COVER SHEET FOR TECHNICAL MEMORANDUM

TITLE Performance Simulation and Measurement of a Virtual Memory Multi-programming System for a Small Computer

MM 71 - 1383 - 3

CASE CHARGED – 39395

DATE – January 20, 1971

FILING CASES – 39395-61

AUTHOR – H. Lycklama

FILING SUBJECTS – Small Computers
Simulation
Operating System
Memory Management
Virtual Memory
Multiprogramming

ABSTRACT

The performance of an operating system for a small computer in a virtual memory multi-programming environment is described in terms of through-put and response. Both direct measurements and simulation runs have been utilized to obtain quantitative results on the operating system performance. The findings were used as aids in the evaluation of and as guidelines in making refinements to the operating system. Memory management strategy and user file organization were found to be the two major factors affecting the through-put and response of the operating system. The simulation model has been extended to study the effects of increasing memory size, increasing disk speed and in increasing memory speed for a small computer system. The results were then incorporated into an economic model of the system pointing out the cost effectiveness of the improvements.

SEE REVERSE SIDE FOR DISTRIBUTION LIST

TABLE OF CONTENTS

		Page
1.0	Introduction	1
2.0	System Description	2
3.0	Measurement Techniques	4
3.1	A Note on Simulation	7
4.0	Initial Operating System	8
4.1	Simulation and Measurement Results for Operating System 1 (O.S. #1)	12
5.0	Modified Operating System (O.S. #2)	16
5.1	Further Improvements to O.S. #2	18
6.0	Some Two Level Memory Considerations	21
6.1	Discussion of Critical System Parameters	23
7.0	Further Simulation Results on O.S. #2	26
7.1	System Features as a Function of Disk Speed	27
7.2	System Features as a Function of Memory Size	30
7.3	System Features as a Function of Memory Speed	32
7.4	Comparison of the Various System Configuration Simulation Results	33
8.0	An Economic Model of the System	34
9.0	Conclusions	38

Bell Laboratories

subject: Performance Simulation and Measurement of a Virtual Memory Multi-programming System for a Small Computer
Case 39395-61

date: January 21, 19'

from: H. Lycklama
MM71-1383-3

MEMORANDUM FOR FILE

1.0 INTRODUCTION

This study was initiated in order to understand and measure quantitatively the response and through-put of a small time shared computer system. The operating system utilizes the concepts of demand paging and virtual memory. It is difficult to assess the merits of a particular operating system without having precise measurements by which the effectiveness of certain strategies can be compared with those employed by other operating systems. Memory management strategy is an important part of the overall operating system design. Also user file organization has a marked effect on the average access time to secondary storage. Any model of the operating system must describe these two activities in detail in order to give close agreement with measured results on the system and in order to predict the effects of changes in implementation of these two activities. In particular one would like to know the trade-offs involved in increasing memory size, increasing disk speed and in increasing memory speed for a small computer system.

This study describes the results of measurements on a virtual memory multi-programming operating system designed for an 8K, 16-bit word computer with 376,000 words of secondary storage.(1) A simulation model was developed to gain insight into the operating system and hence point out the bottlenecks in the system. The findings were then used as aids in the evaluatio of and in making refinements to the operating system. The disk allocation scheme as described in (1) has been altered considerab to avoid some of the bottlenecks pointed out by the simulation model. This resulted in a three-fold increase in through-put. Through-put here is defined as the percentage of total elapsed time during which the CPU is executing a user's program. The new disk allocation scheme was built into the simulation model. This model pointed out some more bottlenecks. Improvements in core management and file organization resulted in a further gain of about 30% in through-put. CPU utilization, system response, disk activity and system overhead are some of the parameters which wer monitored in the simulation and actual system measurements.

Close agreements of the simulation model results with the actual system measurements was achieved over a wide range of system parameters. The model was further extended to study the results of increasing memory size, increasing disk rotational speed and decreasing memory cycle time. These results were then incorporated into an economic model of the system pointing out the cost effectiveness of the improvements.

2.0 SYSTEM DESCRIPTION

The basic system hardware configuration consists of a Honeywell DDP-516 computer with 8192 words of 16-bit, 0.96 μsec core memory. Secondary storage consists of a fixed head, 64-track disk (~376,000 words) operating at 1800 RPM and connected to the computer via a fast DMA (Direct Memory Access) channel. Most of the I/Ø devices are connected to the computer via a serial I/Ø bus(2) as shown in Figure 1. These consist of four model 103 Dataphone data sets, a PDP 8/I computer and a special memory interface. A control teletype and a 201 Dataphone data set are interfaced separately to the computer I/Ø bus.

Software for the system consists of 4000 words of syste code (the operating system) and many segmented user programs, which are brought into the 4000 word area of user core on demand. Each user has a thread-save block to store all his temporary data which allows all code to be re-entrant. Programs are broken up into segments of various lengths. Data is broken up into segment of 64 words each to keep the core requirements of each user down to a reasonable amount. The system code handles virtual memory addressing, core management, I/Ø processing, thread changing (changing control from one user to another) and the allocation of all system resources.

The main user programs currently supported by the system include:

Text Editor - for creating and updating files.
Assembler - to assemble program segments.
TRAC - a character string manipulating language.
FSNAP - an interpretive calculating language.
RØPAK - a relocatable octal debugging package.

and several programs for communicating with the GE-635 computer. The operating characteristics of all of these programs depend very much on the file organization and core management strategies implemented in the operating system. If the file which the user is accessing is small, the response time to the user's typed-in request for service is negligible regardless of the number of users on the system. However, as the length of a user's file increases, the time to access a file will increase as some part of his file may be off on secondary storage. Thus it is important that his files be organized for minimal access time with or without other users on the system. One of the purposes of the present simulation study is to gain insight into the factors affecting the file access time.

3.0 <u>MEASUREMENT TECHNIQUES</u>

The evaluation parameters which one employs to characterize a multi-programming system can be divided into two categories. First of all there are the global system capacity parameters including through-put, response time and cost per operation. Then there are the local parameters which are much more dependent on the individual hardware and software configurations. These include:

(i) average secondary storage access time

(ii) average data transfer rate

(iii) average instruction execution time

(iv) average executive system processing time

(v) average user processing time

(vi) effective busy time of processor

and (vii) hardware utilization - processor
- memory
- secondary storage.

There are various techniques which can be used to measure these parameters depending on the accuracy with which one wishes to measure the parameters and whether one is more interested in global than in local parameters of the system. In this study we are concerned with studying the system in some detail and hence are interested in measuring both types of parameters.

Characteristic of all such computer system studies, one must measure the parameters of interest under a given load on the system. One technique often employed is the stimulus approach in which a certain job is entered into a mix of running programs and the response time noted. However, this method suffers from the fact that the results are not reproducible and the effects of system changes are difficult to detect. The use of a benchmark task is more rewarding. Such a task can be run under a new version of the operating system and the measured parameters compared directly with those obtained on an earlier version of the operating system. The benchmark task also has the advantage over using a normal mixture of jobs in that the fluctuations in the nature of the load do not wipe out the effect to be measured. The use of benchmark tasks is emphasized in the present study.

Three major techniques were employed to measure the system parameters. Hardware was installed on the present system in the form of megacycle clocks and external meters to monitor various system parameters. The meters were used to monitor average CPU utilization, user time, system time and idle time by indicating the corresponding clock counting rates. Other meter were used to indicate such parameters as the average disk word transfer rate and the number of users logged into the system.

Software was used in conjunction with the megacycle clocks to turn them on or off at the appropriate times in the operating system to measure time spent in various sections of system and user programs. A clock monitor program keeps track of the total time spent in various programs and types out the accumulated results on command. The monitor program itself only takes up about 128 words of core memory and produces no appreciabl interference with the operation of the system.

The use of hardware and software techniques allows one to measure the parameters of the present operating system but does not allow one to predict the results of changing some features in the operating system. For this reason a simulation model was built to allow for extrapolation beyond the present system configuration. A benchmark task was used to measure the system response and through-put as a function of various changes in the operating system. The particular task chosen was one which exercises many of the tasks of the text editor, namely, it creates a file of 30,000 characters, rewinds the file and then deletes the file.

3.1 A Note on Simulation

High level simulation languages have been developed to model computer systems, e.g., GPSS(3), SIMSCRIPT(4), SIMULA(5), et However, the execution times of simulation model programs written in these higher level languages tend to be long and their versatility leaves something to be desired. For example, we found that to run a typical model in SIMSCRIPT required three times as long as the same model written in FORTRAN. Other systems analysts have come to the same conclusions regarding higher level simulation languages. The universality and versatility of FORTRAN make it a good language upon which to build a simulation mo

The system model is based on a list of discrete events which occur when the system changes state, e.g., from user to system, system to idle, etc. Entities which are created, e.g., users, change their attributes at each change of state. The system parameters being monitored require updating at each change of state. In the model the angular position of the disk is defined as a linear function of the simulated running time of the model. Such close attention to detail was found to be necessary in order to closely describe and model the system. Benchmark tasks are used to exercise the system both in the simulation model and on the DDP-516 computer system to permit comparison of the simulated and the measured results. It is important to generate tasks which are long enough to collect meaningful statistics.

4.0 INITIAL OPERATING SYSTEM

A fairly detailed description of the operating system will be presented below in order to introduce the parameters of interest in the model. A more detailed discussion can be found in (1). Memory management is one of the most important aspects of the operating system. Program and data segments are brought into core on demand. When core memory becomes fully occupied and space is required for another segment, the segment table is searched for data blocks or program segments which are not presently in use and can be pushed out. One of the important variables involved in this core request is the minimum core request, MINCR, i.e., the minimum amount of core space freed up when core is full. Freeing up a certain minimum amount of core reduces the overhead for the next request for core space but increases the probability of pushing out a segment which may be required soon. This is a trade-off which will be investigated. Freeing up core involves marking as holes those data blocks or program segments which have not been updated since being brought into core and putting on a disk queue those segments which have been altered. The segments on the disk queue are written out on disk under control of the interrupt routine. When the sum of the holes is great enough to satisfy the space requirements for the segment, core is shifted down over the holes and the segment is read in from disk or a new data segment created, as the case may be, at the top of the currently used core.

Program segments typically vary in size from 20_8 to 1400_8 words with a majority of the segments being less than 200_8 words long. User files are organized as linked data segments each 100_8 words long. Each segment contains 118 characters (2 per word). The data segment size, DTSGSZ, is another system variable which can be optimized for the system. However it is desirable to keep segments small to allow for a better mix of users. The value of this parameter will be discussed in some detail in a subsequent section.

About 376,000 words of disk storage are available for user files and programs. Disk is organized as shown in Figure 2. The disk name table contains the names of all the program segments on disk with a cross-reference pointer to the corresponding ID table entry. The disk ID table contains the disk address of each program or data segment on disk. All unused ID's are so indicated. The ID table entry also contains the size of the segment. The important aspect of this disk allocation scheme is that to read a segment from disk, two disk accesses are required,

(1) read ID table entry to get segment size and disk address

(2) read segment down from disk.

To write a new segment on disk requires five disk accesses,

(1) read ID table entry (as two entries are made per 8-word sector)

(2) write changed ID table entry on disk

(3) check disk image of ID table entry

(4) write segment in allocated disk space

(5) check disk image of segment.

Another aspect of this scheme is the number of disk accesses required to allocate and de-allocate ID's. Initially all available ID's are put on a chained queue which is organized on a last in first out (LIFO) basis. Allocating an ID requires one disk access to read the next ID from the end of the chain. To de-allocate an ID requires three disk accesses similar to the first three steps in writing a segment on the disk. This process takes a minimum of two disk revolutions as the same disk sector must be accessed three times. During this time all other ID table references by other disk queue entries are blocked to prevent multi-programming collisions and also since all ID table entries are read into a fixed location in the system.

A disk queue is maintained containing all of the disk access entries which require service. Each time this queue is serviced, the entry with the least latency is given priority. There is of course a certain minimum disk delay necessary, DSKDEL, before an entry can be serviced. This is the time required to search the queue for the entry with the least latency after the disk angular position has been read. DSKDEL is another variable which can be optimized for the system. If the entry with the least latency cannot be serviced, it is blocked and the next best entry is given service. Of course the greater the number of blocked entries, the less efficient the disk queue server will be. When all entries are blocked, they are all immediately unblocked and the next best entry is serviced provided it is not again blocked. One reason for a block occurring is that a disk - user core transfer cannot be initiated while a core shift is taking place. Another reason is that only one ID table entry can be examined at a time. Thus for a write ID table entry operation, a disk table flag is set up after the entry has been read from disk so that another ID table entry cannot be written over the previous entry while it is being set up for the subsequent write and compare operations. Thus the disk table flag is up for at least two complete disk revolutions.

4.1 Simulation and Measurement Results for Operating System 1 (O.S. #1)

A benchmark task consisting of creating, rewinding and deleting a file of 30,000 characters was run on the system. A simulation model was also run for comparison. The results are summarized in Table 1. The column headings are as explained below:

TIME -	total elapsed time taken to perform the benchmark task (in seconds).
TIDL -	time during which the CPU is idle.
TUSR -	time during which the CPU is active in the user's program.
TSYS -	time during which the CPU is active in system programs, e.g., virtual addressing, space finding, servicing disk queue, etc.
TMDUT -	time during which the disk is being utilized including setup, access and transfer time.
TMDTF -	time during which the ID disk table entry is in use and another ID table entry cannot be read from disk.
TMDSF -	time during which a disk sensitive task is in progress and core shifts are inhibited.
TMCSF -	time during which core shifts are in progress and a disk sensitive task cannot be initiated.
TMDDP -	time during which some disk queue entries are blocked (these entries are re-enabled only when no others are available for service).
TMIDF -	time during which an ID is being allocated to or de-allocated by a user (during this time other users are prevented from doing the same).
NCSFT -	number of core shifts which occurred during the monitored time.

NCTHD - number of thread changes which occurred during the monitored time (this is the number of times which the CPU was reassigned to another user).

NDSKI - number of disk accesses which occurred during the monitored time.

The close agreement of the simulated results with the measured results for the various number of users on the system indicates that all of the gross features of the operating system have been included in the simulation model. It is quite evident from the results that too much time is being spent by the CPU in the idle state. Even by dividing the total work up between five users, only a 25% saving in total time is achieved. Basically the reason for the large percentage of idle time is that too many disk accesses are required for one operation, e.g., five accesses to write a segment on disk. The cause of the poor mix of jobs is that some disk accesses inhibit the initiation of other similar types of accesses until the results of the first disk access have been altered, rewritten on disk and compared at least two complete revolutions of the disk later.

More specifically the bottlenecks in the operating system are as follows. The time during which the disk table entry is in use, TMDTF, is rather long. During this time other disk queue entries requiring the use of the disk table entry (eight words in core memory) are blocked. One possible solution to this bottleneck would be to allow for multiple disk table entries to be read into core. A simulation run showed that this would reduce the time for the benchmark task from 79 seconds to 55 seconds. However, the modification in system program required to implement this change would contribute to the system overhead and still would not eliminate the basic reason for excessive idle time, i.e., too many disk accesses. The best possible solution would undoubtedly be provided by avoiding the use of an ID table on disk altogether and having some form of a disk table permanently in core memory. This would eliminate the need for many of the disk accesses.

The time during which an ID is being allocated or de-allocated is also a large fraction of the total benchmark task time. During this time all other users are prohibited from allocating or de-allocating ID's and also from using the disk table entry in core. This again causes some disk queue entries to be blocked and prevents the disk queue from being serviced very efficiently. A solution to this bottleneck would be to read or write a block of say 64 ID's at a time and use these 64 for allocation and de-allocation until one runs off either end of the block of ID's. This would greatly reduce the number of disk accesses required for allocating and de-allocating ID's. However, again this is only a "half-way" solution and does not completely eliminate the possibility of having blocked queue entries. The results of the two major bottlenecks discussed above, namely the limitation to the use of one disk table entry at a time and the multiple disk accesses required to handle allocation and de-allocation of ID's, can be seen in TMDDP, the time during which some disk queue entries are blocked.

Another major bottleneck is presented by the core management strategy. As discussed previously, when core space is required for a segment and no space is available at the top of the currently used core, the segment table is searched for segments which can be thrown out. Those which must be rewritten on disk are put on the disk queue and serviced according to the least latency principle. Meanwhile the segments which remain in core are shifted down to make room at the top for the new segment. However core cannot be shifted while a disk sensitive task is in progress. This turns out to be at least 25% of the time. Similarly a disk sensitive task cannot be initiated while core is being shifted down. These conflicts tie up memory unnecessarily and hence diminish the through-put of the system. A good solution for this bottleneck (which was later implemented) is to shift core only when absolutely necessary, i.e., put the new segment in an existing hole in the core map. However, this solution would only be effective if the probability of finding a hole of the right size were high. Such is not the case in operating system 1 as there are no restrictions on the size of the segments.

Some improvements were implemented in the existing operating system. A simulation run demonstrated that the response time for the benchmark task previously described with one user could be improved from 79 seconds to 70 seconds by randomizing the allocation of the ID's for the data segments. This can be achieved by putting the available ID's on the chain of ID's in a random order. An actual test run on the system gave close agreement with the simulation results. Another improvement was achieved by modifying the disk queue servicing strategy. Now all blocked entries are re-enabled immediately before looking for the next best disk queue entry to service. This reduced the response time for the same benchmark task by another four seconds.

However, the improvements made are only minor ones. Only up to a 15% saving in response time can be made. The fact remains that there is too much idle time as too many disk accesses are required to perform a basic operation. The disk queue cannot be serviced very efficiently as one queue entry may require several disk accesses before another entry can be serviced. Also the bottleneck caused by the inhibition of core shifts while a disk sensitive task is in progress remains.

5.0 <u>MODIFIED OPERATING SYSTEM (O.S. #2)</u>

It was obvious that a radical change in the disk organization was necessary to improve the response of the system. Therefore, the disk storage area was reorganized as shown in Figure 3. The number of possible segment sizes has been reduced to eight, all multiples of 100_8 words ranging from 100_8 to 1400_8 words. Now there is no longer a disk ID table kept up on disk, but a disk map is kept in core using one bit to indicate whether or not each block of 64 words is allocated. The ID of each program and data segment contains the actual disk address at which the segment is physically stored on disk and the segment size. Thus to read a segment from disk requires only one disk access (compared to two previously) and to write a segment require only two disk accesses (compared to five previously), one to write the segment and another one to perform the disk compare operation. To allocate or de-allocate an ID now requires no disk accesses compared to one and three, respectively under the old operating system. Now there are no longer any conflicts in servicing a disk queue entry as it is impossible to have a blocked entry. The process of allocation or de-allocation of an ID no longer inhibits the allocation or de-allocation of another ID. Hence the disk queue can be serviced very efficiently.

Some measurement and simulation results for O.S. #2 are shown in Table 2. These can be directly compared to those for O.S. #1 as listed in Table 1. It can be seen that the response time has been reduced by at least a factor of 3 over the initial version of the operating system for the corresponding number of users. This can be attributed almost directly to the fewer disk accesses required to perform the benchmark task. The time spent by the CPU in system routines has also been cut by a factor of two from about 6 seconds to 3 seconds. The user time of course remained constant at about 5 seconds. As evidenced by the fewer number of core shifts which occurred under O.S. #2 the core management strategy has been changed in this operating system. Segments brought down from disk or created for the user are put in existing holes of equal size if possible to minimize the number of core shifts required. The probability of finding a hole of the proper size is now relatively high as all segments are a multiple of 64 words in size.

One of the most important advantages of this second operating system is the fact that users now mix much better. Dividing the benchmark task up between five users cuts the total response time by a factor of about 2 compared to a factor of about 1.3 in the initial operating system. This can be attributed to the fact that disk queue entries can now be serviced effectivel according to the least latency principle.

5.1 Further Improvements to O.S. #2

Two more improvements were implemented in operating system #2 on the basis of encouraging results projected by simulation runs. By randomizing the order in which the ID's are allocated, a 30% improvement in response time was projected for 1 user executing the 30,000 character file benchmark task. This was borne out by a run of the same benchmark task on the system. A random allocation of ID's seems appropriate for this system as the files must be accessed in both directions, forwards and backwards.

The second improvement made was to make the core management strategy more efficient. In the first operating system a core shift was required whenever space was needed for a segment and core was fully occupied. In the second operating system a core shift was required only when an exact match could not be found between an existing hole size and the requested segment size. A further improvement was now implemented in which a segment was placed in consecutive holes if possible to fulfill the space requirement. A set of simulation runs were performed to study the effects of the change in core management strategy. The simulation model was run for 60 seconds with three users on the system repeating the benchmark task outlined earlier. Typical runs indicated that the number of core shifts during a 60 second period would be reduced from 250 to 40 with the change in strategy from core shift only to core shift only if no exact match found and from 40 to 1 with the change in strategy from core shift only if no exact match found to core shift only if not enough consecutive hole space exists. Corresponding to these figures the time spent by the CPU in the user state typically increased from 19 sec. to 24 sec. and from 24 sec. to 27 sec. respectively. This amounts to a dramatic reduction in the number of core shifts required to manage core and an almost 50% increase in throughput for the system. Needless to say this improvement in core management strategy was implemented in the operating system.

Table 3 lists the results of simulations and measurements for the improved version of O.S. #2 for the same benchmark tasks as used to obtain the numbers in Tables 1 and 2. The statistical data portrayed in Tables 1, 2 and 3 bring out some of the important features of the operating systems under study. In all of the benchmark tasks monitored for the various versions of the operating system, the user time is of course constant but the system overhead time for the second operating system is down by a factor of two from that for the first operating system. For the improved version of O.S. #2 the system time is decreased somewhat more. Most important of all, the idle time for the improved version of O.S. #2 has been reduced by a further 40% over the factor of four improvement of O.S. #2 over O.S. #1. Dividing the benchmark task up between five users has further decreased the response time from 12.7 seconds to 10.3 seconds for the improved version of O.S. #2. This is a measure of how well the users "mix" or share resources in the multi-programming environment. The improvement in core allocation strategy has all but eliminated the need for core shifts. This is much more important in O.S. #2 than in O.S. #1 as in O.S. #2 almost all disk transfers are disk sensitive tasks and the disk is being utilized a large fraction of the time. Thus there is a much smaller probability of having t inhibit disk activity due to a core shift in progress or vice vers of having to inhibit a core shift due to a disk sensitive task in progress. The disk utilization time has been reduced considerably by the implementation of random ID allocation. This indicates a smaller average disk access time and more effective servicing of the disk queue entries. The total time to process a task or a number of users is very dependent on the efficiency of the disk handler routine.

The results for the average disk access time and the average word transfer rate for the various operating systems studied are summarized in Figures 4 and 5 respectively. The average disk access time decreases quite rapidly as a function of the number of concurrent users for O.S. #2 and even more so for the improved version of O.S. #2 but is almost a constant for O.S. #1. Similarly the word transfer rate to and from disk under operating system #2 increases quite satisfactorily as the number of simultaneous users increases. However, for O.S. #1 the word transfer rate is almost insensitive to the number of concurrent users. Clearly O.S. #2 and even more so the improved version of O.S. #2 handle the disk queue much more effectively than O.S. #1. Users "mix" or share system resources much more efficiently under operating system #2.

6.0 SOME TWO-LEVEL MEMORY CONSIDERATIONS

A two-level memory hierarchy has certain limitations which must be taken into account in order to make efficient use of secondary storage. The access time of a two-level memory structure is given by:

$$t = f(1+\alpha r)$$

where f = access time for the fast memory,

r = ratio of access time of the slow memory to that of the fast one, and

α = fraction of all accesses that must be made to the slow memory.

For the Honeywell DDP-516 system under discussion, $f \simeq 1$ μsec and $r \simeq 10$ msec/1 μsec = 10,000 for a moderate number of users on the system. It is difficult to get α to approach 1/10,000 for a system with only 4K words of user space. Thus the average access time for this memory structure cannot approach that of the fast memory.

The utilization factor of a disk can be defined as:

$$u = \frac{\bar{t}}{\bar{t} + d}$$

where d = dead-time due to rotational latency, and

$\bar{t}$ = data transfer time.

For an average number of requests on the disk queue $\bar{N}$ for a disk with a time τ for one rotation of the disk,

$$d = \frac{\tau}{\bar{N} + 1} .$$

To transfer a block of data (64 words) from disk requires,

$$\bar{t} = \frac{64}{5880} \tau \simeq \frac{\tau}{90} .$$

For $\overline{N} = 9$, the disk utilization factor is in the order of 10%. The maximum word transfer rate from disk, i.e., the bandwidth of disk is 5880 x 30 = 176,400 words per second. This compares with a core bandwidth of about 1 million words per second. Even with five active users on the system the disk utilization factor is only about $\frac{9,200}{176,400}$, i.e., slightly greater than 5%. This corresponds to an average number of requests on the disk queue of about six.

6.1 Discussion of Critical System Parameters

There are certain system parameters which can be optimized for an efficient operating system. The disk delay (DSKDEL), i.e., the time to determine which is the next disk activity to initiate for a minimum access time, is one such parameter. Both simulation runs and actual runs on the system indicate that the best value for this parameter is about 20 sectors (8 words per sector). As shown in Figure 6 the through-put of the system as measured by the user time forms a broad maximum in the region of DSKDEL = 20. This corresponds to just slightly more than the average processor time taken to search the disk queue. A simulation run assuming a hardware queuer and server demonstrated that the through-put of the system could be increased by about 8%.

A second parameter which can be optimized is MINCR, the minimum core request when core space is required for a segment. Then when say 128 words of core are required for a segment, a minimum of MINCR words of core are freed up if possible. The optimum value of this parameter was found to be about 1024 words for good response and through-put in the types of tasks simulated in this study. However, the optimum value of this parameter is almost certainly a function of the memory size, the types of tasks and their mix.

A third parameter is the size of the data segments themselves. A size of 64 words was chosen for this system for a number of reasons. This size is compatible with a multi-programming system which may have many segments in core simultaneously providing for ease of core management. In fact all program segments are multiples of 64 to simplify core management. The amount of data which a user wishes to access at any one time is rarely more than 64 words. Actually taking into account the time to transfer a block from disk and the time taken by the processor to process the block, a size of 64 words strikes a happy medium as will be shown in the following discussion.

The time to transfer a block, size n, from disk is given by:

$$t_b = t_a + nt_w$$

where t_a = time to access start of block, and

t_w = time to transfer one word.

The time for the processor to process a block, size n, is given by:

$$t_p = t_s + nt_k$$

where t_s = time to initiate block transmission, and

t_k = time to process one word.

One can then define the ratio of these as:

$$u = \frac{t_p}{t_b} = \frac{t_s + nt_k}{t_a + nt_w}$$

for which $u > 1$ the system is processor bound,

$u = 1$ the system is balanced, and

$u < 1$ the system is I/Ø bound.

From this the desired block size for a balanced system is given by,

$$n = \frac{t_a - t_s}{t_k - t_w} .$$

In a typical text editor application in which one is copying a file, the times are approximately,

$$t_a = 10 \text{ msec.}$$
$$t_s = 1 \text{ msec.}$$
$$t_k = 300 \text{ }\mu\text{sec.}$$
$$t_w = 5 \text{ }\mu\text{sec.}$$

in an environment of 3 to 4 users. Then the optimum block size is $n \simeq 30$ words. However, in another typical text editor operation in which one deletes the file from which the new file has been copied, the time to process one word is about 30 μsec and then the optimum block size is $n \sim 360$ words. Assuming that one is accessing files much more frequently than one is deleting them, a data block size of 64 words is a reasonable compromise between the two extremes and still compatible with the core size limitations.

7.0 FURTHER SIMULATION RESULTS ON O.S. #2

All other simulation results and measurements were obtained with the parameters fixed at the values discussed in the previous section. The operating system is the improved version of O.S. #2 with random allocation of ID's and almost complete elimination of core shifts implemented in the system. The benchmark task used was the same as outlined previously with the exception that the file size simulated was 15,000 characters regardless of the number of users on the system. The benchmark tasks were run repeatedly for a total time of 60 seconds as a function of the number of users from one to eight. The operating system features studied include:

(1) response - time to complete one benchmark task

(2) through-put - as measured by the user time

(3) average disk access time

(4) average word transfer rate

(5) processor utilization - idle
- system
- user

all as a function of the number of users on the system and as a function of disk speed, memory size and memory speed.

7.1 System Features as a Function of Disk Speed

The basic system configuration consists of 8K of 1 μsec memory and an 1800 RPM, 376,000 word disk. The results of the simulation runs and of the measurements on the basic configuration are summarized in Figure 7. The simulated and measured results are in close agreement in the region from 1 to 3 users where the results can be compared. It is noted that the idle time approaches a minimum for eight users such that more efficient use cannot be made of the processor by adding more users. The overhead time, i.e. the system time increases quite rapidly as a function of the number of users. This can be attributed to the larger number of thread changes taking place and the greater amount of core memory management required. User time or through-put increases almost linearly as a functio of the number of users, indicating an effective sharing of the system resources. The disk utilization time stays almost constant at slightly less than 75% of the total time simulated.

The next system configuration simulated has the same features as the basic configuration but now has a higher speed disk, i.e., 3600 RPM, but still with 376,000 words. The simulation run results are summarized in Figure 8. There are no measured results to compare with but the success of the simulation model in predicting performance for the basic system gives one confidence that the results are indeed valid. The disk activity has been modelled in detail so that the disk's angular position is a function of time and so that the disk delay necessary to determine the next best queue entry to service, is included. As would be expected the through-put for one user is greater than with the 1800 RPM disk, but certainly not twice as great. This is partly due to the fact that the system is no longer as well balanced as it was designed to be for the 1800 RPM disk and also partly due to the fact that the system is not disk-bound. The disk utilization time is even less now as the average disk access time has been shortened. The through-put certainly increases as the number of users increases but not as quickly as for the basic system. This can be attributed to the fact that the 3600 RPM disk system becomes processor-bound much more quickly than the 1800 RPM disk system. Beyond about 6 users the idle time actually increases slightly while through-put levels off. At first glance this may seem surprising but there is an explanation for this behavior. As the number of users increases, memory becomes heavily occupied, i.e., more and more segments are tied down in core and cannot be pushed out when more space is required. The number of entries put on the disk queue for service when core space is required is not as great as would have been possible if there were fewer users and hence less memory was tied down. The queue cannot be serviced as efficiently now with the result that the average disk access time actually increases thereby reducing the total through-put. This effect is not as noticeable with the 1800 RPM disk as the disk delay time is a smaller fraction of the disk rotation time. As the disk delay time becomes a larger fraction of the disk rotation time, the chance of a queue entry's disk address falling in this area becomes greater. An entry whose disk address falls in this critical area cannot be serviced until a full disk revolution later. A few simulation runs with a smaller disk delay time indicated that the idle time was a very sensitive function of the disk delay time. In fact the idle time curve did flatten out rather than turn up when the disk delay time was decreased towards the idealistic value of zero.

The same system configuration was again simulated, this time with a 376,000 word disk running at 900 RPM. The results are portrayed in Figure 9. As might be expected the system is highly disk-bound with the disk utilization time approaching 100% of the total simulated time. The through-put for one user is much less than for the higher speed disks simulated previously. Two more users are easily handled by the system but beyond three users the through-put is not increased significantly. The problem again becomes one of memory restrictions.

For the various speed disk systems simulated, the optimum number of users would appear to be about five, from the point of view of through-put. The important thing to realize here is that through-put does not increase in direct proportion to disk speed. The percentage increase in through-put from the 900 RPM disk system to the 1800 RPM disk system is much greater than from the 1800 RPM disk system to the 3600 RPM disk system. This can be attributed to the fact that as the disk speed is increased the system becomes more and more processor bound.

7.2 System Features as a Function of Memory Size

Another parameter which can be varied in the simulation model and in the actual system to some extent also, is the size of memory available to the user. Some preliminary measurements and simulations were carried out to demonstrate in detail the effects of varying the memory size. A benchmark task dealing with a file of 6000 characters was run for a total of 60 seconds for various values of available memory size with only one user on the system. The results of the measurements and of the simulation runs are compared in Figure 10. The reason for choosing a file length of 6000 characters is that a file of this length would fit completely in a memory size of ~9500 words. This represents an extreme case and hence a useful data point. In the region of 8K or fewer words where the simulated and measured results can be compared, there is close agreement. The simulated results have been carried beyond the point where the complete file fits in memory. Of course at this point idle time and disk utilization time drop to essentially zero. The through-put increases at a faster rate than linear. Interestingly enough the system time does not increase appreciably. This can be attributed to a simplified core management task for a larger core size. Figure 11 shows the corresponding response time as a function of memory size. The response time, which here is defined as the total time to complete one benchmark task, decreases approximately inversely as the memory size available to the user. Of course when the complete file can be contained in memory, response is at a minimum of 1.15 seconds. With an 8K memory size the response time is only about a factor of two greater. This seems like a reasonable response for a virtual memory system.

To further study the effects of increasing the size of memory, it was decided to simulate the system with the three different disk speeds as before but now with an extra 4K of memory added on to effectively double the core space available to the user. Figure 12 shows the results of the simulation runs for the 1800 RPM disk with 12K of core. It is obvious that the through-put is increased for a larger core size but the percentage increase over the through-put for the 8K system is greater for a small number of users than for a larger number of users. The corresponding results for the 3600 RPM and 900 RPM disk systems with 12K of core are shown in Figures 13 and 14. The same basic conclusions can be drawn for these systems as for the 1800 RPM disk system. Note that for all systems, the idle time does decrease as a function of the number of users and does not increase for eight users above what it is for five users as it did in the case of 8K of core and the 3600 RPM disk. The improvement is a result of less severe core restrictions. Again it can be noted here that the increase in through-put of the system is not directly proportional to the amount of core space available to the users.

7.3 System Features as a Function of Memory Speed

A third system design parameter which one can vary is the speed of memory. Here the effects of doubling the speed of memory and hence cutting the cycle time of the processor in half are studied via simulation. The results of a series of simulation runs for an 8K computer supported by an 1800 RPM disk are shown in Figure 15. As expected this system is highly disk-bound as it is not properly balanced for a 0.5 μsec memory. Through-put increases linearly with the number of users and shows little sign of slackening off with up to 8 users. With slower memory, 5 users was about the optimum number of users.

The results as shown in Figure 15 were given further credibility by simulating the same system, this time with a 3600 RPM disk. As expected the response time was one-half of that for the basic system with an 1800 RPM disk and a 1 μsec cycle time. Essentially the time scale has been cut by a factor of two. Processor speed has a very marked effect on response time of the system.

7.4 Comparison of the Various System Configuration Simulation Results

A better perspective of the results of changes in disk speed, memory size and memory speed can be gained from a study of Figures 16 and 17. Figure 16 displays the average disk access time as a function of the number of users for all of the variations in disk speed, memory size and memory speed studied. The increase in disk rotation speed certainly decreases the average disk access time but not in direct proportion to the increase. Increasing memory size has only a small effect on the average disk access time, but in all cases it does have a marked effect when there are many users on the system. There is no upturn in average disk access time with eight users as there was with the 8K memory. The reduction in disk access time as a function of the number of users is most dramatic in the case of a faster memory cycle time, decreasing from eleven to about five milliseconds for eight users.

Figure 17 shows the response time for the benchmark task in question as a function of the number of users for the various values of disk speed, memory size and memory speed simulated. Increasing core size for the 900 RPM disk system has a marked effect on response as now a larger fraction of the user's file is held in core and less disk accessing is required. Even for the 1800 RPM disk system the decrease in response time is appreciable for a larger memory size. However for the 3600 RPM disk system the effect of increasing core size is less noticeable. Doubling the speed of memory has the most pronounced effect on response time. The slope of the curve on Figure 17 corresponding to a higher speed memory is much less than that of the other curves for the 1 μsec memory. Of the three major parameters varied in the system design, memory speed is the most important one.

8.0 AN ECONOMIC MODEL OF THE SYSTEM

The simulation results have given a quantitative measure of the effects of changes in the hardware configuration and changes in software strategy, e.g., ID allocation and memory management. However, one would like to give a dollar value to the cost effectiveness of these changes. Hence we propose a simple model of a multi-programming virtual memory system (as outlined in this paper) which can support up to eight users who are using the system for data retrieval and file updating and thus accessing files in a manner similar to that simulated in the benchmark tasks. We give a dollar value to each piece of hardware in the system based on a per unit cost for a quantity of about fifty such systems. Then for each hardware addition or removal the total cost of the system is adjusted accordingly. The estimated cost for each piece of hardware is listed in Table 4 and is assumed to represent today's (1970) market prices fairly closely. Thus for each system configuration, it would cost about $2000 to add communication hardware to support an additional user terminal. The cost of a basic system supporting only one terminal would be $26,000.

There are two ways of measuring the cost effectiveness of the changes in the hardware configuration. One method is to look at the variation in the product of the total cost per user times the response time of the system. The longer the response time that each user sees the more expensive it becomes for a user to execute a certain task. A second method is to look at the variation in the total cost per unit of through-put. Here the user time per given time interval obtained in the simulation runs is used as a measure of useful system through-put. Actually it turns out that the resulting curves are of similar shape and the corresponding relative values on the curves are almost identical. This is to be expected as response is almost inversely proportional to through-put per user.

Figures 18 and 19 display the results obtained for this study making the assumptions outlined above for eight different system configurations. Figure 18 gives the relative cost ratio obtained as a product of the total system cost per user and the response time of the system to the benchmark task. Figure 19 gives the relative total system cost per unit of through-put. An analysis of the curves leads to some surprising results. Consider the effect of the increase in disk speed. For each different disk speed simulated the increased disk speed pays for itself in terms of improved performance. However for more than five users the system becomes more expensive to operate, i.e., cost performance deteriorates. For more than 6 or 7 users the 3600 RPM disk does not pay for itself when compared to the 1800 RPM disk. Increasing memory size from 8K to 12K for each different disk speed improves the cost performance when there is only one active user on the system. For three or more users on the system the extra 4K of core does not pay for itself, i.e., the response may improve with more memory but not enough to offset the extra core costs. The explanation for this is that the memory management algorithm used does not know which segments to throw out when making space. Thus to perform the benchmark task the whole file must be paraded into core three times. Having a larger core only means a slightly increased probability that the data segments which are to be accessed will be in core memory when needed. Hence extra core is really only helpful when file sizes are small and one can be guaranteed that most of the file will remain in core while it is being accessed. If the greater part of the file does not remain in core during accessing, extra core will not be beneficial unless it is large enough to contain the greater part of all user files being accessed.

Higher speed memory is seen to be very cost effective. The doubling in memory speed actually doubles the processing power of the CPU. For one or two users the system is more expensive than the other configurations modelled. However, if the system is meant to support three or more users it is more cost effective than any of the other configurations. In fact as the number of users on the system increases the relative cost of the system decreases whereas for the other configurations there is a definite turning point at about five users beyond which the relative cost increases. Even a faster disk improves the cost effectiveness as the system is disk-bound with a 0.5 μsec memory. However, for eight users the cost effectiveness is not much improved over that for five users. With the larger number of users the system again becomes core limited and processor-bound as with the basic system configuration.

For all configurations simulated here except for the 0.5 μsec memory system the optimum number of users seems to be about five. For fast memory the more users the less the relative cost for the system is (up to about eight users). This economic model of the system brings to light, not the factors which improve the response and through-put of the system, but rather the factors which improve response and through-put enough to produce an economical saving. It must be noted that the results obtained here assume that the users are busy 100% of the time, i.e., no "think time" has been included in the system model. No factor has been included to take into consideration the cost of human time which would make poor response expensive The number of users referred to on the x-axes of all the relevant figures is actually the number which are busy 100% of the time while other users may be in an idle state. Thus the number of users is really to be regarded as a statistical average of the number of users waiting for a response from the system.

9.0 CONCLUSIONS

The simulation and measurement results have suggested some further improvements in the operating system. For instance, the rewind operation in the TEXT editor has been simplified by storing the starting ID of the current file being accessed. Thus when one wishes to rewind the file, one merely links up to the starting ID. Previously one would read in each segment of the file using the back pointing ID to get to each preceding data block until the beginning of the file was reached. This was primarily a disk-bound operation. Another improvement which is presently being implemented is the breaking up of some large program segments into smaller ones so as to ease the core management task by providing for an easier flow of smaller program blocks into existing holes in core. This also has the advantage of diminishing the amount of memory occupied by each user at any one time and thus provides for better interaction and less severe core restrictions.

The implementation of meters on the computer system portraying the allocation of the system resources has proven very useful in providing insight into the operating system. The meters give a dynamic picture of the use of the system resources as various types and mixtures of programs are being run. Hence it is an easy matter to determine whether the system is disk-bound or processor-bound. The meters show up the system bottlenecks instantly whereas software measurements and simulation results usually require some further analysis to determine the bottlenecks.

This paper has given quantitative measurements of a particular operating system in a multi-programming, virtual memory environment. These measurements and the simulation results have given insight into the operating system and provided a good understanding of the factors which affect the through-put and response of such a system. The simulation of the basic model has given a good reference on which to base further simulations of modified versions of both the operating system and the hardware configuration of the system. The economic model proposed has given an insight into which possible improvements to the system would prove to be cost effective. Increasing processor speed is by far the most economical improvement provided the system is being utilized by more than two users for a good fraction of the time.

It is hoped that the results presented in this paper will prove useful in the design of operating systems for small computer systems, especially those which depend heavily on core management and disk I/O activity. The optimization of these two activities has been found to be of extreme importance in maximizing the through-put of the system described here.

ACKNOWLEDGMENT

The author wishes to thank H. S. McDonald for his aid in formulating an economic model of the computer system.

H. Lycklama

MH-1383-HL-RP H. LYCKLAMA

Att.
References
Tables 1-4
List of Figures
Figures 1-19

REFERENCES

(1) C. Christensen, A. D. Hause. "A Multi-programming Virtual Memory System for a Small Computer", AFIPS - Conference Proceedings, Vol.36, 1970, pp 683-690.

(2) D. R. Weller. "A Loop Communication System for I/O to a Small Multi-User Computer", MM70-1384-1.

(3) H. Herscovitch, T. H. Schneider. "GPSS III - An Expanded General Purpose Simulator," IBM Systems Journal, Vol. 4, No. 3, pp 174-183, 1965.

(4) H. M. Markowitz, B. Hausner, H. W. Karr. "SIMSCRIPT: A Simulation Programming Language", Prentice-Hall, 1963.

(5) O. Dahl, K. Nygaard, "SIMULA - an ALGOL-based Simulation Language", Comm. of ACM, Vol 9, No. 9, pp 671-678, September 1966.

Users	TIME	TIDL	TUSR	TSYS	TMDUT	TMDTF	TMDSF	TMCSF	TMDDP	TMIDF	NCSFT	NCTHD	NDSKI
Simulated 1†	78.7	67.1	5.2	6.4	72.6	35.9	21.3	1.4	20.2	44.6	90	-	3350
Measured 1†	79	68	5	6	71	34	21	1	22	45	85	-	3400
Simulated 2‡	71.8	59.8	5.2	6.8	69.8	37.3	23.3	2.3	34	48	120	560	3400
Measured 2‡	72.5	60.5	5	7	68	35	22	2	40	53	105	540	3400
Simulated 5*	59.1	44.5	5.3	9.3	57.9	36.0	16.3	3.1	35	41.1	170	650	3350

† 30,000 character file

‡ 15,000 character file each

* 6,000 character file each

TABLE 1

Users	TIME	TIDL	TUSR	TSYS	TMDUT	NCSFT	NCTHD	NDSKI
1†	24	16	5	3	19	6	-	900
2‡	16	8	5	3	14	7	460	1020
5*	12.7	4.2	5.2	3.3	9.6	5	385	960

† 30,000 character file

‡ 15,000 character file each

* 6,000 character file each (simulated)

TABLE 2

Users	TIME	TIDL	TUSR	TSYS	TMDUT	NCSFT	NCTHD	NDSKI
Simulated 1†	17.2	9.7	5.2	2.3	11.9	0	-	979
Measured 1†	17.6	10.0	5.3	2.3	12.3	0	-	982
Simulated 2‡	14.6	6.5	5.2	2.9	11.5	1	440	990
Measured 2‡	14.7	6.4	5.3	3.0	12.0	1	458	1020
Simulated 5*	10.5	2.3	5.1	3.1	7.1	2	493	1012

† 30,000 character file

‡ 15,000 character file each

* 6,000 character file each

TABLE 3

Price List of Computer System Components

Processor	$ 4,000
8K Memory (1 µsec, 16-bit words)	10,000
12K Memory (1 µsec, 16-bit words)	14,000
8K Memory (.5 µsec, 16-bit words)	20,000
Disk (32 track, ½M words, 900 RPM)	6,000
Disk (64 track, ½M words, 1800 RPM)	8,000
Disk (128 track, ½M words, 3600 RPM)	12,000
Terminal	2,000
General Interface	2,000

TABLE 4

LIST OF FIGURES

Fig. 1. Hardware configuration

Fig. 2. Disk Map (O.S. #1)

Fig. 3. Disk Map (O.S. #2)

Fig. 4. Average disk access times for the various operating systems

Fig. 5. Secondary word transfer rates for the various O.S.'s

Fig. 6. Effects of varying disk delay time

Fig. 7. Simulation results for basic system (1800 RPM disk)

Fig. 8. Simulation results for 3600 RPM disk system

Fig. 9. Simulation results for 900 RPM disk system

Fig. 10. Processor usage as a function of memory size for one user

Fig. 11. Response time as a function of memory size for one user

Fig. 12. Simulation results for 1800 RPM disk, 12K of 1 μsec memory

Fig. 13. Simulation results for 3600 RPM disk, 12K of 1 μsec memory

Fig. 14. Simulation results for 900 RPM disk, 12K of 1 μsec memory

Fig. 15. Simulation results for 1800 RPM disk, 8K of 0.5 μsec memory

Fig. 16. Average disk access time as a function of number of users for the various system configurations

Fig. 17. Response time as a function of number of users for the various system configurations

Fig. 18. Relative cost per user times response time for various system configurations

Fig. 19. Relative cost/CPU time unit for various system configurations

HARDWARE CONFIGURATION

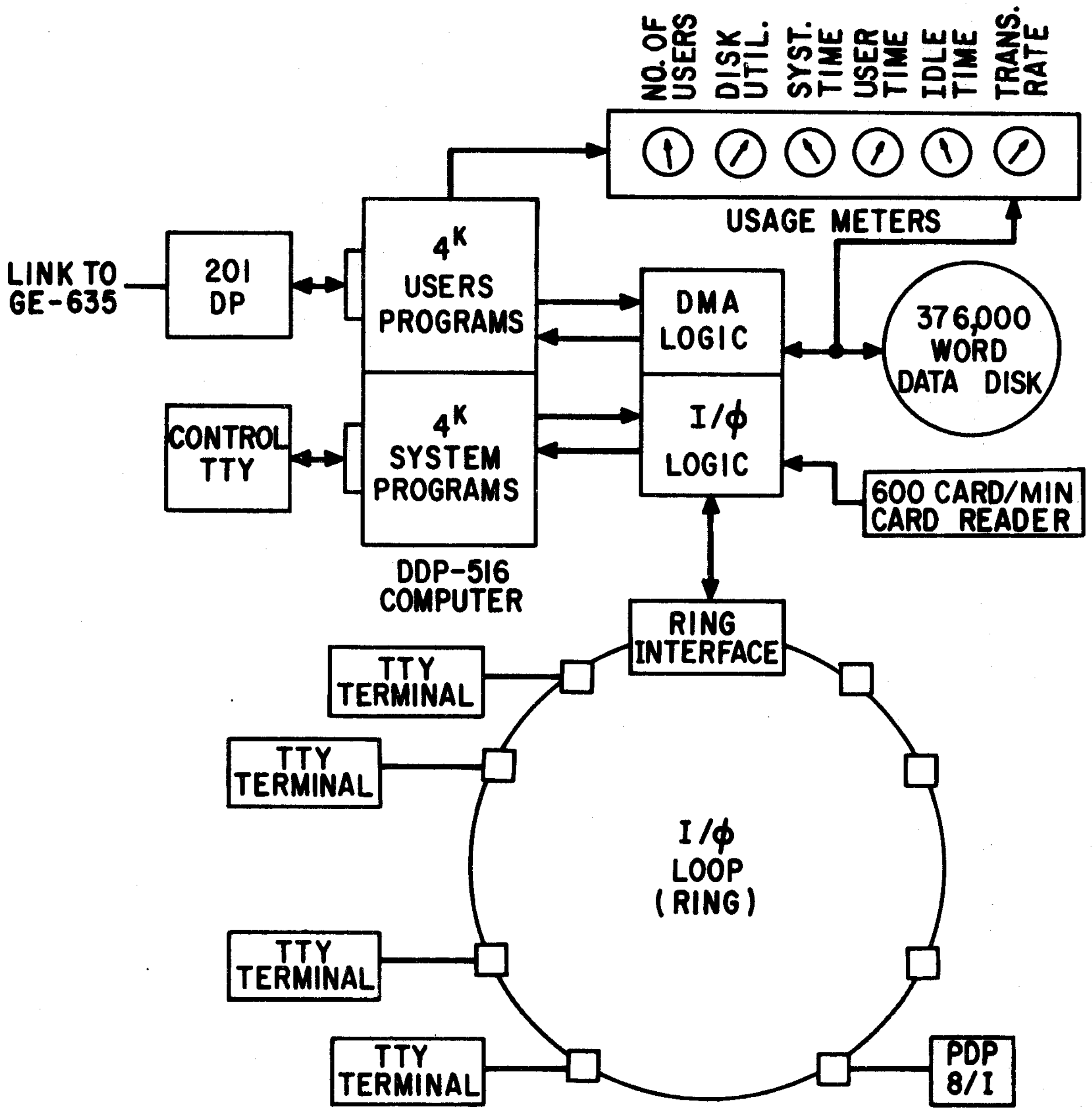
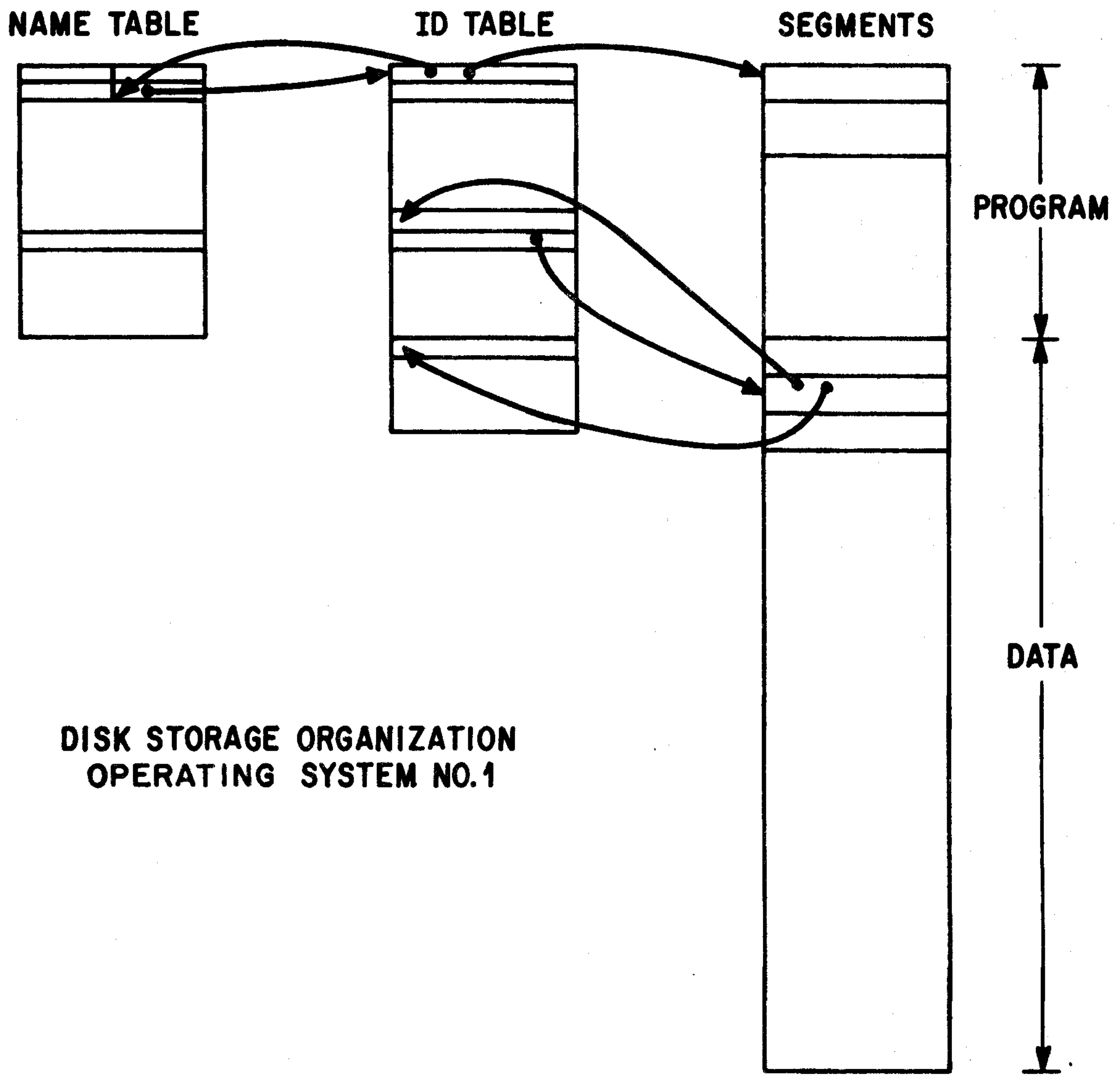


FIGURE 1

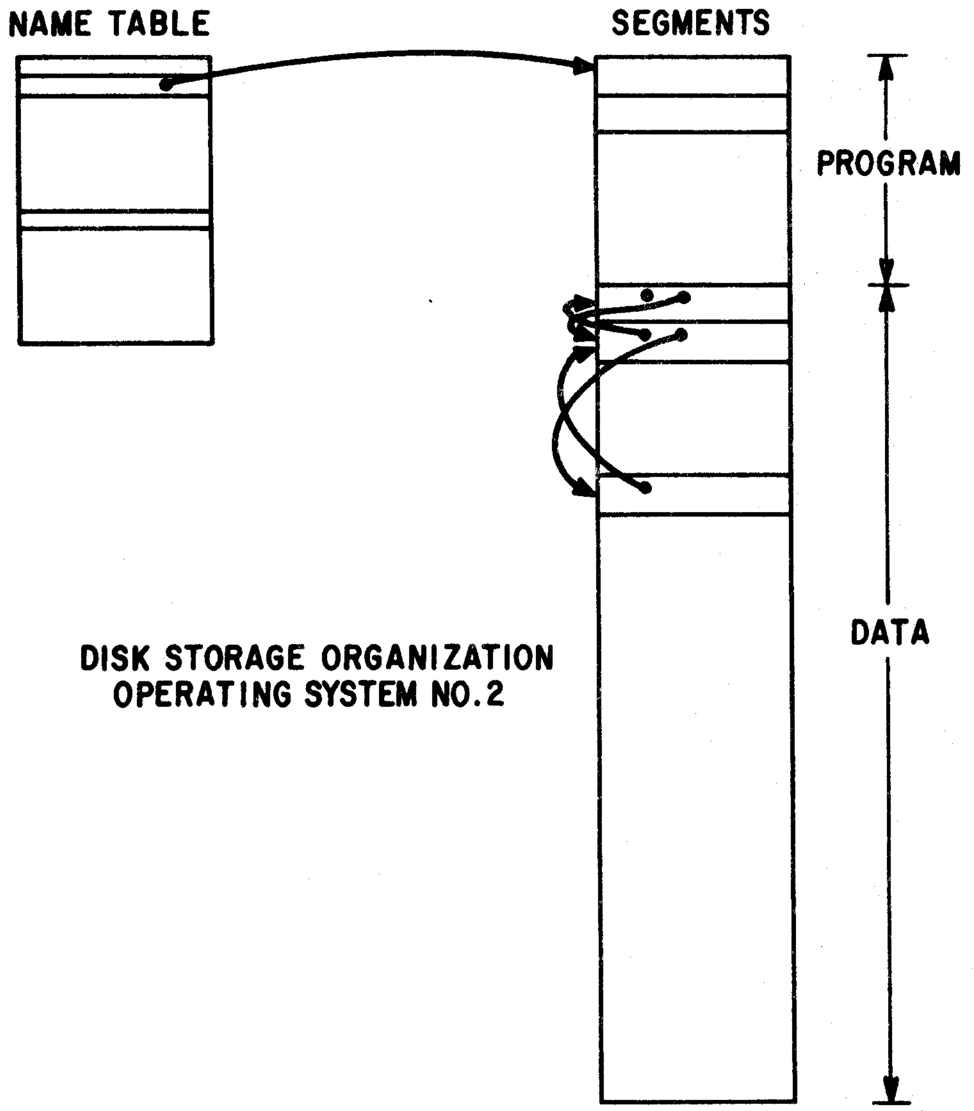
DISK MAP (O.S. NO.1)



DISK STORAGE ORGANIZATION
OPERATING SYSTEM NO.1

FIGURE 2

DISK MAP (O.S. NO. 2)



DISK STORAGE ORGANIZATION
OPERATING SYSTEM NO. 2

FIGURE 3

AVERAGE DISK ACCESS TIMES FOR
THE VARIOUS OPERATING SYSTEMS

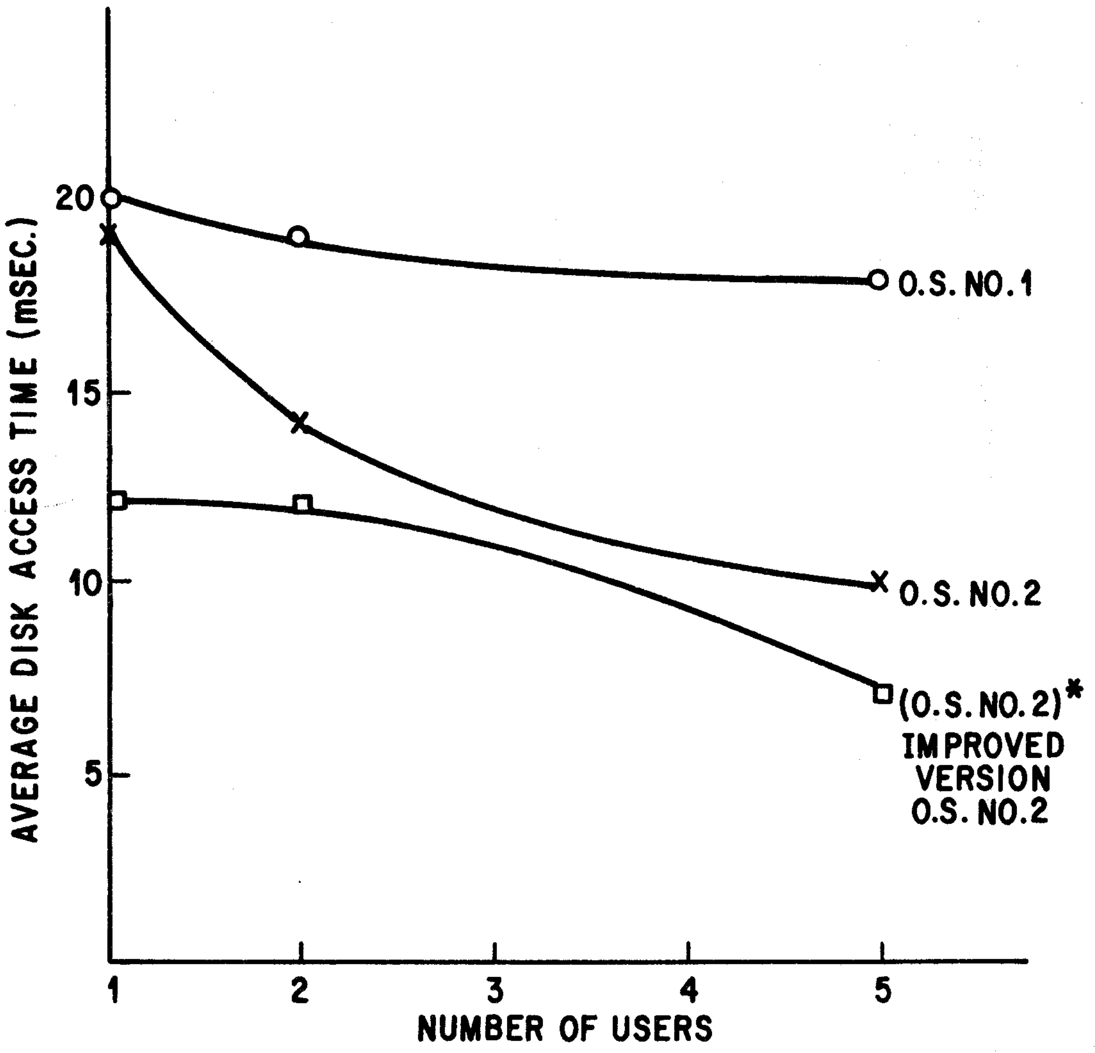


FIGURE 4

SECONDARY WORD TRANSFER RATES FOR THE VARIOUS OPERATING SYSTEMS

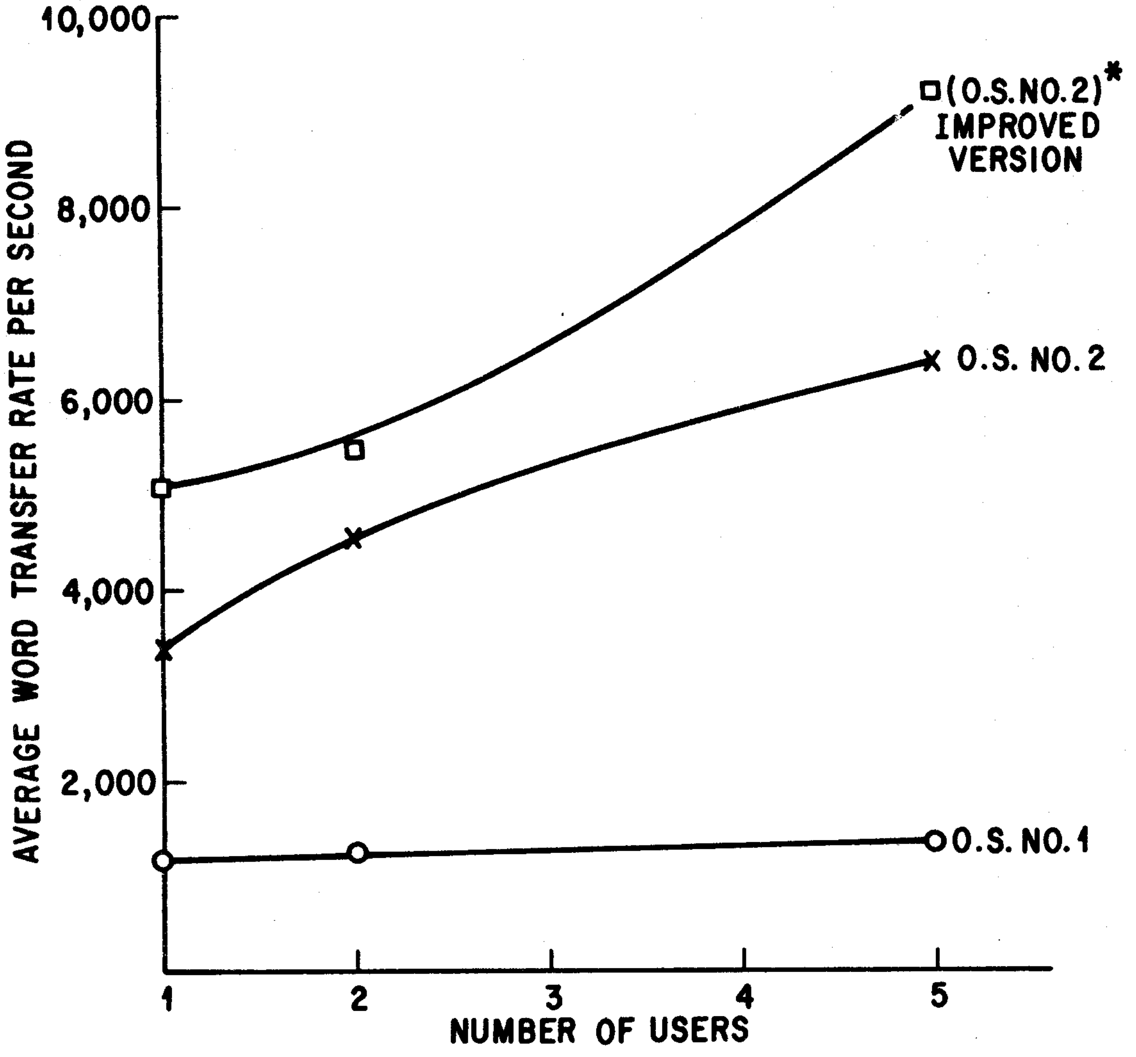


FIGURE 5

EFFECTS OF VARYING DISK DELAY TIME

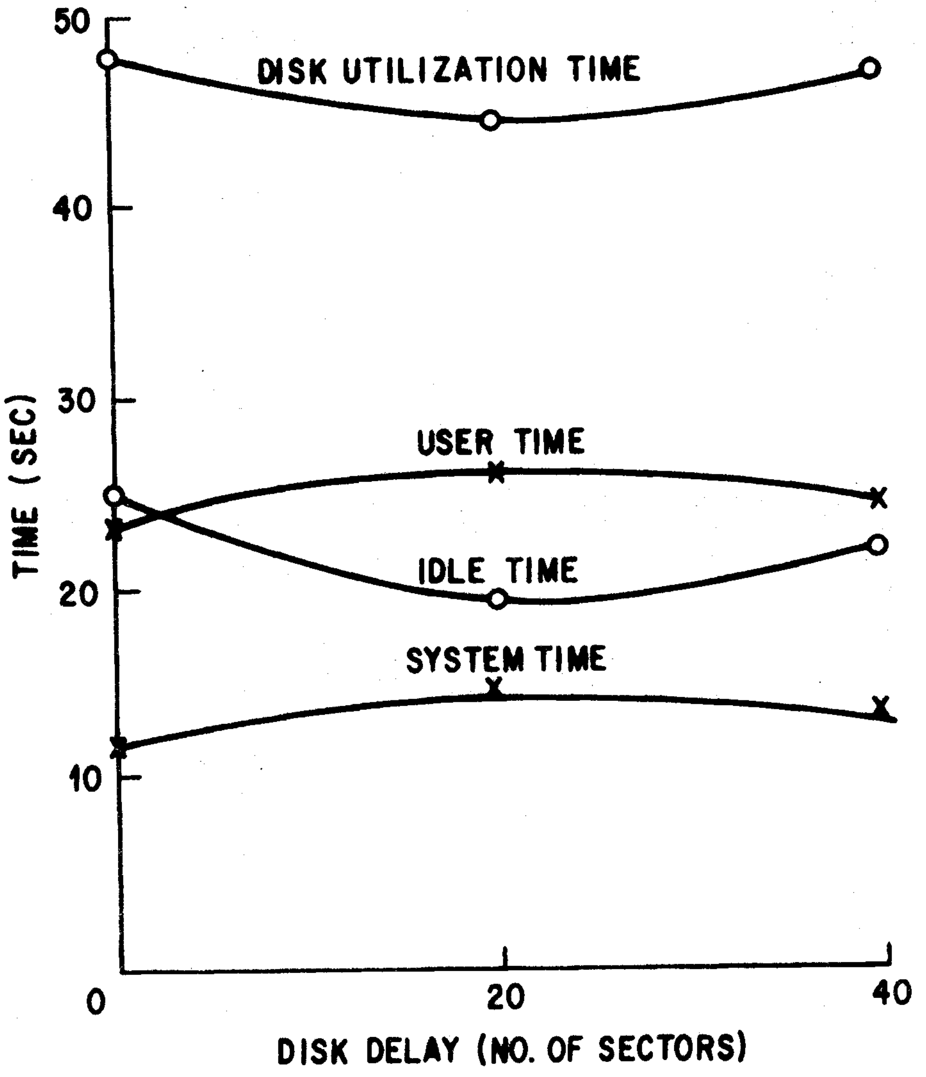


FIGURE 6

SIMULATION RESULTS FOR BASIC SYSTEM (1800 RPM DISK)

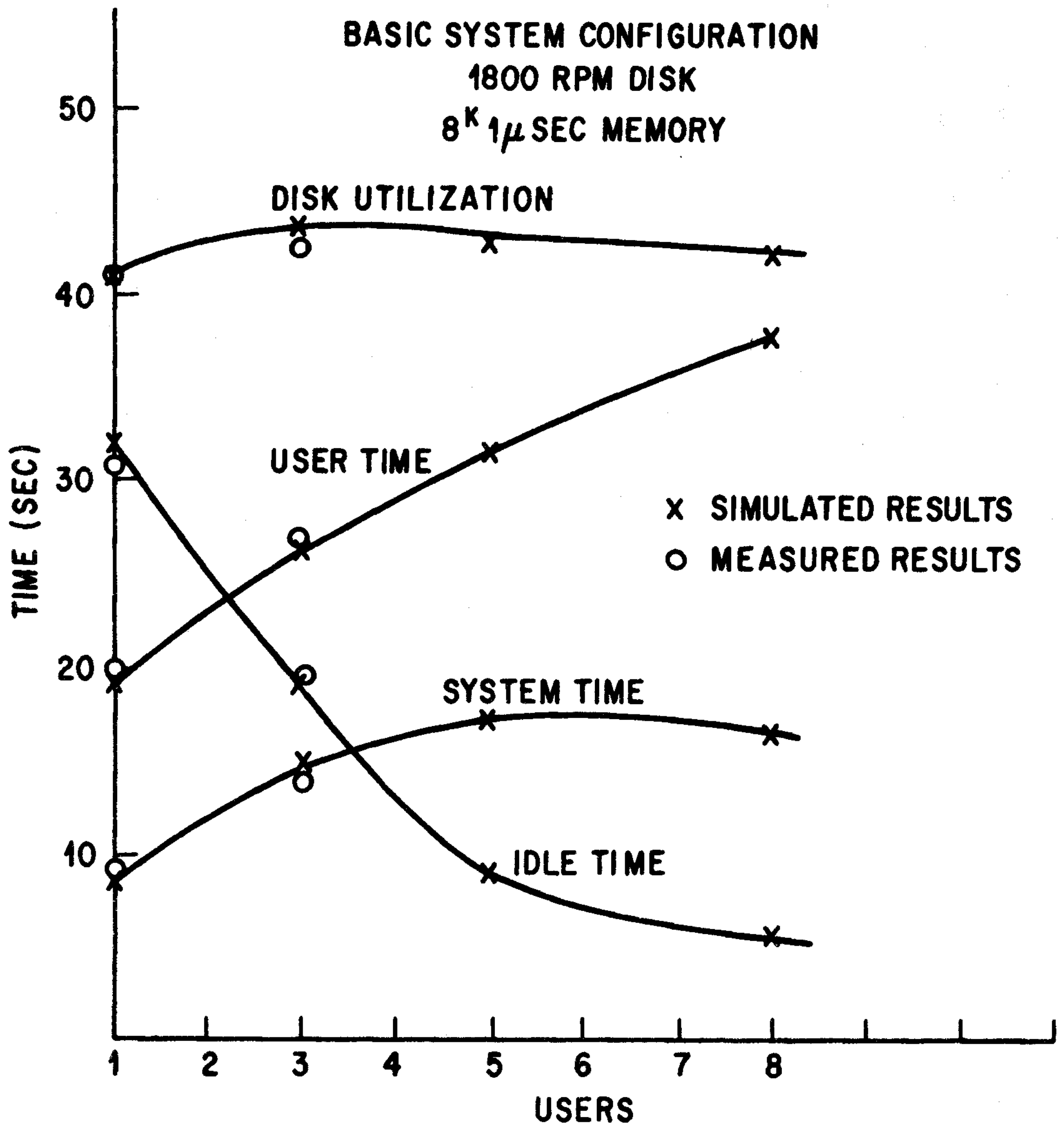


FIGURE 7

SIMULATION RESULTS FOR 3600 RPM DISK SYSTEM

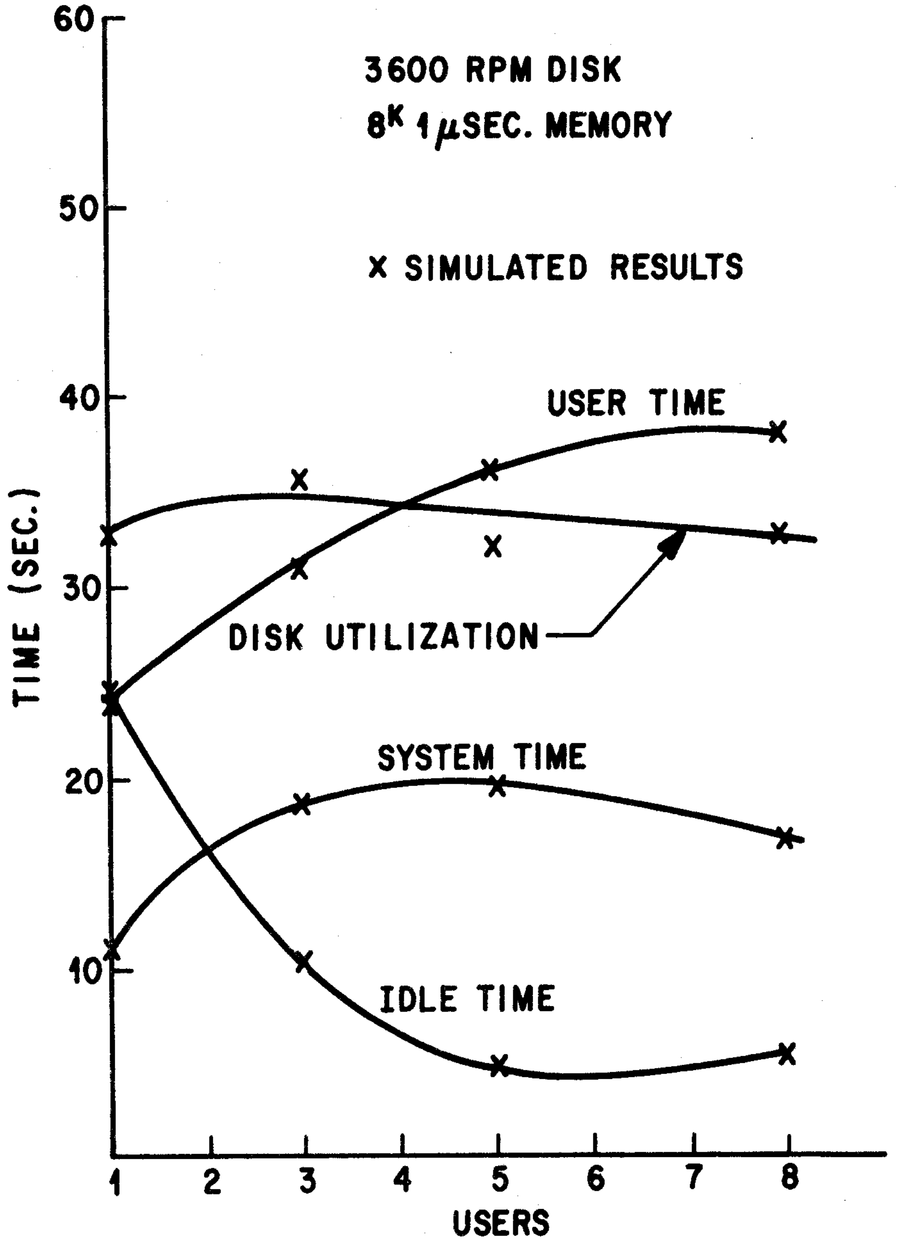


FIGURE 8

SIMULATION RESULTS FOR 900 RPM DISK SYSTEM

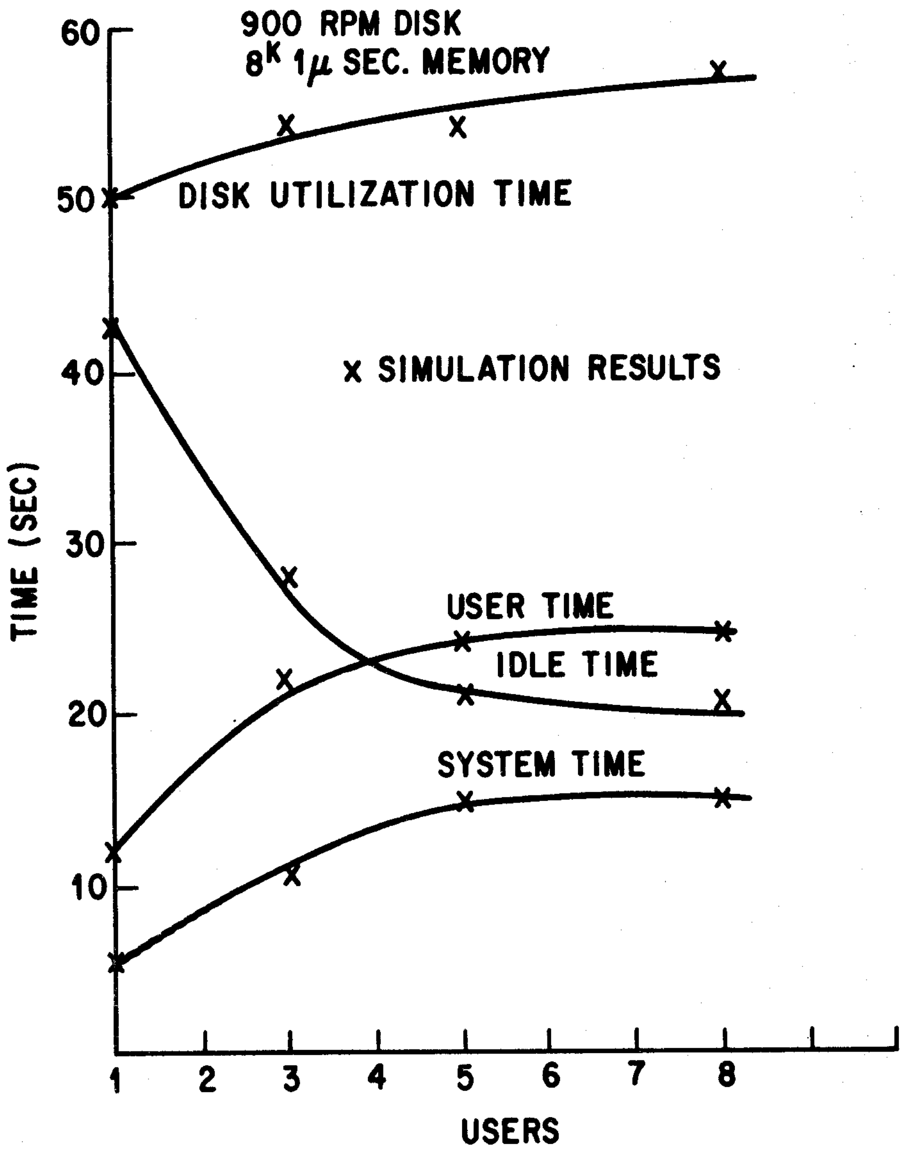


FIGURE 9

PROCESSOR USAGE AS A FUNCTION OF MEMORY SIZE FOR ONE USER

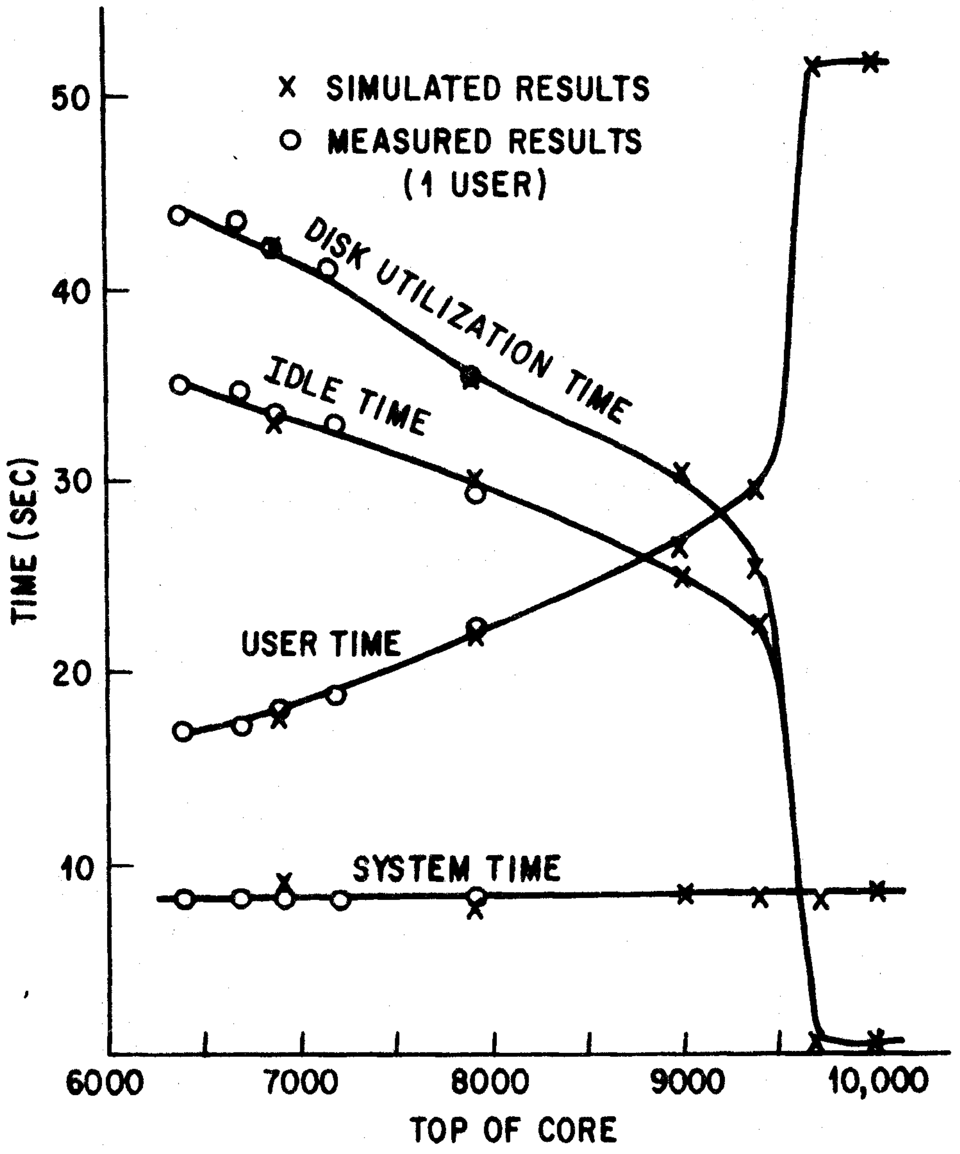


FIGURE 10

RESPONSE TIME AS A FUNCTION OF MEMORY SIZE FOR ONE USER

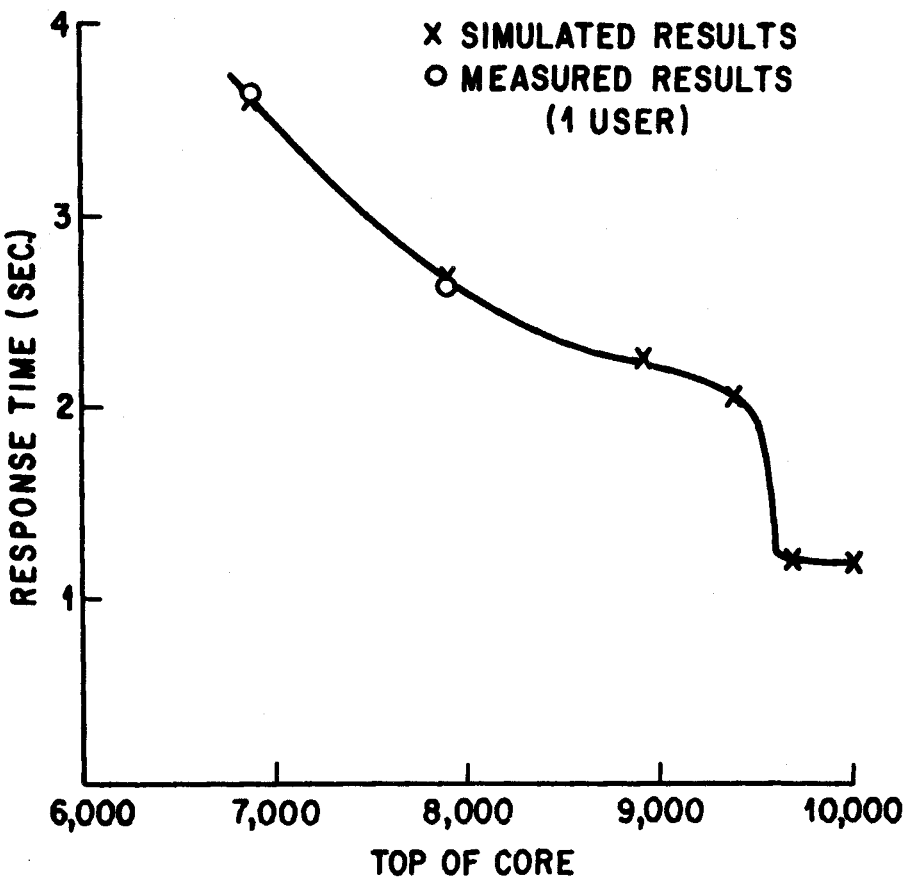


FIGURE 11

SIMULATION RESULTS FOR 1800 RPM DISK SYSTEM WITH 12^K OF 1μSEC. MEMORY

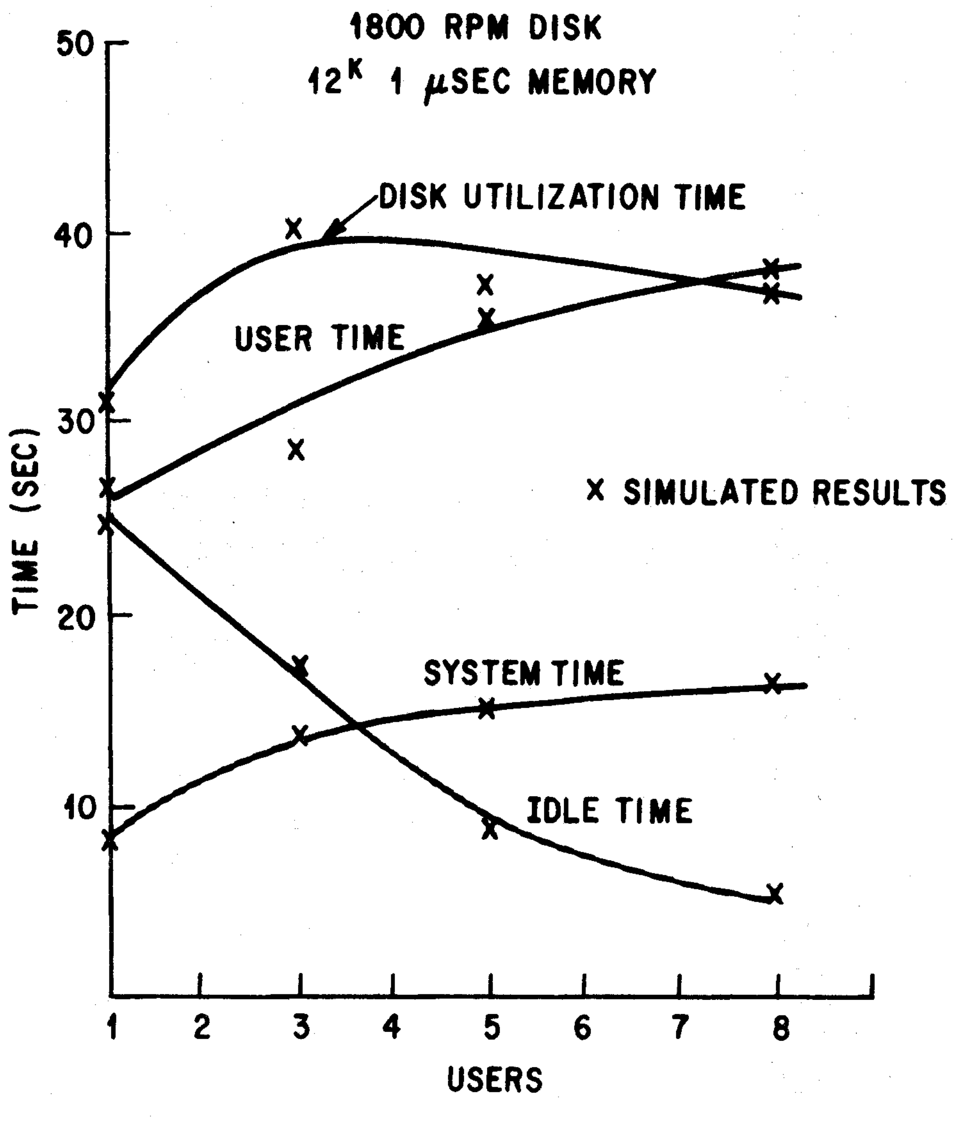


FIGURE 12

SIMULATION RESULTS FOR 3600 RPM DISK SYSTEM WITH 12^K OF 1μSEC MEMORY

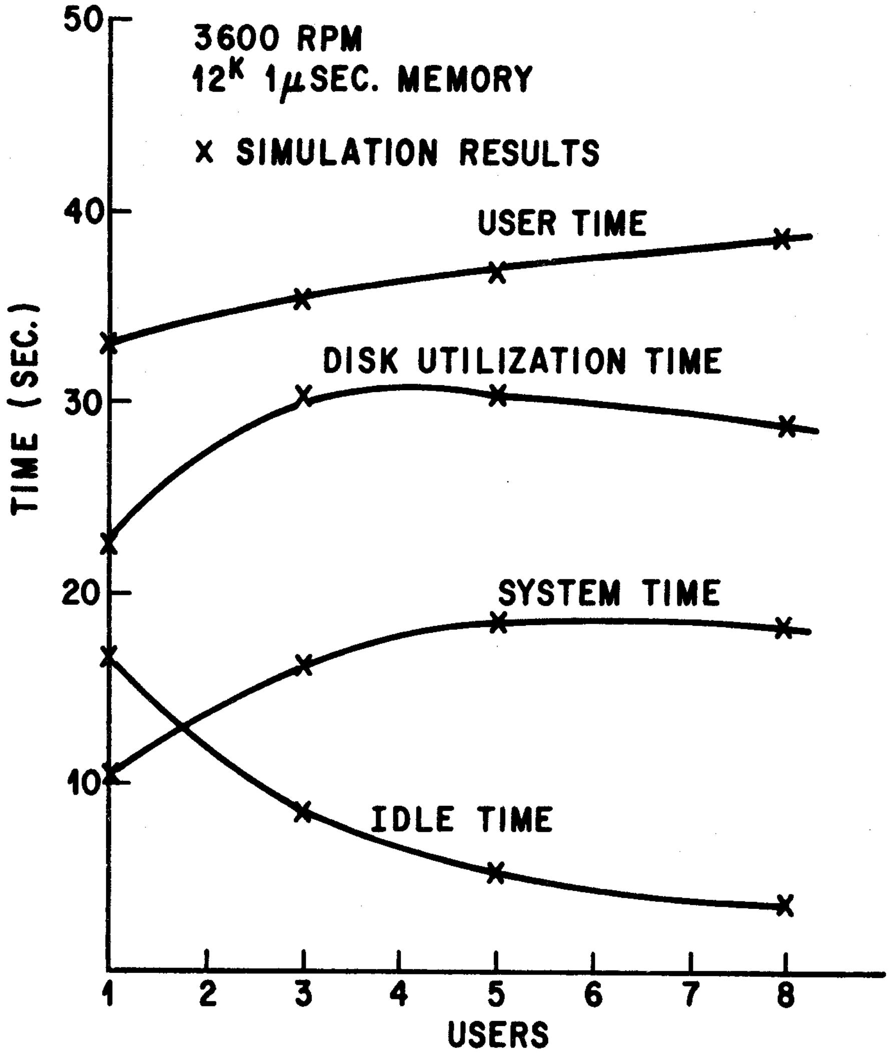


FIGURE 13

SIMULATION RESULTS FOR 900 RPM DISK SYSTEM WITH 12^K OF 1μ SEC MEMORY

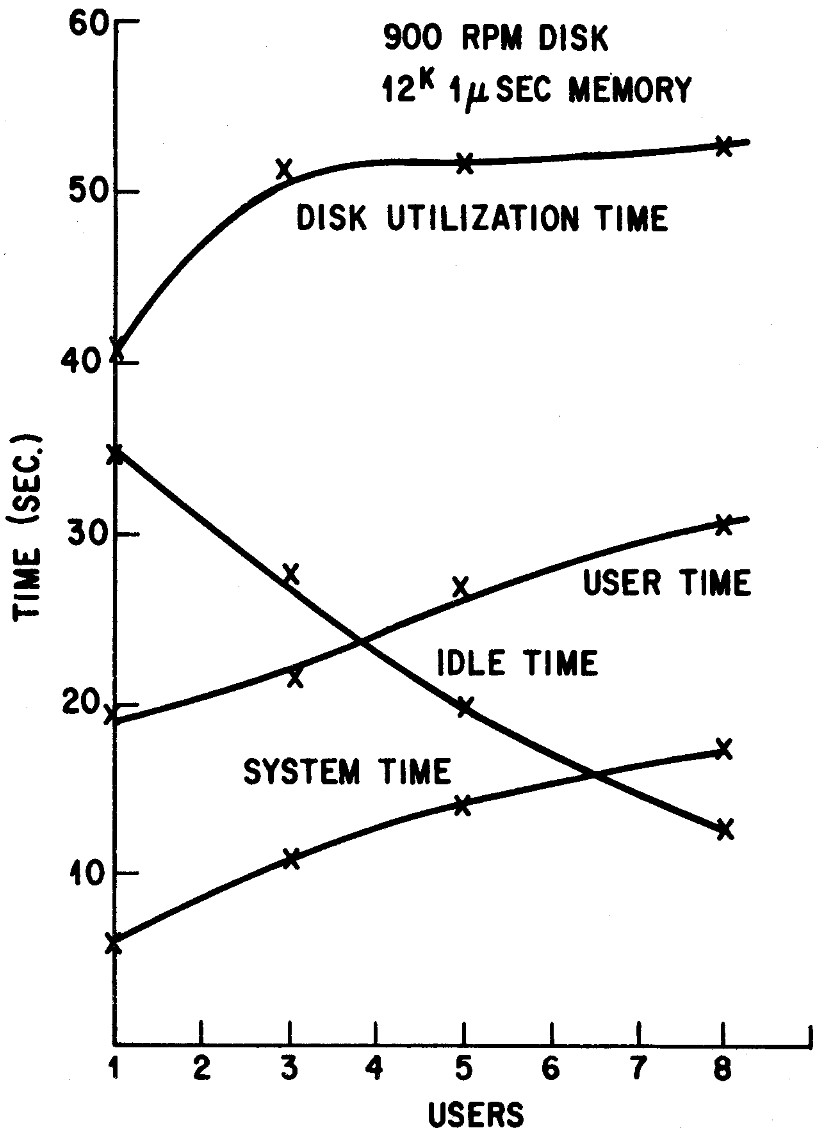


FIGURE 14

SIMULATION RESULTS FOR 1800 RPM DISK SYSTEM WITH 8^K OF 0.5μSEC MEMORY

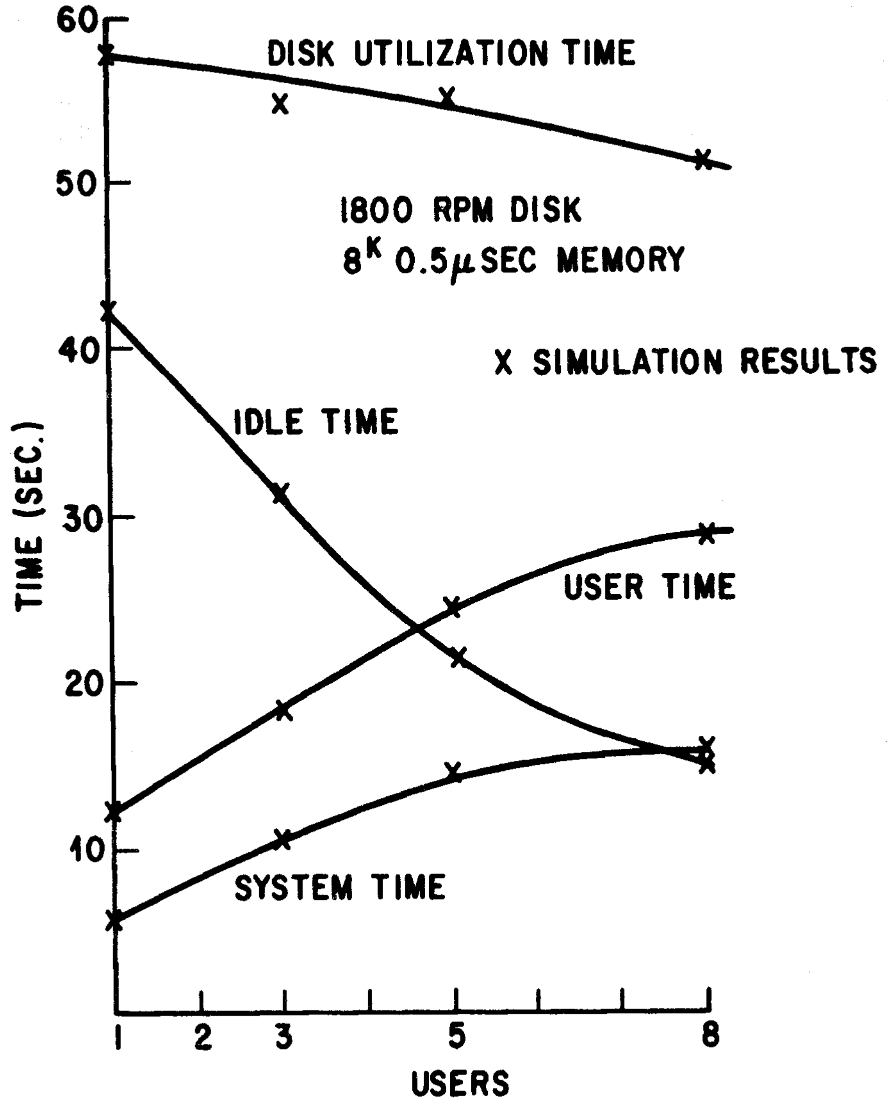
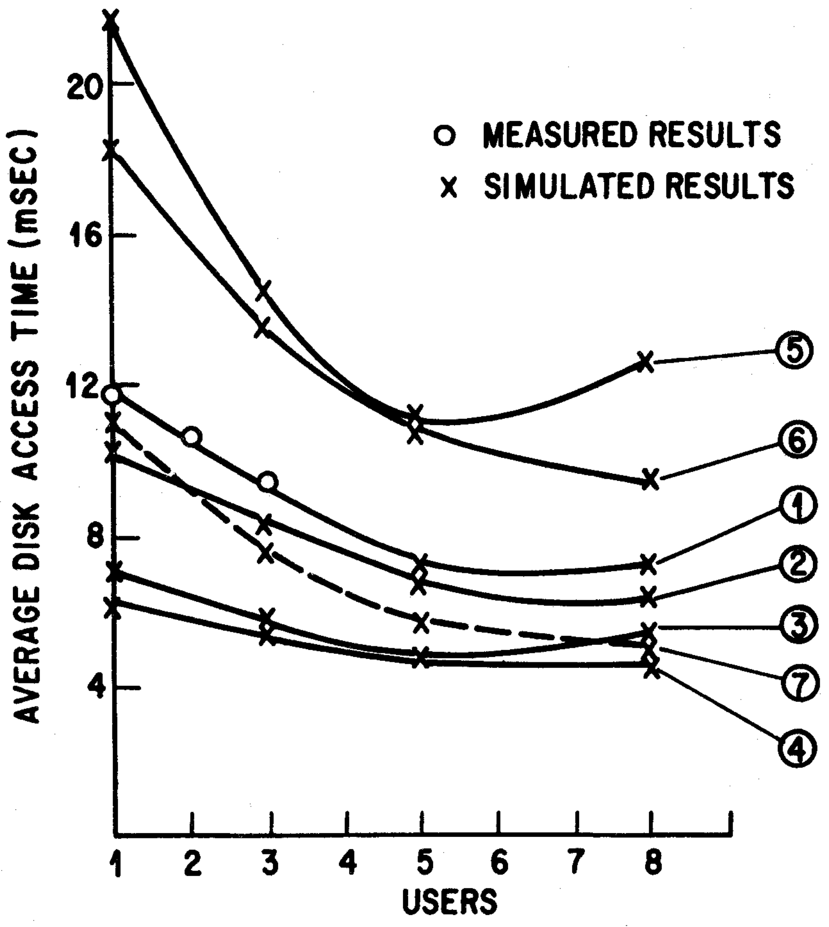


FIGURE 15

AVERAGE DISK ACCESS TIME FOR THE VARIOUS SYSTEM CONFIGURATIONS



① 1800 RPM DISK, 8^K 1μSEC MEMORY

② 1800 RPM DISK, 12^K 1μSEC MEMORY

③ 3600 RPM DISK, 8^K 1μ SEC MEMORY

④ 3600 RPM DISK, 12^K 1μ SEC MEMORY

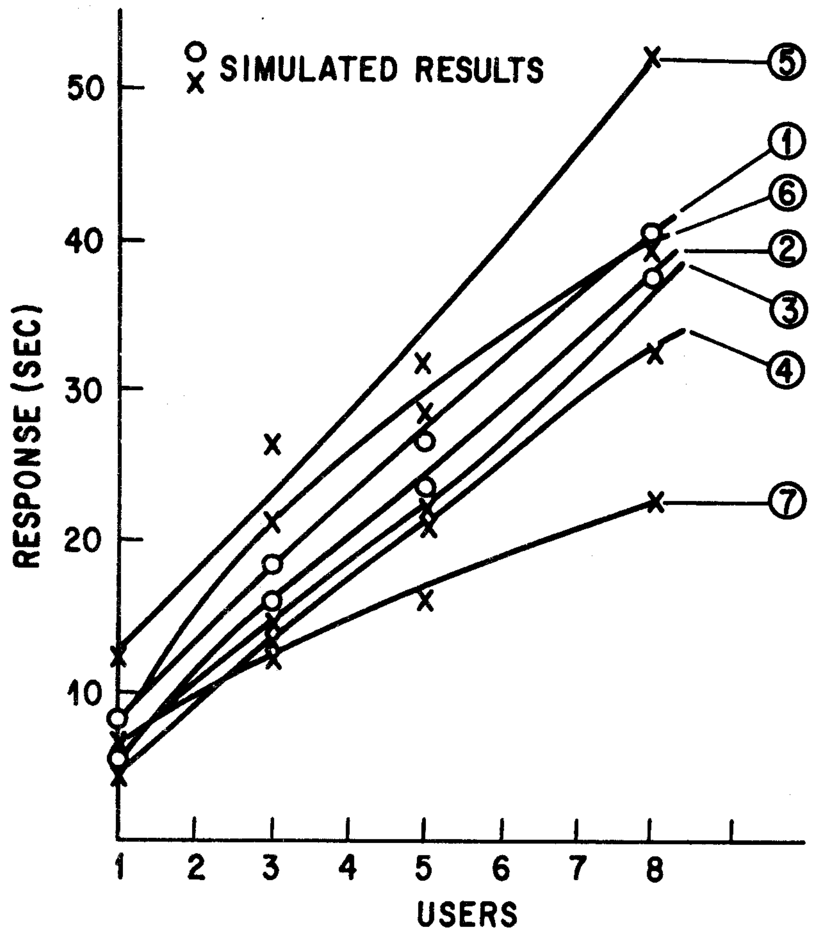
⑤ 900 RPM DISK, 8^K 1μSEC MEMORY

⑥ 900 RPM DISK, 12^K 1μSEC MEMORY

⑦ 1800 RPM DISK, 8^K 0.5μSEC MEMORY

FIGURE 16

RESPONSE TIME FOR THE VARIOUS SYSTEM CONFIGURATIONS



① 1800 RPM DISK, 8^K 1μSEC MEMORY

② 1800 RPM DISK, 12^K 1μSEC MEMORY

③ 3600 RPM DISK, 8^K 1μSEC MEMORY

④ 3600 RPM DISK, 12^K 1μSEC MEMORY

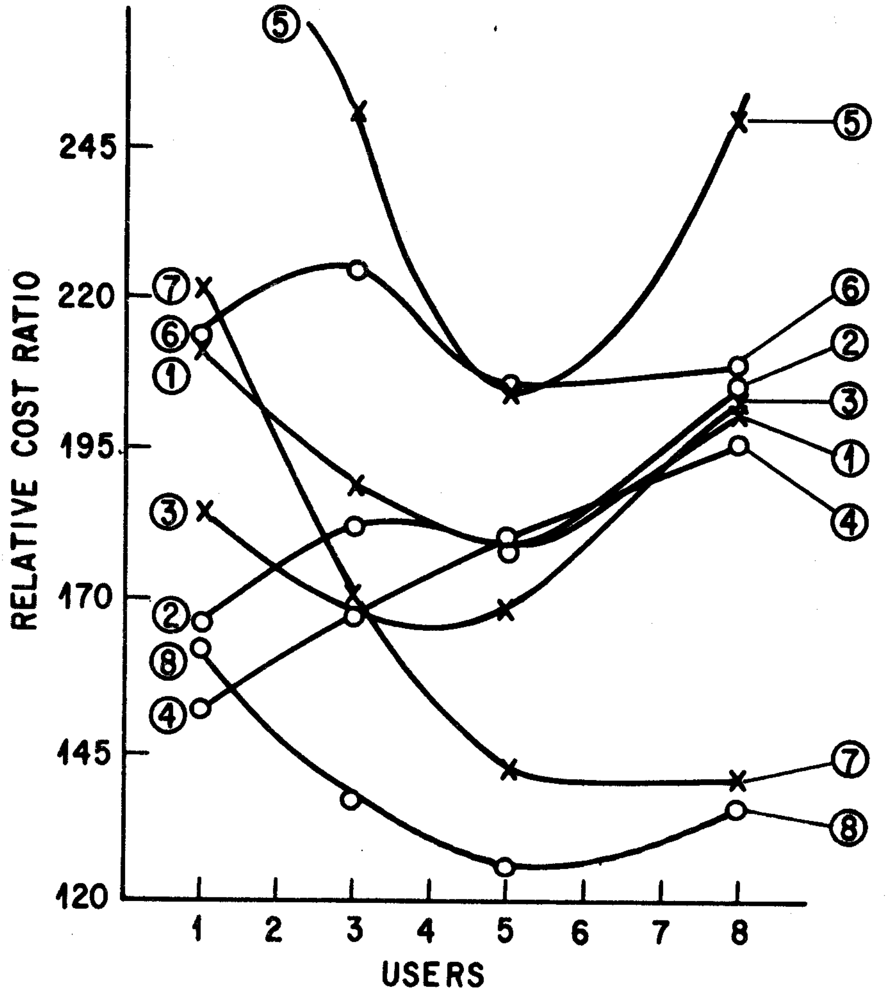
⑤ 900 RPM DISK, 8^K 1μSEC MEMORY

⑥ 900 RPM DISK, 12^K 1μSEC MEMORY

⑦ 1800 RPM DISK, 8^K 0.5μSEC MEMORY

FIGURE 17

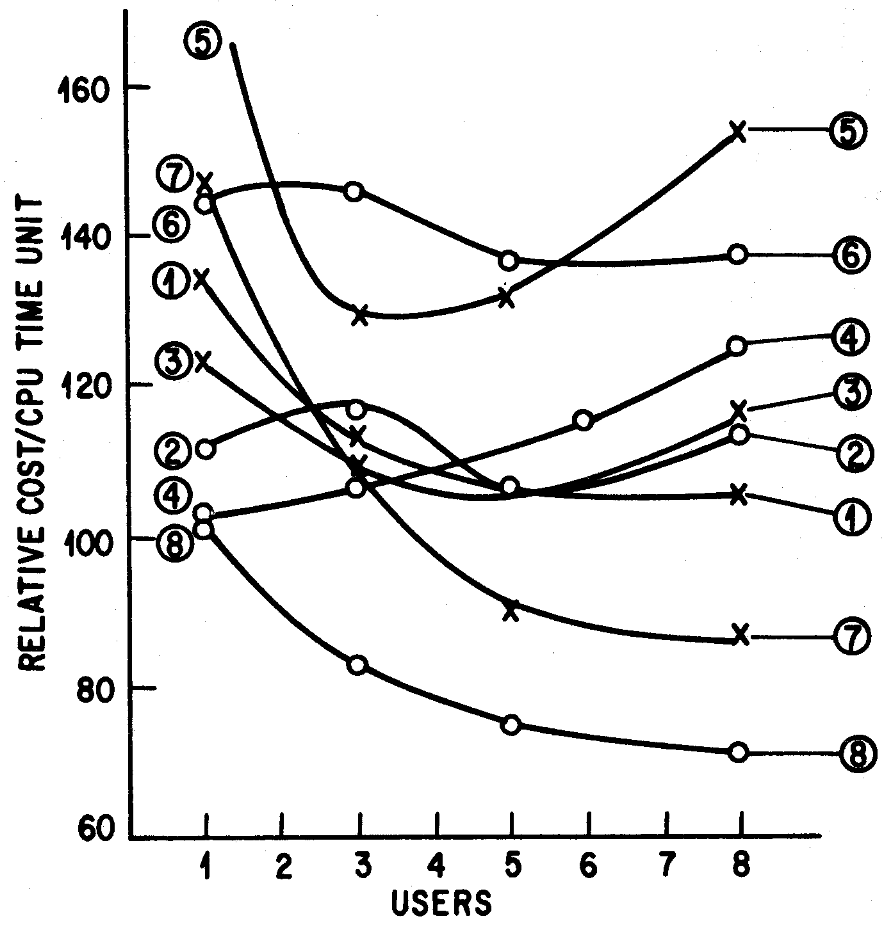
RELATIVE COST PER USER TIMES RESPONSE TIME FOR VARIOUS SYSTEM CONFIGURATIONS



① 1800 RPM DISK, 8K 1μSEC MEMORY
② 1800 RPM DISK, 12K 1μSEC MEMORY
③ 3600 RPM DISK, 8K 1μSEC MEMORY
④ 3600 RPM DISK, 12K 1μSEC MEMORY
⑤ 900 RPM DISK, 8K 1μSEC MEMORY
⑥ 900 RPM DISK, 12K 1μSEC MEMORY
⑦ 1800 RPM DISK, 8K 0.5μSEC MEMORY
⑧ 3600 RPM DISK, 8K 0.5μSEC MEMORY

FIGURE 18

RELATIVE COST/CPU TIME UNIT FOR VARIOUS SYSTEM CONFIGURATIONS



① 1800 RPM DISK 8^K 1μSEC MEMORY

② 1800 RPM DISK, 12^K 1μ SEC MEMORY

③ 3600 RPM DISK, 8^K 1μSEC MEMORY

④ 3600 RPM DISK, 12^K 1μSEC MEMORY

⑤ 900 RPM DISK, 8^K 1μSEC MEMORY

⑥ 900 RPM DISK, 12^K 1μSEC MEMORY

⑦ 1800 RPM DISK, 8^K 0.5μSEC MEMORY

⑧ 3600 RPM DISK, 8^K 0.5μSEC MEMORY

FIGURE 19