A MODULAR SIMULATION OF TSS/360

John W. McCredie
Assistant Professor of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Steven J. Schlesinger
Department of Computer Science
Carnegie-Mellon University
Pittsburgh, Pa. 15213

Summary

This paper describes a modular simulation of TSS/360, a virtual memory time-sharing computer system. A detailed model of the table driven scheduler is embedded in an environment consisting of higher level models of other subsystems. The original goal of the project was to provide a useful tool to aid in the tuning of the monitor's scheduling algorithm. As the simulation evolved, the model served as a vehicle to study proposed hardware changes.

Introduction

TSS/360 is the time-sharing operating system for the IBM 360 Model 67[1]. This computer differs from the standard 360 line because of hardware additions designed to create an efficient two dimensional addressing structure utilizing segments and pages.* Hardware performs the dynamic address translations necessary to make a large virtual memory a reality. The goal of demand paging organizations is to implement a system in which each user may act as if he has a large, private, directly addressable memory. Some parts of this memory may be shared with other users of the system. The operating system manages all system resources and frees the user from many allocation problems.

The algorithm which schedules and dispatches tasks in this multiprogrammed, time-shared environment is a section of TSS called the table driven scheduler. Since each TSS system has a different user community to satisfy, and a different hardware configuration, the parameters in the table driven scheduler may be set by each installation. Many of the parameters within the table are branching codes to other sections of the table. Thus the scheduling algorithm and its parameters are variables that must be determined locally. This flexibility places the responsibility for tuning TSS upon system analysts at each installation.

This paper describes a simulation designed to aid in determining the value of entries in the TSS schedule table. Why simulate to solve this problem? An alternative would be to try different schedule tables in the operational system. Evaluating the results of different tables would be a hard process in a production environment. Bad designs would degrade the system and adversely impact the user community until adequate evaluations were complete. The complexity of the scheduling algorithm ruled out an analytic approach based upon queueing theory. A limited simulation seemed to be a worthwhile approach. To be useful, such a model would have to describe the basic effects of schedule table changes on the response of the system. The model could not be too expensive in terms of design and running time.

*See Wilkes[2], chapter 4, for a good discussion of virtual memory design.

The main purpose of a descriptive model is to account for observed phenomena of physical systems. The complexity of TSS requires that any small model address itself to a limited and constrained subset of input and output state variables. The model is thus an abstraction of certain important features of interest to an analyst or designer. Simplifications required to make the abstraction manageable limit both its scope and power. A flexible, modular design allows for future expansion both to correct for oversimplifications discovered during validation, and to adapt to changing goals. The design proceeded in a top down manner with various subsystems being represented by simple probabilistic approximations at first and then by more complex descriptions as the model evolved. The scheduling algorithm section is much more detailed than other subsystems such as the paging disks. The model is written in SIMULA, an Algol based discrete-event simulation language in which modular design is easy.

Related Work

Papers in the simulation literature describe a number of different approaches to the modeling of computer systems. Fine and McIsaac[3] describe a simulation of the System Development Corporation time-shared computer, the Q-32. Scherr[4] presents a detailed model of the CTSS system at M.I.T. Both of these studies model all subsystems at a common level of detail. Models by Katz[5,6] concentrate on state transition approaches which include many details concerning software structure and program characteristics. Barker and Watson[7] and Martin[8] discuss very detailed models of the IBM 360 including virtually all hardware and software functions. These latter studies are large efforts requiring a number of man-years of effort. Nielsen[9,10] has developed programming languages for building models of computer systems. In [9] he uses his earlier work to model the 360/67. This model is another member of the class of very detailed, complex, total system models. SCERT (System and Computer Evaluation and Review Technique) is a proprietary development of COMRESS, and is used to evaluate different computer systems with respect to a job profile from an installation. CASE (Computer-Aided System Evaluation) is a commercial product similar to SCERT in which users complete forms to describe workloads, hardware configurations, and software systems. SAM (System Analysis Machine) is another language specially designed to help build models of computer systems.**

The common denominator of these simulations is an attempt to model all aspects of a system under study at a uniform level of detail. In design philosophy, they are attempts to provide tools capable of answering almost any reasonable question about the system. This generality must be paid for by large investments in

**See the recent article in Computer Decisions[11] for a discussion of the commercial packages SCERT, CASE, and SAM.

personnel and computer time.

The goal of this paper is to show that a useful model can be designed to answer a limited set of questions about a complex system without detailed modeling of all system components. This model has the basic characteristic that different system modules have vastly different levels of detail in the simulation. The areas of emphasis may change as the model evolves.

Choice of Language

The model is written in SIMULA, an Algol based discrete event simulation language. We chose a simulation language over a procedural language such as Algol or Fortran since the program must constantly schedule and cancel events. We did not use GPSS for several reasons. SIMULA is a compiled language and therefore executes much faster than GPSS which is interpreted. Since SIMULA is a direct extension of Algol, all of the procedural and arithmetic capabilities of Algol are present. Perhaps the most important feature was the natural use of the "process" concept of SIMULA in modeling computer systems. Such systems are naturally viewed as a set of hardware, software, and human elements which interact. In SIMULA each process is a member of a class of entities having the same data and event structures. Processes may interact with each other in the same manner as computer subsystems. This feature gives SIMULA programs a degree of modularity which greatly eases model modification and expansion.*

One analyst spent about three man-months studying the system and building the model which runs on a Univac 1108. The compiled code requires 4,000 words of storage. Five minutes of simulated system time takes about one minute of 1108 time.

Model Design

Model design proceeded in a top-down fashion. In order to keep the model as simple as possible, elements of the real system were included only when necessary because of interactions with the scheduler. The primary goal of the model was to obtain the distribution function of response times experienced by a user at an interactive terminal as a function of the type of request submitted. The model simulates interactions between hardware, software, and the user population. Figure I illustrates the model's structure.

Three hardware facilities appear: the CPU, the paging devices, and memory. The CPU appears implicitly in all software elements of the system and in the execution of user programs. No CPU characteristics such as clock cycle time or instruction times are included, although they are implicit in the amount of computation time used by user programs.

Two types of paging devices are included in the model: disks and drums. Disks are viewed as an infinite source of new pages demanded by executing programs and as an infinite storage facility for pages written out by the monitor. The disk units on the real system were IBM 2314's. Actual operation of these units is complex since arm seeks on different spindles can be overlapped, and software disk management routines try to optimize arm movements to maximize the flow of pages into core. Instead of modeling this process directly, we determine the access time of a page by drawing a number from a distribution. The statistical characteristics of this distribution reflect the operation of the actual system. The parameters of the distribution were determined by observations from the system logging information. Drums are represented by their revolution times and their capacity in pages. The distribution of access times for pages from a drum is uniform from zero to the revolution time.

The 360/67 at CMU has a relatively small amount (128 pages) of high speed core (.75 μsec. cycle time). The monitor uses all but a few of these high speed pages. User programs reside in LCS (Large core storage, 8 μsec. cycle time). Since all user programs compete for and share 550 pages of LCS, these pages are explicitly included in the model. The high access time of LCS is reflected in the computation time of user programs.

There are two major software routines in the model - the timer interrupt handler and the table driven scheduler. Minor software functions occur implicitly in other parts of the model. A timer interrupt occurs when a user program has CPU control at the end of its time quantum. The interrupt handler may, depending on

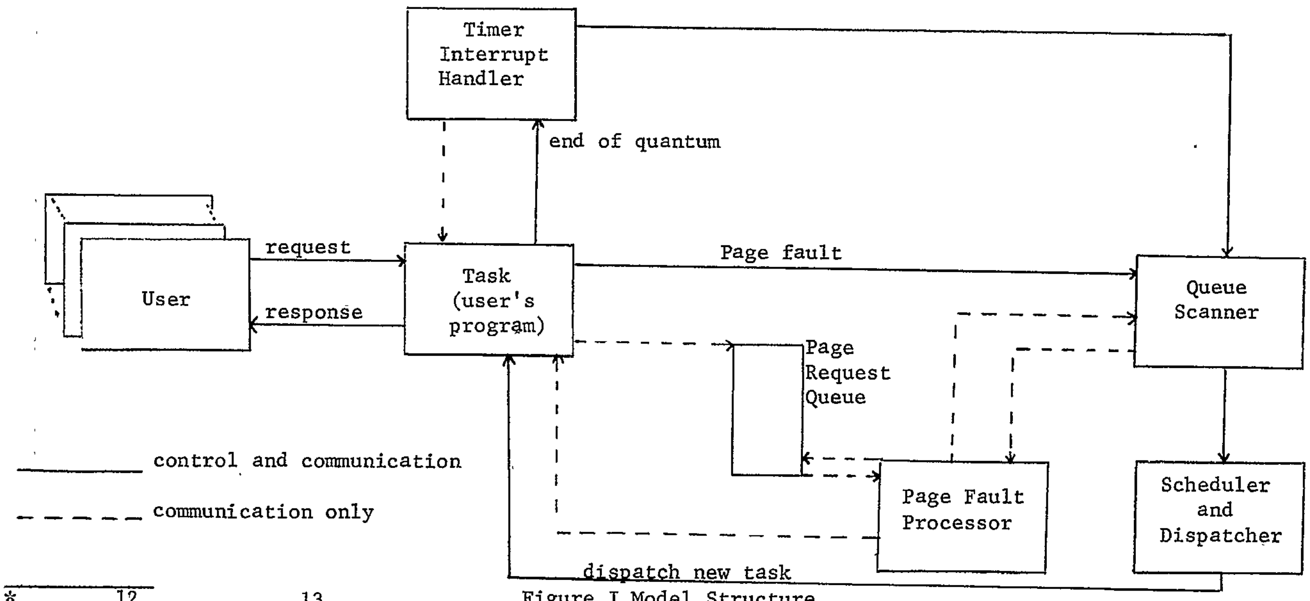


Figure I Model Structure

*See Dahl[12] and McCredie[13] for discussions of the structure and use of SIMULA.

the program's scheduling parameters, do any of the following: force a time-slice end and write pages onto the disk (or drum); change the scheduling parameters; give the program an additional time quantum. The timer routine in the model performs the functions of several subroutines of TSS/360 which are called when a timer interrupt occurs.

The table driven scheduler is the most detailed portion of the simulation. Each program in the system is assigned an entry in the schedule table. This entry contains maximum limits on CPU usage and paging activity of a program. A program exceeding the limits is penalized by loss of eligibility for CPU allocation and possible lowering of priority. This penalty occurs by changing the program's schedule table entry to a new one depending upon how the program exceeded the bounds of its previous entry. For each maximum there is a new schedule table entry to which the program will be assigned if that limit is exceeded. The monitor interrupts a program during its time slice to check if any bounds have been violated.

The scheduler maintains several lists of programs, each one having a different eligibility for CPU allocation. User programs move among the lists depending on their schedule table entry and operating characteristics. The schedule table in the actual system has limits on additional operating characteristics of programs to enable fine-tuning of the scheduling algorithm.

Program behavior is characterized by periods of CPU usage separated by page faults. When the program receives CPU control from the scheduler it is interrupted only by a timer interrupt or page fault. While it has CPU control, no classification is made of language used, system functions called, or other modes of activity. The only parameter of interest during CPU usage is the time necessary to complete the user request. Paging activity is based upon the working set concept of Denning[14]. Each request specifies the working set size for that request. The user program then calls for sufficient pages to fill the working set. There is no distinction concerning the contents of each page. Only the number in core is of interest.

The users in the model make terminal requests and wait for system response at the terminal. A request consists of the amount of CPU time required and the number of new pages which have to be brought into core for a complete working set. The model draws both of these parameters from distributions either approximating observed system behavior or testing hypothetical modes of system use. The distribution of response time experienced by a user is the statistic of primary interest in the simulation. Non-conversational batch jobs have the same structure as interactive tasks, but the distributions of their operating characteristics and their priorities are different.

Model Validation

A prime reason for building a simulation of a complex system is to construct a test environment in which one may experiment with different structural configurations. A model is the set of functional relationships indicating how output state variables respond to different settings of input variables. If the main purpose of a particular model is to describe system responses over a limited and well documented operating range, the structural correspondence of the model and the real system need not be exact. There are often conflicting hypotheses that adequately describe a set of empirical data.* For example, the output of a stochastic process may appear to be Poisson with rate λ. If one has only aggregate data, he may not be able to determine if the actual mechanism within the "black box" is a single Poisson source with rate λ, or n sources with rates λ/n. For many purposes it may not matter which hypothesis is correct; both may lead to the same conclusion.

If, however, the central purpose of a simulation is to aid in the study of a class of structural changes, it is obvious that these basic relationships in the model and the real system must be as nearly identical as possible. Within a single model one may find gross functional simplifications of a descriptive nature for some subsystems, and very detailed structural relations for others. As a model evolves, and its goals change, some sections may become more detailed and others more simple. A modular approach to model formulation and construction will allow this type of growth. A poor language or a bad initial design may force a complete re-write as the system evolves. This consequence is expensive in terms of both time and money.

At each stage of model development, one should have a validation procedure to insure that a simulation is doing the job for which it was designed. Our model, with its many simplifications in certain areas and great detail in others, should predict system response to a wide range of inputs. Since we expended much effort to model the schedule table as described in system manuals, we were confident of its accuracy. However, other areas of the model are rough approximations. In an effort to avoid what many people have called "the model-is-reality syndrome", we used three different validation procedures.

The 360 at Carnegie has a software monitoring facility that allows us to make significant measurements of the system in a production environment. The analyst initializes the system to create an output record, on magnetic tape, for every internal system event of interest. This logging tape is saved for later off-line statistical processing. From this information we constructed an input generator that creates tasks having the same statistical distributions as those of users in the real environment. Statistics from the simulation along with those of the actual system appear in Figure II.

A second method of comparing the model with the real system is to have humans go through a script and then compare actual responses with the simulation. The only advantage of this test over that of the preceding paragraph is that the properties of the user's tasks are known exactly and may be varied at will. Carnegie's 360 may also be driven by a simulated user environment in which the system responds to a completely repeatable script representing an arbitrary number of terminals following predetermined programs. This test was more demanding of system resources than the previous one. All users made similar demands on the system since they were following the same script. The comparison of the simulation and the actual system appears in Figure III.

An alternative to the previous tests is to probe the system periodically with test runs while it is in the normal production environment. The sample jobs for this test were the same as those used in test two, but all other users were doing normal work. We loaded the simulation with users requesting resources based upon our statistical profile of the general user. Into this

*See Kac[15] for a good discussion of the use of models in science.

environment we inserted a user requesting the resources of the test probe. Figure IV compares the average response times for the test run in both the model and TSS.

The results of these three experiments, plus the knowledge that we carefully modeled the details of the scheduling algorithm, give us confidence that we have captured the essential features of the scheduler and the environment in which it operates. The modular nature of the simulation makes extension to other areas of TSS design natural. In future studies of additional subsystems the present scheduler may be replaced by a simple priority discipline if goals change and a macro approach to scheduling is practical. An abstract model is applicable to a limited number of questions about the real system. It must change with the questions. At every stage a reasonable test of validity should confirm the assumptions of the analyst. Using a model in uncharted domains without validity checks, even though it may be well validated in some areas, is one of the most dangerous practices found in many simulation studies.

Present and Future Model Uses

The original goal of the model was to aid in tuning TSS/360 to the job load of the CMU environment. The first use of the simulation involved a group of users who proposed paying a higher rate to receive better service from the system. The proposal to achieve this service differential was to cause the high priority class to follow a different algorithm through the schedule table. Tests using the model indicated that benefits of the proposed solution were marginal compared to the increased charge. A different plan, based upon the concept of guaranteed terminal access to a certain number of higher priority users, provided the required service differential.

Late in the spring of 1970 Carnegie announced that starting with the fall term, TSS/360 would be the general purpose campus computing facility. The Computation Center would phase out OS/360 and restrict the 1108 to large scientific users. Previously TSS has served a limited user community. The transition of many users from OS/360 and 1108/EXEC II requires that new services be provided under TSS. An example of one such service is the introduction of a fast turnaround, student oriented, Fortran compiler called WATFIV. The model is presently being used to determine how best to schedule this service in an environment of conversational and batch tasks. It is also forecasting the number of users who can adequately be served in each category.

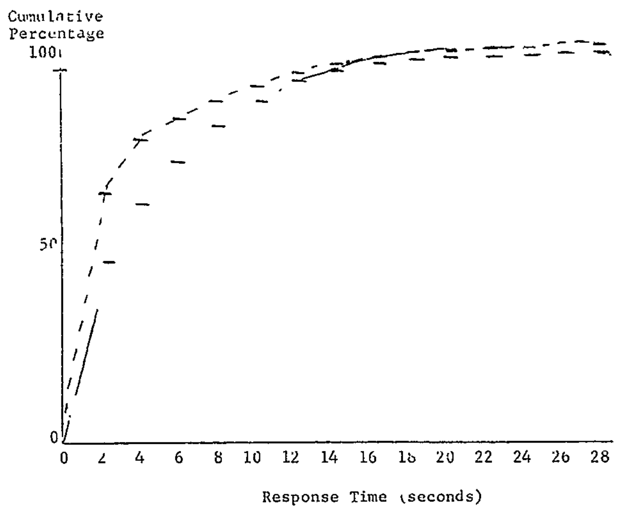
LCS, the slower core storage, causes the effective instruction rate of most user programs to be approximately one per 8 microseconds. In the fall of 1970 new LCS, with a two to three fold improvement in cycle time, will replace the old core. The modular nature of our system enables us to reconfigure the simulation to predict overall system improvement due to LCS upgrading. Figures V and VI are examples of the type of studies possible with minor alterations to the model. The high paging rates generated by heavy loads cause predicted improvement due to faster LCS to be less than expected. Faster core does not alter the paging rate, and tasks spend a significant portion of time waiting for pages. The model allows us to quantify the function relating core speed to response time. The effects of adding additional units of our present speed LCS are presented in Figure VI. The improvements are substantial, but the cost is large.

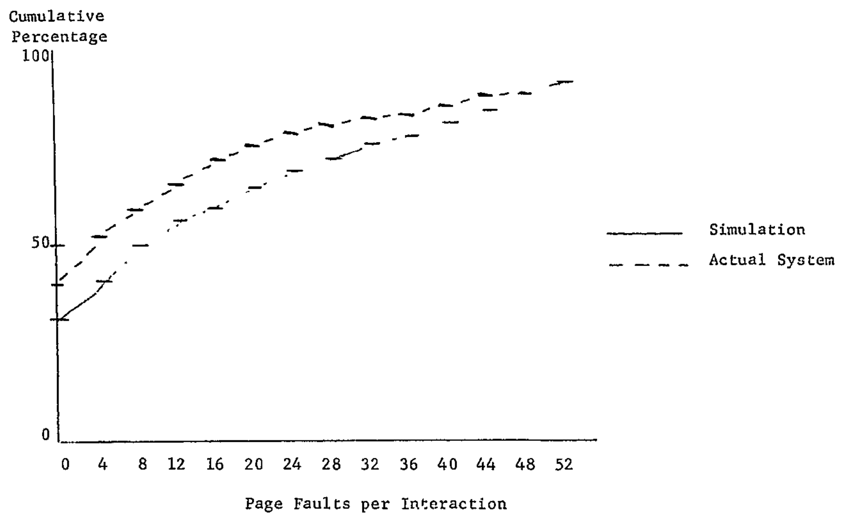
Costs of alternative configurations are straightforward to calculate, but the benefits are very hard to assess without a good model. It is not possible to validate our predictions until the new core is installed and statistics are gathered. If the model proves itself in this new domain, we feel that the simulation will be a valuable working tool for continued use in studies of both hardware and software subsystems of TSS/360.

This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (F44620-70-C-0170) and is monitored by the Air Force Office of Scientific Research.

References

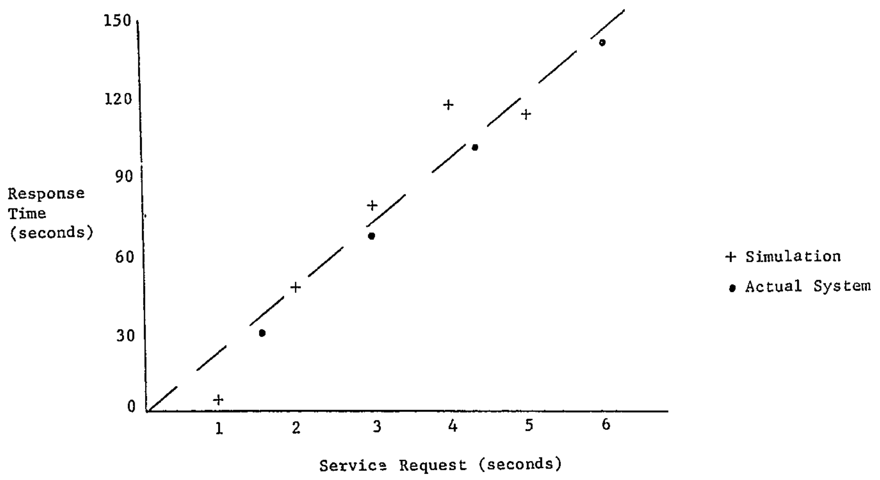
1. IBM Time-Sharing System - Concepts and Facilities, Form C28-2003.
2. Wilkes, M. V., Time-Sharing Computer Systems, American Elsevier, New York, 1968.
3. Fine, G. H., and P. V., McIsaac, "Simulation of a Time-Sharing System", Management Science, vol. 12, No. 6, Feb. 1966, p. B180-B194.
4. Scherr, A. L., An Analysis of Time-Shared Computer Systems, Research Monograph No. 56, M.I.T. Press, Cambridge, Massachusetts, 1967.
5. Katz, J. H., "Simulation of a Multiprocessor Computer System", Proceedings AFIPS 1966 SJCC, vol. 28, p. 127-139.
6. Katz, J. H., "An Experimental Model of System/360", Communications of the ACM, vol. 10, No. 11, Nov. 1967, p. 694-702.
7. Barker, P. E., and H. K., Watson, "Calibrating the Simulation Model of the IBM System/360 Time-Sharing System", Third Conference on the Applications of Simulation, 1969, p. 130-137.
8. Martin, W. L., "System Analysis Program - A Simulation Technique", Summer Computer Simulation Conference, 1970, p. 98-103.
9. Neilsen, N. R., "The Simulation of Time-Sharing Systems", Communications of the ACM, vol. 10, No. 7, July 1967, p. 397-412.
10. Neilsen, N. R., "ECSS: An Extendible Computer System Simulator", Third Conference on the Applications of Simulation, 1969, p. 114-129.
11. Bairstow, J. N., "A Review of Systems Evaluation Packages", Computer Decisions, vol. 2, No. 6, July 1970, p. 20.
12. Dahl, O., and K., Nygaard, "Simula - An Algol-Based Simulation Language", Communications of the ACM, vol. 9, No. 9, Sept. 1966, p. 671-678.
13. McCredie, J. W., "Structure of Discrete Event Simulation Languages", Summer Computer Simulation Conference, 1970, p. 88-97.
14. Denning, P., "The Working Set Model for Program Behavior", CACM, vol. 11, No. 5, 1968, p. 323-333.
15. Kac, M., "Some Mathematical Models in Science", Science, vol. 166, No. 3906, Nov. 7, 1969, p. 695-699.





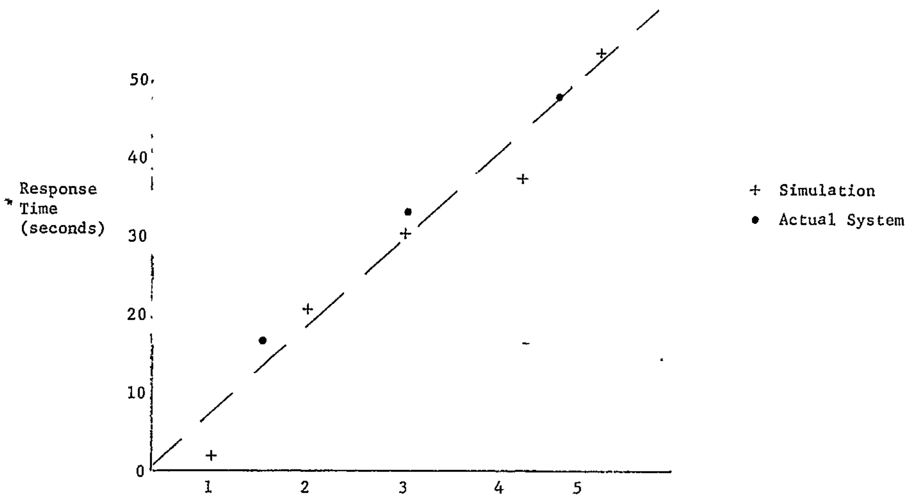
Normal User Mix - Model vs. Real System

FIGURE II



Average Response vs. Service Request - Model and Real System

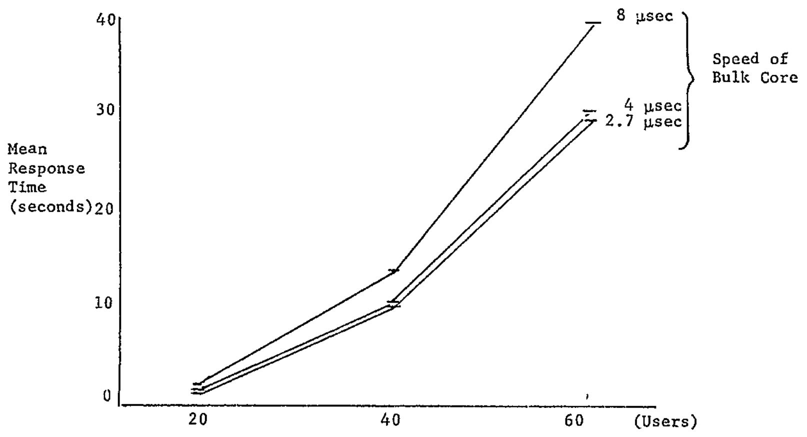
FIGURE III - HEAVY USAGE SCRIPT



Service Request (seconds)

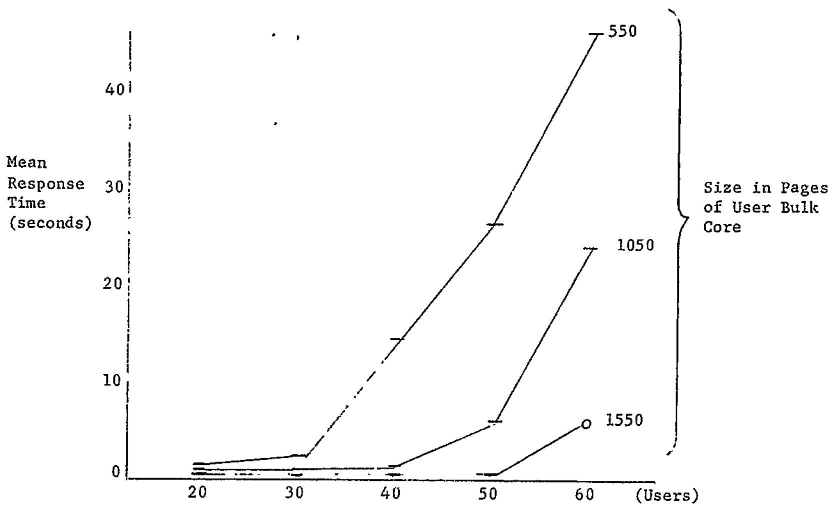
Average Response vs. Service Request - Model and Real System

FIGURE IV - PROBE JOB IN NORMAL MIX



Mean Response vs. Number of Users - Different Speeds of Bulk Core

FIGURE V



Mean Response vs. Number of Users - Different Seizes of Bulk Cor^

FIGURE VI