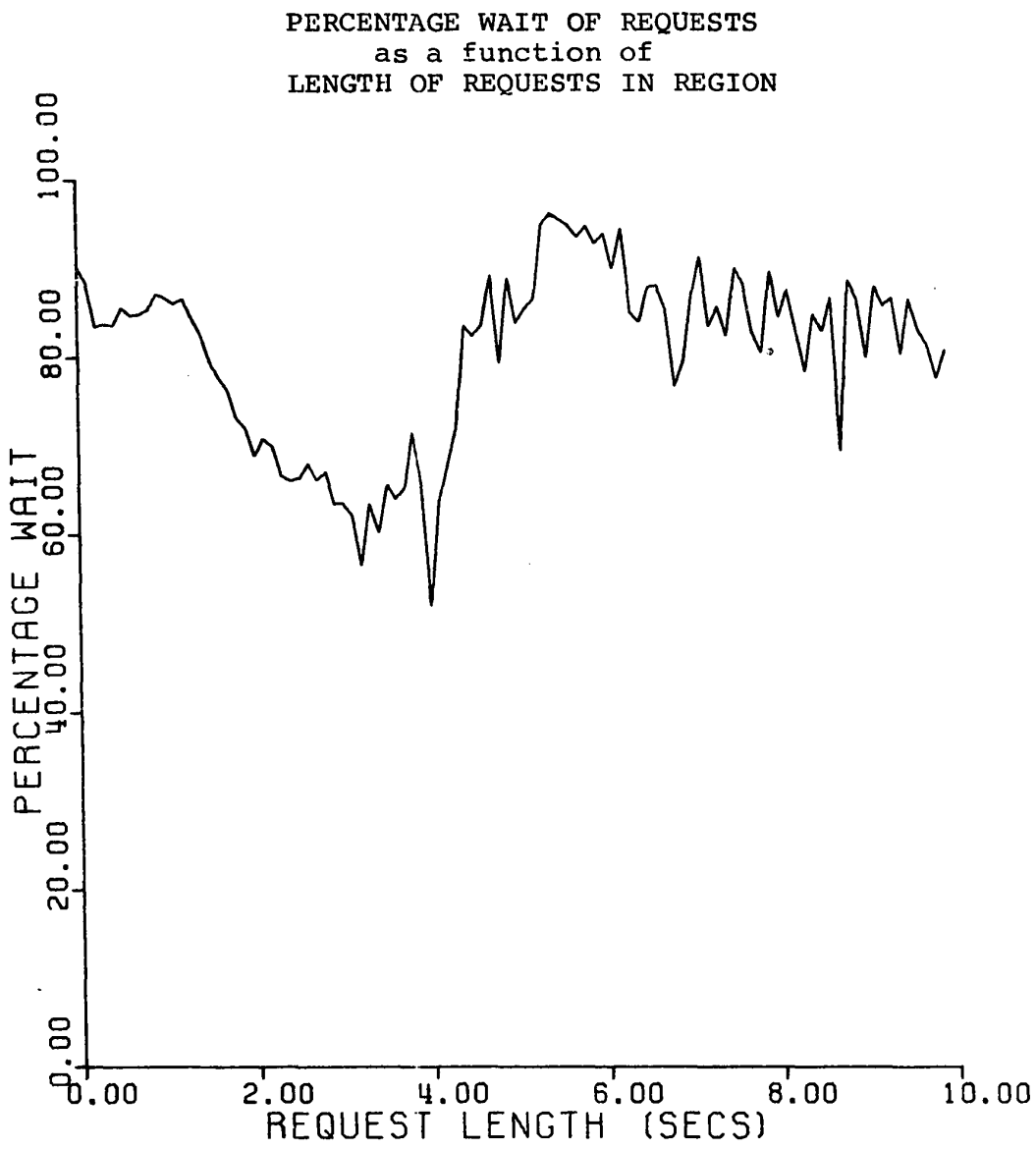

PERCENTAGE WAIT OF REQUESTS
as a function of
LENGTH OF REQUESTS IN REGION
PERCENTAGE WAIT
0.00
20.00
40.00
60.00
80.00
100.00
0.00
2.00
4.00
6.00
8.00
10.00
REQUEST LENGTH (SECS)

Figure 6

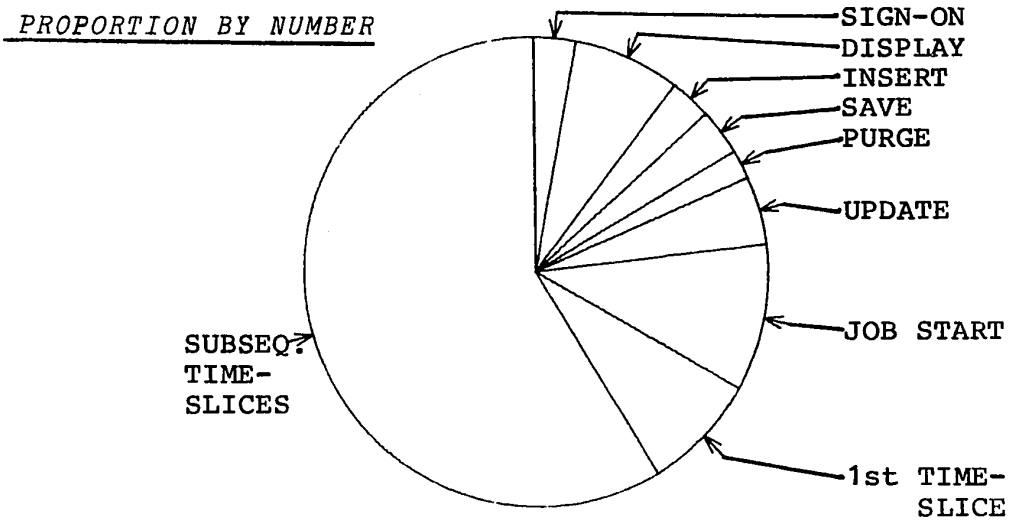
SUMMARY BY JOB TYPE

job type	approx size	number	average length	percentage wait time
sign-on	7.1K	2,500	0.20 secs	94%
display	4.6K	6,209	0.63	92
insert	4.6K	2,446	1.00	95
save	18.3K	2,660	1.12	89
purge	18.3K	1,683	0.60	94
update	19.5K	3,929	1.25	91
job start†	70.0K	8,545	2.09	86
1st time slice	10-110K	6,818	2.72	62
subseq. time slice	10-110K	49,392	0.86	55

† All programs written by users start here and may go to 1st time slice after BPS Compiler has scanned the job, or when FORTRAN G Compiler takes control. Subsequent time-slices are for those jobs that require more than one time-slice to complete.

Table 7

USER REGION USAGE BY JOB TYPE



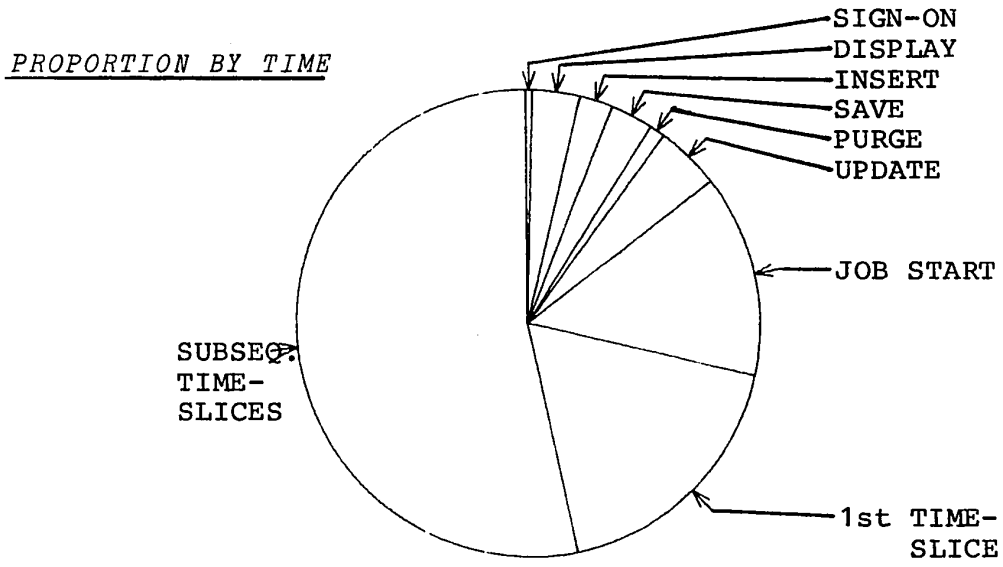


Figure 7

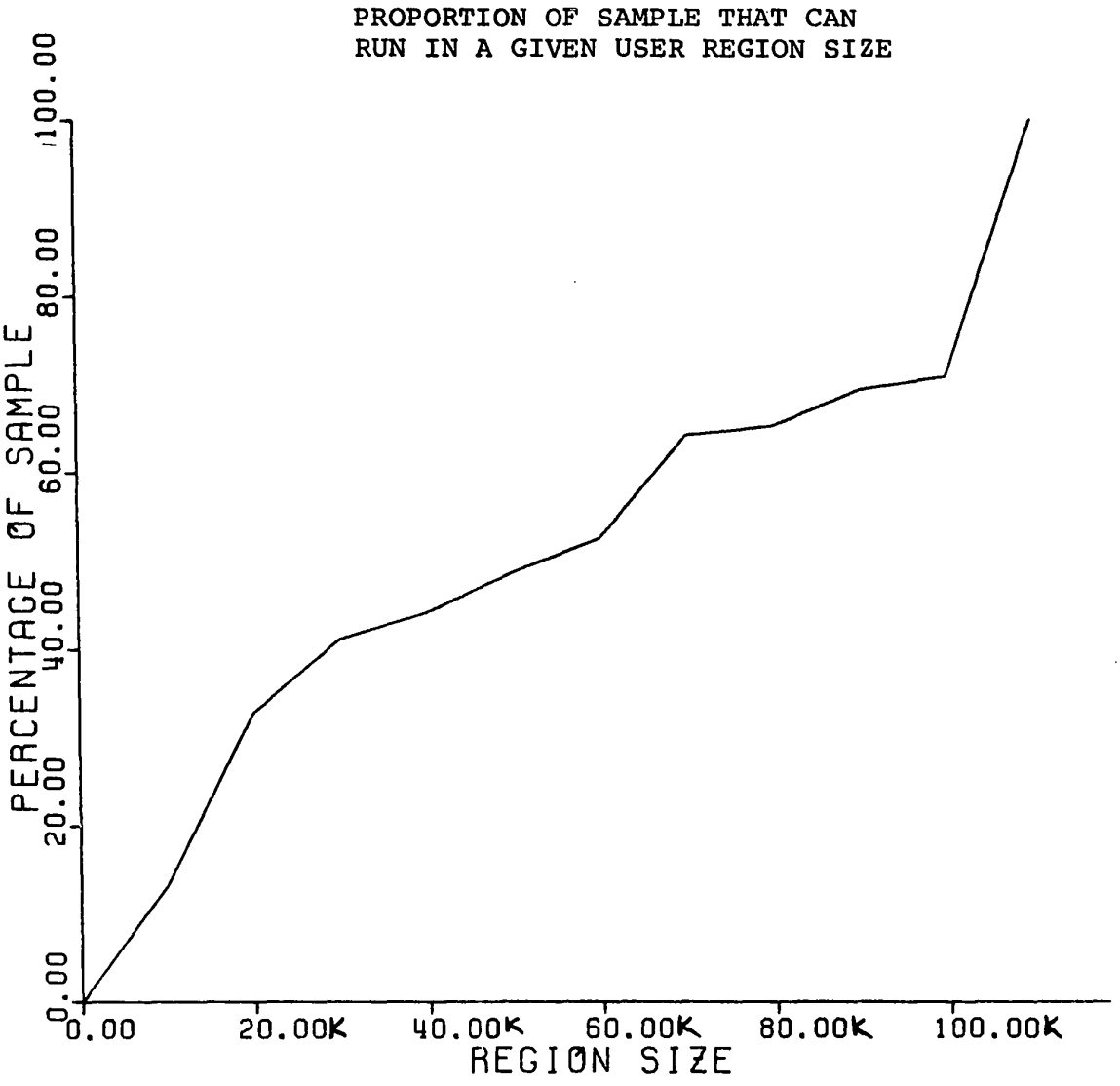

PROPORTION OF SAMPLE THAT CAN
RUN IN A GIVEN USER REGION SIZE
PERCENTAGE OF SAMPLE
0.00
20.00
40.00
60.00
80.00
100.00
0.00
20.00K
40.00K
60.00K
80.00K
100.00K
REGION SIZE

Figure 8

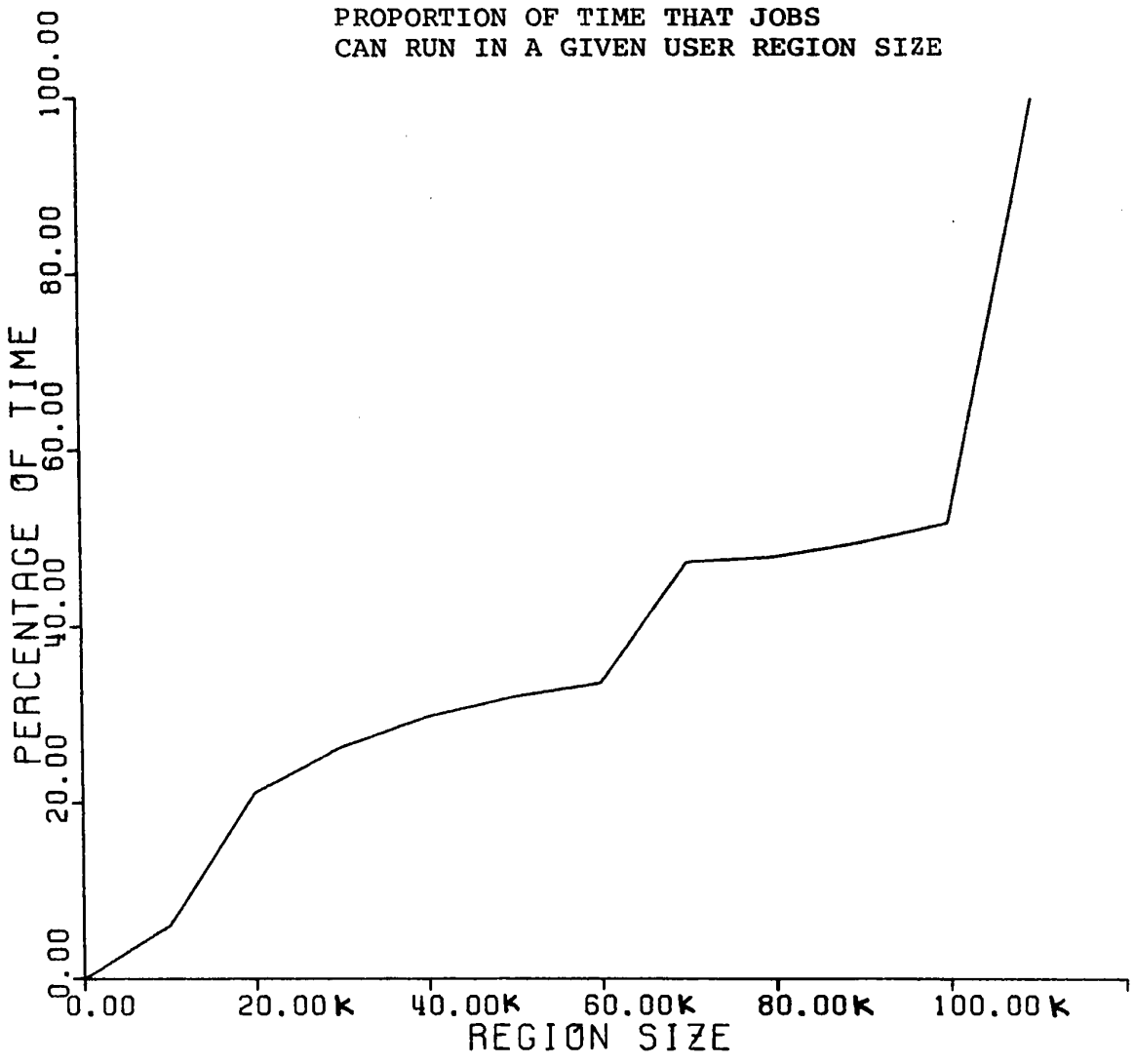
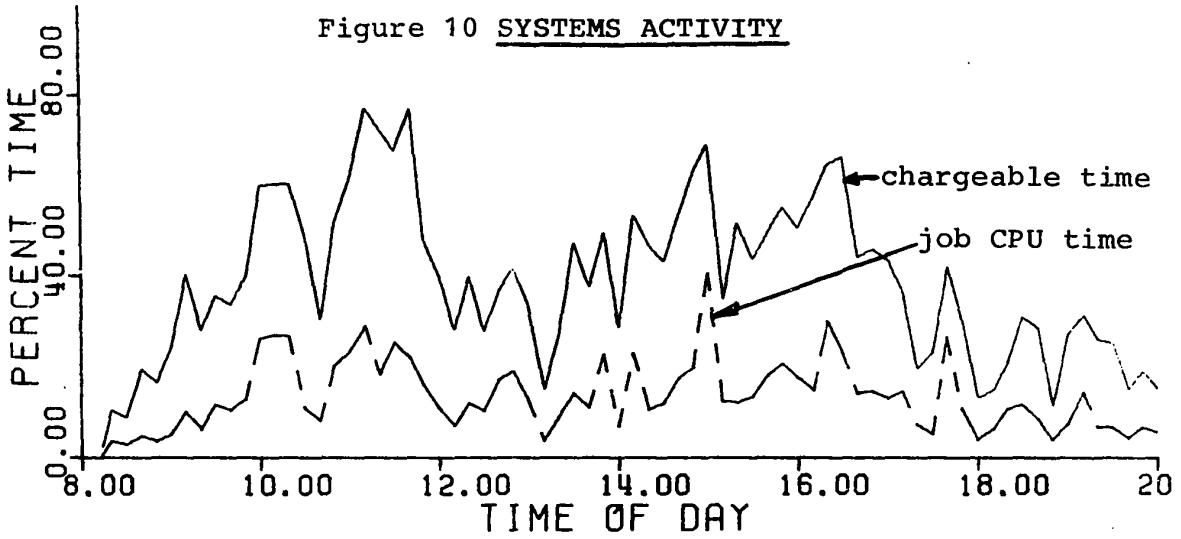
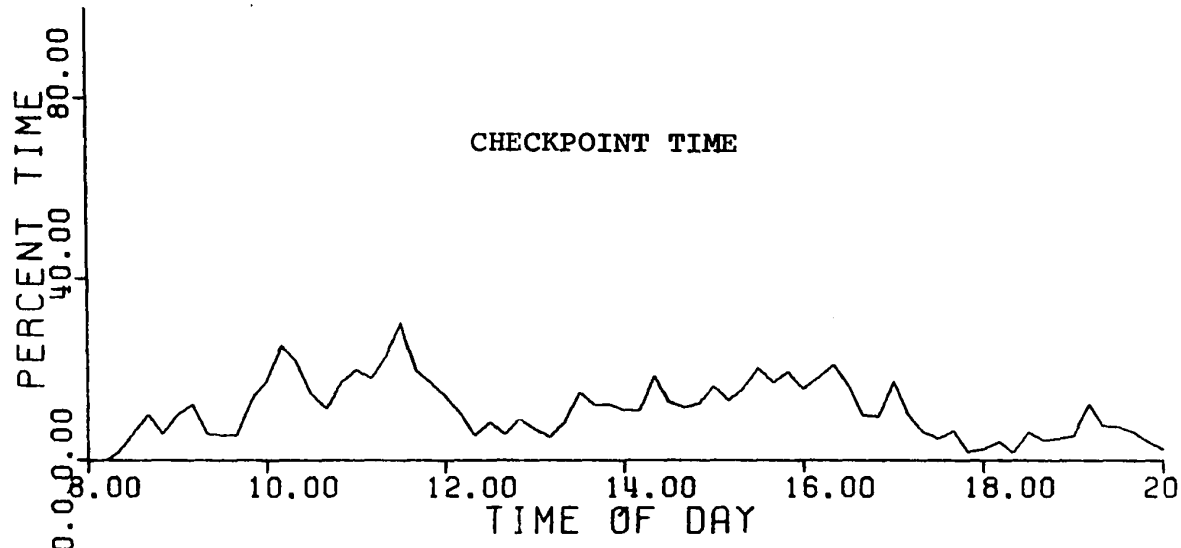
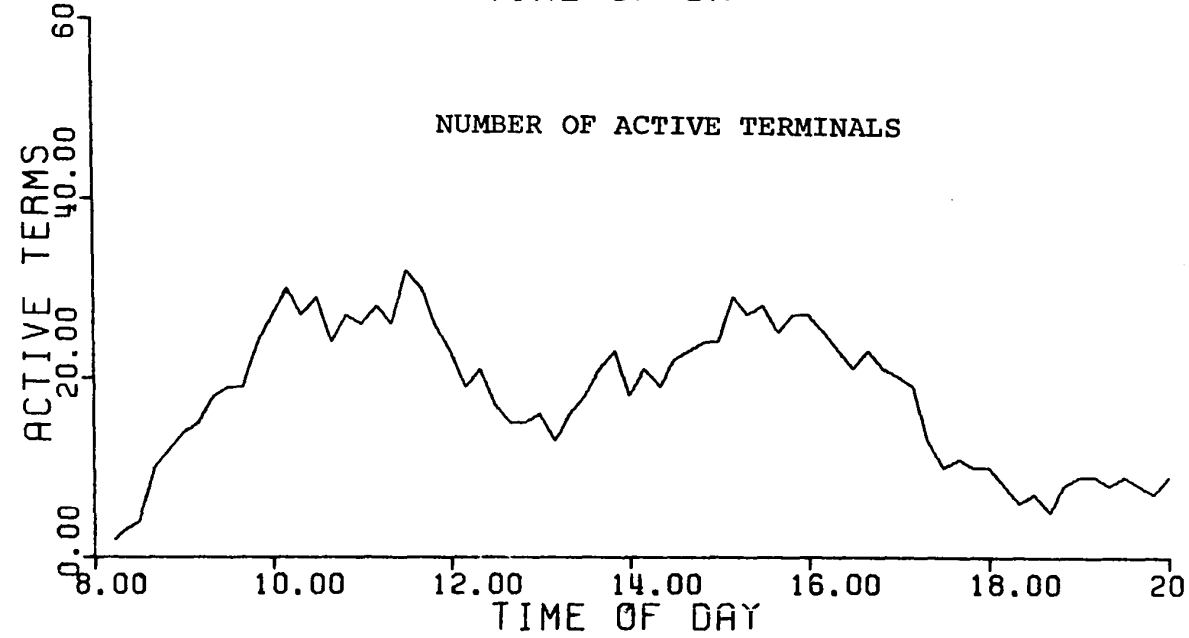

PROPORTION OF TIME THAT JOBS
CAN RUN IN A GIVEN USER REGION SIZE
PERCENTAGE OF TIME
0.00
20.00
40.00
60.00
80.00
100.00
0.00
20.00K
40.00K
60.00K
80.00K
100.00K
REGION SIZE

Figure 9

Figure 10 SYSTEMS ACTIVITY







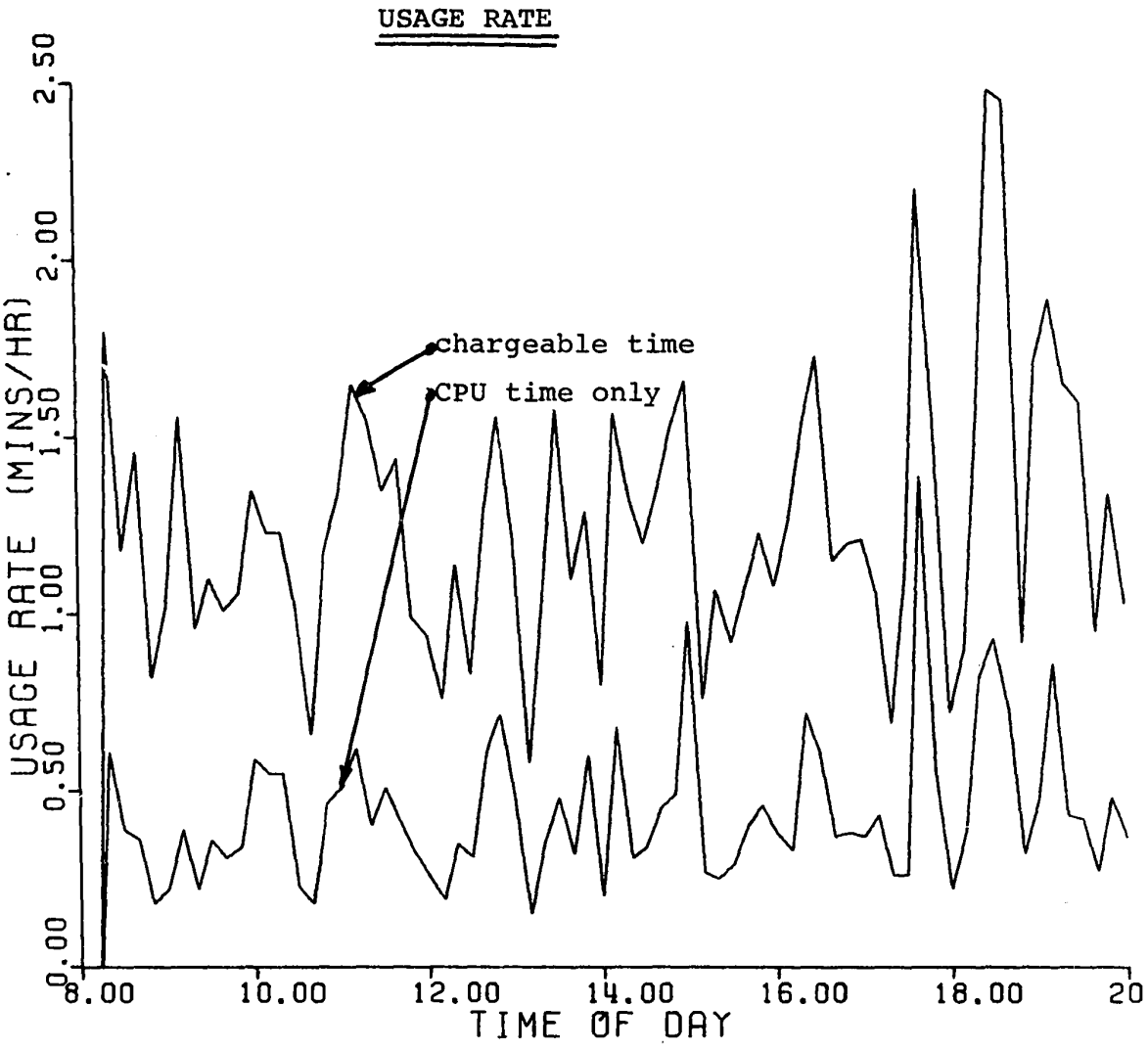

USAGE RATE
USAGE RATE (MINS/HR)
2.50
2.00
1.50
1.00
0.50
0.00
chargeable time
CPU time only
8.00
10.00
12.00
14.00
16.00
18.00
20
TIME OF DAY

Figure 11

BIBLIOGRAPHY

The scope of this paper does not permit an in-depth presentation of certain aspects of the subject matter. The following selected bibliography includes material of use by those who wish to investigate these subjects in greater detail. To facilitate its use in this manner, the works are grouped under major topic headings.

ACCOUNTING SYSTEM DESIGN

1) Hootman, Joseph T.
The Pricing Dilemma
Datamation
August 1969, pp. 61-66

2) Selwyn, Lee L.
Computer Resource Accounting in a Time Sharing Environment
AFIPS Conference Proceedings
1970 SJCC, Volume 36, pp. 119-130

3) Yourdon, Edward
Call/360 Costs
Datamation
November 1 1970, pp. 22-28

DOS/OS COMPATIBILITY FEATURE

4) Allred, Gary R.
System/370 Integrated Emulation under OS and DOS
AFIPS Conference Proceedings
1971 SJCC, Volume 38, pp. 163-168

5) IBM
DOS Emulator Logic (on IBM System/370 under OS)
IBM Systems Reference Library
Form Number GY26-3741

PAGING HARDWARE

6) Gibson, Charles T.
Time-Sharing in the System/360 Model 67
AFIPS Conference Proceedings
1966 SJCC, Volume 28, pp. 61-78

7) IBM
Model 67 Functional Characteristics
IBM Systems Reference Library
Form Number GA27-2719

PERFORMANCE EVALUATION

8) Drummond, M.E. Jr.
A Perspective on System Performance Evaluation
IBM Systems Journal
Volume 8, Number 4 (1969) pp.252-263

9) Yourdon, Edward
An Approach to Measuring a Time-Sharing System
Datamation
April 1969, pp. 124-126

PERFORMANCE EVALUATION USING HARDWARE MONITORS

10) Bonner, A.J.
Using System Monitor Output to Improve Performance
IBM Systems Journal
Volume 8, Number 4 (1969), pp. 290-298

11) Cockrum, J.S. and Crockett, E.D.
Interpreting the Results of a Hardware Systems Monitor
AFIPS Conference Proceedings
1971 SJCC, Volume 38, pp. 23-38

PSYCHOLOGICAL REACTIONS OF TIME-SHARING USERS

12) Boehm, B.W., Seven, M.J. and Watson, R.A.
Interactive Problem-Solving-- An Experimental Study of "Lockout Effects"
AFIPS Conference Proceedings
1971 SJCC, Volume 38, pp. 205-210

13) Miller, Robert B.
Response Time in Man-Computer Conversational Transactions
IBM Systems Development Division
Poughkeepsie, New York
Paper Number TR 00.1660-1

RAX HISTORY

14) IBM
Lockheed Pioneers Remote Computing
IBM Computing Report
April 1966, Volume 1, Number 2, pp. 16-18

RAX INTERNAL ORGANIZATION

15) IBM
Remote Access Computing System (RAX) System Manual
IBM Systems Reference Library
Form Number GY20-0101

RAX OVERALL DESCRIPTION

16) IBM
RAX Program Description Manual
IBM Systems Reference Library
Form Number GH20-0354

SCHEDULERS; DESIGN OF TIME-SHARING

17) Doherty, Walter J.
Scheduling TSS/360 for Responsiveness
AFIPS Conference Proceedings
1970 FJCC, Volume 37, pp. 97-111

18) Hellerman, H.
Some Principles of Time-Sharing Scheduler Strategies
IBM Systems Journal
Volume 8, Number 2 (1969), pp. 94-117

19) Wilkes, Maurice V.
A Model for Core Space Allocation in a Time-Sharing System
AFIPS Conference Proceedings
1969 SJCC Volume 34, pp. 265-271

SIMULATION OF OPERATING SYSTEMS

20) Chang, W.
A Queuing Model for a Simple Case of Time Sharing
IBM Systems Journal
Volume 5, Number 2 (1969), pp. 115-125

21) Kleinrock, Leonard
A Continuum of Time-Sharing Scheduling Algorithms
AFIPS Conference Proceedings
1970 SJCC, Volume 36, pp. 453-458

22)Seaman, P.H.
The Role of Digital Simulation
IBM Systems Journal
Volume 5, Number 3 (1966), pp. 175-189

23)Seaman, P.H. and Soucy, R.C.
Simulating Operating Systems
IBM Systems Journal
Volume 8, Number 4 (1969), pp. 264-279

TIME-SHARING TECHNIQUES

24)Cluff, Milton H. and Thompson, David
Principles of Time-Sharing--
Class Notes
IBM World Trade Systems Centre
Bulletin Number 39
February 1969

Virtual Machines

25)Meyer, R.A. and Seawright, L.H.
A Virtual Machine Time-Sharing System
IBM Systems Journal
Volume 9, Number 3 (1970), pp. 199-218