AN ADAPTIVE MICROSCHEDULER FOR A MULTIPROGRAMMED COMPUTER SYSTEM

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

by

E. M. Pass

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

In the School of Information and Computer Science

Georgia Institute of Technology

March, 1973

AN ADAPTIVE MICROSCHEDULER FOR A

MULTIPROGRAMMED COMPUTER SYSTEM

Approved:

Chairman: John M. Gwynn, Jr.

Member: Michael D. Kelly

Member: S. Paine Lenoir

Special External Reader: Stephen R. Kimbleton

Date approved by Chairman: March 27, 1973

ACKNOWLEDGMENTS

This dissertation could not have appeared in its current state without the assistance of the thesis advisor, Dr. John M. Gwynn, Jr., and the members of the reading committee, Dr. Michael D. Kelly, and Mr. S. Paine Lenoir, Jr. Recognition also should be given to the outside reader, Dr. Stephen R. Kimbleton, for his assistance. Dr. Norman R. Nielsen deserves commendation for his efforts in the field of computer systems simulation, upon which some of the current research is based. Dr. James W. Walker provided valuable information concerning statistical considerations relating to the experimental portions of this research. Appreciation is extended to Mrs. Linda Wix and Mrs. Shirley Isbell for assistance in the preparation of research drafts and final report, and thanks go to my wife, Kay, for clerical help in the preparation of research drafts and bibliography. This research was partially supported by the National Science Foundation through Grant GN-655.

TABLE OF CONTENTS

Table of Contents (continued)

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of this research is the scientific development and verification of a new, adaptive model of internal scheduling of resources, with the goal of the optimization of computer system performance. A general system effectiveness measure is defined which parametrically encompasses the prototypical system effectiveness measures to be considered. The adaptive internal scheduler then selects such tasks for resource allocation request fulfillment that a local system effectiveness measure, derived from the general measure, is optimized, leading to semi-optimization of the general measure. The adaptive scheduler functions as a second-order exponential estimator with trend detection. A predictor-corrector algorithm functions as the adaptive controller by varying the estimator's parameters and the time of application of the estimator in response to the nature of the sequence of deviations between the predicted and actual values of resource utilization. In order to validate the new scheduling model, a workload description in the form of task profile distributions was gathered by a software monitor on the Georgia Tech B5700 running a live job stream. A simulator was developed to allow the comparison of the new scheduler with other nonadaptive schedulers shown to be good by various researchers, under various general system effectiveness measure proto-

types. The simulator was validated by running it with the B5700 TSSMCP scheduler against the B5700 workload job profiles. Values resulting from the simulation checked against those of the measured B5700 system quite well. The results of other simulation runs show that the new adaptive scheduler is clearly statistically superior to other schedulers under most measures considered and is inferior to no other scheduler under any measure considered.

The other schedulers compared were the following:

B5700 TSSMCP
Round-Robin
First-in-First-out
Instantaneous Exponential Estimation
Complete History.

The prototypical system effectiveness measures considered were the following:

Throughput (tasks/unit time)
CPU Utilization (CPU)
Revenue (CPU plus 1/3 I/O)
Resource Usage (CPU plus I/O plus CORE)
I/O Utilization (I/O)
System Utilization (busy time/total time)
System Cost Nullification (3.14 CPU plus I/O plus 8 CORE)
Latency (measure of response time).

CHAPTER I

FOUNDATIONS

Objectives of Research

This research has as its primary objective the design of a microscheduler (an internal resource allocator) which will perform semi-optimally with respect to several measures of system effectiveness. Associated with this objective is the goal of the development of a means of comparing the proposed microscheduler and other prototypical microschedulers with respect to several system effectiveness measures. In order to accomplish this goal, simulation was used for the comparison process. Included in the research is an examination of existing prototypical microschedulers and measurement techniques.

Scope of Research

The framework of this research is based upon the Burroughs B5700 Time Sharing System, with which the author worked for several years at the Georgia Institute of Technology. In order to limit the research area, the external scheduling in terms of ordering of input tasks is not controllable by the proposed microscheduler. Previous research has shown that external scheduling is quite amenable to semi-optimal processing by pre-scheduling techniques (Grochow, 72A, Grochow, 72B,

Hoffman, 71). Thus the task stream is considered fixed and typical of the Georgia Tech B5700 workload. In order to facilitate the measurement of the environment and the simulation of the microschedulers and system effectiveness measures, only a one-CPU system configuration is considered, though the equipment at Georgia Tech includes two CPU's. Under the preceding conditions, simulation is used to compare microschedulers with respect to various measures of system effectiveness.

Need for Research

In a dynamic resource allocation environment, as is found in most modern digital computer systems, the microscheduler is the single most important software module in the portion of the operating system which controls the operations of the system. The scheduling of tasks in a multiprogrammed computer operating system environment is a major determining factor of the actual performance of a computer system. Proper selection tends to optimize the usage of the resources of the system and reduces the total average elapsed time each task requires to complete its assignment. Improper scheduling can so degrade the system that a multiprogramming environment behaves in a slower and less efficient manner than does a good, similar non-multiprogrammed computer system (Bowlden, 67, Hoffman, 71, Wulf, 69).

This degradation may occur in several areas. One, the most familiar to many users, concerns the overcommitment of resources to

tasks, causing page thrashing in the case of the main storage resource, for instance. Another concerns the vast overhead imposed by certain hardware/software configurations even on the simplest tasks, such as experienced by early implementations of TSS/360/67 which had reported 95 per cent overhead (Nielsen, 67). Still another area concerns performance deficiencies in which the scheduler wastes resources by not assigning them well or in a timely manner. Effort spent in improving the scheduler may thus produce disproportionally large improvements in the performance of a complete computer system. The intent of this research is to investigate a specific method by which the operations of internal schedulers may be improved.

The physicist James Clerk Maxwell used a device called a "demon" to illustrate certain thermodynamic concepts. One of the tasks considered as given to a demon was that of separating hot particles from cool ones by opening a door to allow hot particles to pass into another chamber, but closing it to keep cool particles out of the other chamber. Later physicists were able to prove through advanced thermodynamic arguments that more energy would be required in the decision process which distinguishes the hot particles from the cool ones than could be derived from the separation of the particles on that basis.

An analogous argument applies in the case of the proposed microscheduler. If the microscheduler is to succeed in its mission, its average operational cost absolutely must be less than the average gain in

productivity, with respect to some measure. Otherwise, it is as useless as a Maxwellian demon, perfectly capable of separating good tasks from bad ones, but with such overhead as to render its actions futile on a practical basis.

Translated into specifications for the microscheduler, the preceding comments entail the requirement that the scheduler must use less total weighted resources than it saves if it is to succeed. Thus there is an almost continuous set of tradeoffs between sophistication, speed, cost, and resource requirements of the microscheduler. The nature of this set of tradeoffs is so complex and time-and-case variant that the best which can be done, in general, is to perform the best suboptimization possible in specific cases and to hope that severe pathological cases will not occur or will be mollified by the improved microscheduler (Denning, 70, Graham, 72). This is the philosophy followed in the current research.

Purpose of Microschedulers

In a multiprogrammed environment the system's resources must be shared in a carefully controlled manner. The microscheduler provides the logical basis by which the sharing of resources is controlled. A good microscheduler attempts to keep the most costly resources of the system busy. Typically, microschedulers utilize little information available concerning the tasks in concurrent execution to perform this

control function when required. Many use FIFO, round-robin, priority, deadline time, and main storage requirements to help base their decisions. This can lead to quite nonoptimal patterns of resource allocation and may result in an elapsed time required to process a given set of tasks together longer than that required to process them serially. Several researchers have emphasized this point concerning B5700 operations (Block, 71, Bowlden, 67, Wulf, 69). The microscheduler of this research is designed in such a way that it will, based upon historical and current resource utilizations, attempt to provide a net gain in system effectiveness. This gain is made entirely through the improvement in the decision-making capability of the microscheduler, enabling it to construct better patterns of resource allocation. This assists in reducing the nonoptimal sequences of resource allocation and in avoiding pathological cases.

CHAPTER II

MICROSCHEDULER ANALYSIS

General

> After growing wildly for years, the field of computing now appears to be approaching its infancy. Recent revolutionary technological advances will eventually take us far beyond our newest, biggest, and best computers. Yet computers and computing have already fantastically increased our power to know as well as to do (Hornig, 67).

That statement, made several years ago, could analogously be made today concerning the field of computer performance analysis. The field in the past and into the present has been composed of a jumble of techniques ranging from highly practical and empirical to highly theoretical and esoteric, with little general structure to guide the worker with a real problem (Bagley, 60).

Within the last few years, some real effort has been made to categorize the methods and techniques of the field (Bell, 72). Also in the last few years, emphasis has shifted from a hardware-only performance analysis to a system-oriented performance analysis. The succeeding sections describe the various relevant aspects of performance analysis.

Performance Evaluation

Performance evaluation may be subdivided into two areas, performance projection and performance monitoring. Performance projection refers to the estimation of the performance of a system which does not yet exist or is not available for actual monitoring of performance. The technique may also be used to consider cases which may not be constructed at will, such as overloads on a time-sharing or real-time system, or envisioned but as yet unrealized workloads. Performance monitoring refers to the gathering of data on an existing system. It may take one of several forms, but is normally implemented through hardware or software monitors in the system or through extremely detailed simulation. Even in the case of simulation, hardware or software monitoring is usually needed to derive the necessary statistics with which to accurately base the simulation.

Performance Monitors

Instrumentation in the form of hardware and software monitors provides the means by which a running system may be analyzed to a desired level of detail (Rodriguez-Rosell, 72). They are quite often used to detect performance inadequacies in hardware and/or software configurations and thus provide the human with the knowledge required to reduce such inadequacies (Schwartz, 68, Walter, 67). Hardware monitors have the advantage that they may be attached to the system to be measured without affecting the internal operation of the system at all

(Amiot, 72, Apple, 65, Arndt, 72, Bonnor, 69, Schulman, 67, Shemer, 72, Stang, 69). Most commercial monitors have free-standing tape drives on which to record the data stolen from the system, and they have some number of independent input channels which may be logically OR'ed, AND'ed, NOR'ed, NAND'ed, etc., together to construct signals which may be led to sum units, the contents of which are periodically placed onto the attached tape. Also most commercial monitors are accompanied by a FORTRAN analysis program which can translate the masses of data into human-readable and interpretable form. Hardware monitors have the disadvantage that most of them are incapable of monitoring information the location of which dynamically varies, as in the case of the B5700; this is a major advantage of software monitors (Bemer, 68). Other advantages of software monitors include the generally low cost to install and the tremendous flexibility of investigation (Keefe, 68). The major disadvantage of a software monitor is a direct consequence of the uncertainty principle: the closer a system attempts to measure its own actions, the more disruptive the measurement process is to the system. However, by clever design the degradation of a software monitor may be kept very low and measurable so that its effects may be almost cancelled in the analysis phase (Arden, 69, Bookman, 72, Stanley, 69). A software monitor was developed for and used in this research and is described in a later chapter of this report. The major problems with hardware and software monitors are the storage

analysis of large volumes of data and the reduction of that data into the information required by the researcher. The problems are difficult, but the rewards may be substantial (Baer, 72, Bonnor, 69, Campbell, 68, Cantrell, 68, Feeley, 72, Yeh, 72).

Simulation

Simulation is the most general and powerful of the methods thus devised for the analysis of computer systems. It is probably the most difficult to develop and use properly, but it is potentially far superior to any of the other means, and the possible rewards for its application may be the overriding influence in the decision to use it (Fine, 66, Huesmann, 67). In the case of a system which does not yet exist, simulation may very well be the only manner in which to analyze it accurately. These are the reasons by which simulation was chosen for this current research.

Huesmann asserts, probably correctly, that the state of the art will not currently support a comprehensive simulator capable of simulating any proposed computer system configuration by means of parametric or algorithmic description (Huesmann, 67). This is not to say, however, that entire series or classes of computer systems cannot currently be handled adequately; for instance, Seaman and Soucy developed a simulator for IBM S/360/370 in a language specifically designed for the purpose, CSS (Seaman, 69). The special commercial packages which purport to analyze any existing hardware/software combination,

such as CASE, SAM, SCERT, etc., are in actuality not system simulators but manipulate data taken from tables which empirically describe the behavior of certain configurations of the systems in order to attempt to describe the specific situation (Huesmann, 67).

The CASE and SCERT system simulators are the most widely used commercial packages for the evaluation of standard, existing computer systems (Bairstow, 68, Canning, 68, Comress, 67, Herman, 67, Ihrer, 67, Pomerantz, 70, SPC, 68, Thompson, 68). They are very controversial as to the accuracy of their predictions, the generality of their application, and the cost/performance of the method. Many clients have purchased the use of the systems for some substantial total amount of time, however, and the methods cannot be ignored.

In coordination with other techniques, simulation may be used to model computer systems quite accurately and flexibly (Canning, 68). For existing systems, simulation models may be advanced considerably through the use of information gained by the use of hardware and software monitors. The monitor output may be used almost without further analysis, as in the case of trace-driven modeling (Cheng, 69). It may also be used after much analysis, as in the form of stereotypical job descriptions (Nielsen, 67). If analytic modeling techniques are not feasible, mathematical models may be used to derive information about system performance by following the changes of state resulting from the succession of events affecting the system. This is accomplished by us-

ing numerical techniques to follow the corresponding changes in the mathematical model (Blake, 64).

The simulation of this research is not of the all-encompassing type, such as that discussed under general systems research topics. An extreme example of this is the preparation of a three-eon simulation of the universe reportedly given as an examination question at several technical institutes. In this class of simulation, the entire relevant universe is considered in the model. It is reticulated into interacting submodels, and each of them is similarly decomposed, as necessary, until each terminal model may be analyzed through simulation. Although each simulation has an external environment, the supersimulation has no external environment and no exogenous functions not under control of the simulation. Thus, for the purposes of the current research, all of the system except for the scheduler internal resource allocation is externally described parametrically or algorithmically to the model.

The basis of the class of simulations discussed here is the concept of time. There are two extremes in the methods of maintaining time in simulation models. It is impossible in a digital computer to truly represent a continuous variable such as time since a continuous variable would require infinite precision. Thus a discrete representation of time must be used. In this case, the state of the model is examined at each time interval endpoint. This has advantages for model-

ing continuous variables but is quite inefficient for slowly changing models and is quite inaccurate in many cases.

The other alternative in time-keeping is event-oriented simulation. In this case, discrete time interval representation is used, but the simulated time is advanced to the next event in the model at each stage. This will, in general, produce nonuniform time steps, as opposed to uniform time steps of pure discrete time interval representation. In practice, this is not a severe problem since simple weighting factors can be easily applied to represent the nonuniform time steps in event-oriented simulation.

The simulation model must be capable of logically handling truly parallel events since the model itself executes serially, while the system being simulated in this case does not. Also, the model will not run properly without comprehensive descriptions of exactly what system is being simulated. It must also be capable of presenting its actions in a meaningful format for interfacing its information with the researchers performing the interpretation of the simulation.

Of course, the computer system model cannot be considered deterministic. Realistically, it must be considered stochastic. Many apparent operations of the computer system being modeled are probabilistic in nature to some extent. It is the responsibility of the modeler to specify the natures of the parameters.

To handle probabilistic variables, almost all simulations use

streams of pseudo-random numbers. A variable with almost any probabilistic distribution may be generated by this method. The random number stream is used as domain values and the desired probabilistic values may be derived from the range of a properly chosen function. Pseudo-random numbers are normally generated by additive-multiplicative-modulo operations. An excellent discussion may be found in Knuth (Knuth, 69). Pseudo-random number generation is fast, simple, conservative of storage, reproducible, and generators may be constructed to fit given tests of statistical randomness.

Thus far the simulation itself has been discussed. The representation of tasks in the simulation must be considered carefully. This representation must specify to some desired level of detail the reaction of the system to every class of stimuli in the simulation. The nature of representation may range from constant parameters to algorithmic specification, but in the simulation of this research, it takes the form of sets of probability distributions. This has the disadvantage that it must be dynamic--as the load increases and time progresses, various distributions will change drastically, for instance. This problem is overcome by providing for a series of different distributions allowing for a limited number of different conditions. The number and size of these tables would possibly become prohibitive as the simulation is brought very close to the real-world system.

This is avoided by the use of stereotypical task-mix configura-

tions, since a set of job descriptions describing subclasses of task types could be shared by the tasks in simulated execution. Utilization of the stereotypical approach should allow for simpler simulator construction and faster execution. The purpose of simulation is usually the modeling of a real-world system, not the programming of a complicated simulator.

There exist cases when considerable sophistication must be inserted into the design of the model in order to depict subcomponents of the real-world system more accurately. This is in conflict with the tendency to make the model as simple as possible. A tradeoff must be made between closeness to real-world system and simplicity in the model. The extremes of this tradeoff are triviality and emulation. Somewhere between the extremes lies the proper mixture, as was decided in this study in the simulator.

This research is not concerned with specific hardware/software configurations as such. This fact simplifies the simulator in the areas not concerned directly with the microscheduler subcomponent under consideration. It allows the parametric specification of the other algorithms of the operating system. Otherwise, the model might be unnecessarily complicated in the wrong areas.

The other serious concern involves the procurement of accurate data with which to drive the simulation, especially for the verification phase. A fairly recent development in this area concerns the use of

trace-driven modeling (Cheng, 69). This involves obtaining actual data through the use of instrumentation, summarizing it, and using it in a model of some components of the computer system. This method is powerful but suffers from the same problems as benchmarking--with careful selection of jobs to run, practically any desired distribution of parameters may be obtained. However, with scientific selection, representative distributions may be obtained.

As might be expected, simulation has played a major role in some of the most important analyses of computer operating system performance evaluation. One of the major early time-sharing simulators was developed by Nielsen of Stanford to model the IBM S/360-67 Time-Sharing System (Nielsen, 67). This simulation model is an aid in the IBM S/360/67 hardware/software configuration problem. Thus it is designed to facilitate system configuration changes. This requires his simulator to be flexible enough to handle multiple CPU's, variable sizes and speeds of core memory, variable numbers and speeds of peripheral devices, and even other computers coupled to the one under study. The model is also meant to test changes in proposed software algorithms. Like the configuration parameters, the algorithms are relatively detached from the main body of the simulator for ease of change. The model is not system-inclusive, as it does not consider operators nor users, except parametrically. It is not truly general, for it only handles hardware/software configurations with fixed time quanta devoted

to individual time-slices and fixed page segments devoted to programs and the operating system. The only documented application of that simulator was begun on the IBM S/360/67 time-sharing system one year before the hardware was delivered and two years before workable software (TSS/360) was ready, and it was shown to be quite accurate in comparison with the delivered system. This simulator was used briefly in the current research to investigate core allocation strategies in a paged environment. It was not used as a major simulator because of the difficulty in changing it to do the general resource allocation schemata analysis, especially in a nonpaged case.

Another simulator was developed by Nielsen for the analysis of the effectiveness of certain techniques such as program relocation, disk rotational delay optimization, and swap volume minimization on a proposed time-sharing system for the Burroughs B6700 (Nielsen, 69, Nielsen, 70, Nielsen, 71). Since the B6700 system is based on variable-sized pages and a dynamic-overlay-based core allocation scheme, he is able to present data which provides an indication of the effectiveness of that type of organization as used in a time-sharing environment. He also describes the operational capabilities of the simulation model. Because of the similarities between his model's capabilities and those required in the current research, the structure of his B6700 Time-Sharing Simulator was used as a basis for the major simulator used here in the validation phase. The simulator used in this research is

thus similar to that used by Nielsen in his work for Burroughs in terms of input and output to and from the simulator and in terms of some internal operations; nevertheless, that simulator used in this research differs significantly from Nielsen in terms of capabilities and structure.

Despite these advantages of simulation for modeling computer operating systems, the number of simulation failures has become legion. The problems are concerned with the information loss in the transferral of the structure and data of the system to that of the model. Most of the simulators constructed have been based upon probability distributions of various parameters of interest and extreme care is required to prevent gross loss of structural information in the construction of the model. Nielsen's simulators use sets of statistically-defined task profiles, a method which has proven as good as any other means and better than most others. Another problem concerns the difficulty of modeling complex algorithms embodied in modern computer operating systems.

Because of these problems, the Government Accounting Office recently banned the use of simulation as a sole means of comparing hardware and software systems for procurement analysis (GSA, 72, Highland, 72, Ihrer, 72, Lundell, 72). Despite these problems of simulation programs which have not been properly designed, executed, or analyzed, simulation as a method is still very much a good technique for the analysis of computer operating systems when properly handled.

Structural Performance Analyses

Studies

The preceding section of this research discussed the methods by which performance evaluation may be performed. This section discusses the actual work which has been performed in the field with respect to the current research (Bookman, 72, Bradley, 68). A considerable number of researchers in the field have indicated in the literature that significant increases in computer performance may be realized through dynamic calculation of resource allocation priorities. Many of the techniques attempt to give dynamically higher resource priorities to those tasks which use less of the resource. Thus an I/O-bound task receives a higher CPU priority than a CPU-bound task, etc. The techniques generally assume that recent past performance of a task is a good indicator of recent future performance. Few of the studies reported a systematic effort to analyze or to simulate specific components of the resource allocation process, with Nielsen the major exception. Almost all practical researches involved alterations in the system under study independently of the improvement in the internal resource allocation scheduler (microscheduler), thus clouding the issue of actual improvement due to the microscheduler. No researchers reported a sustained scientific effort to gather workload data, statistically process it, model the computer system and computer operating system under which the workload was processed, and actually compare the operations of the

simulated system under proposed changes with respect to several different system objective measures.

Scherr developed a simplified system-inclusive analytical model of the MIT Project MAC's CTSS (Compatible Time-Sharing System) (Scherr, 67). Although his approach was coarse, it reflected the behavior of the actual system very well, primarily because CTSS is a simply-designed operating system with certain salient attributes which, almost by accident, happen to be very convenient to model (Corbató, 62). Foremost among these is a basic round-robin CPU allocation scheme, a simple core-swapping technique, and a method of polling terminal input which produces an (artificial) negative exponential distribution of inter-arrival times. He demonstrated quite well that an analytical model actually can be general and accurate also.

Stevens used the running average of

(total CPU time divided by total I/O channel time)

as the internal priority for CPU microscheduling (Stevens, 68A). He suggested this as an alternative to artificially assigning high CPU priority to I/O-bound tasks to avoid unnecessary delays in such cases. This is sometimes known as the complete history method. He noted an increase of 18 per cent in CPU utilization and of 100 per cent in throughput. The number of other modifications reportedly made simultaneously to the operating system under study makes it difficult to attribute this increase in performance to improved CPU allocation strategy only.

Rehmann and Gangwere report on the simulation study of resource management in a time-sharing system using a two-queue-level foreground-background CPU scheduling model (Rehmann, 68). They used response time as the performance measure, since they consider it to be a natural user-oriented manner of measuring the performance of the system. The system studied is nonpaged, multiprogramming and uses a core-swapping technique. They note that Cantrell was the first to propose using a time slice to separate long tasks and short ones (Cantrell, 67). His idea was to set the length of a time slice so that 90 per cent of all tasks in the expected workload would complete in the first time slice awarded them. If a task did not complete in the first slice, it was moved to a lower priority level so that longer-running tasks would receive CPU attention during the I/O activity of short tasks, or when no new tasks are available.

Their simulator for this situation was written in GPSS/360 and hand-verified. The model was heavily parametricized, and they were able to treat a large number of cases quite easily. They assume their workload distribution for CPU usage is qualitatively the same as that used by Cantrell and Scherr up to 90 per cent CPU saturation (Cantrell, 67, Scherr, 65). The last 10 per cent of the workload distributions were chosen to be theoretically representative of light, medium, and heavy CPU workloads. This choice was crucial to the study because the model is extremely sensitive to the nature of CPU workload in the

highest 10 per cent of the CPU saturation distribution. If their assumptions were not excessive and their input was accurate, their results for their CPU scheduling model should be very close to reality, assuming the model were actually implemented as stated.

Scherr reported on the construction and application of a simulation program designed to study the main storage fragmentation, run time dilation, throughput, turnaround time, job delay, and relocation effects of the task dispatching strategies embodied in an early version of OS/MVT running on an IBM S/360/65 (Scherr, 66B). His model simulates a variable number of job streams, each emitting tasks to the task dispatcher at statistically defined intervals. The job stream attributes change as the corresponding tasks are processed, as per each prototypical job step type of description. Each job step is described by the three following distributions: core storage space required, CPU time required, and minimum wait factor allowed. The wait factor is the percentage of the elapsed time that the job step would be in the wait state if run serially. The job step priorities increase as the core occupancy time increases. Running times for all except the job step with the highest priority are multiplied by the appropriate expansion factors Scherr derived from Markov analysis of the N-queue-level system he constructed. He constructed an input task stream taken to be representative of a workload consisting of FORTRAN, COBOL, and sorts. The mix stemming from the task classes contained 21 jobs containing 97

job steps. Average job step execution time was taken to be 62.8 seconds, and the wait factor was set at 65 per cent. These figures and the actual task mix chosen were derived from random selection of actual running tasks on the IBM S/360/65 operating under OS/MVT. Scherr concluded with the following points, among others:

> system throughput was unaffected by storage fragmentation, though task completion order was heavily affected by it;
>
> no tasks with very large core requirements were delayed indefinitely, even though his core scheduler favored the smaller tasks;
>
> dynamic relocation did not appreciably increase throughput;
>
> other job streams were run for verification, and most results were quite constant--for instance, maximum main storage utilization rarely left the 80-90 per cent range.

Lehman and Rosenfeld, and in another study Lassettre and Scherr, extended Scherr's earlier study into several areas including that of OS/TSO (Lassettre, 72, Lehman, 68). They fixed execution time of all tasks for several runs in order to permit analysis in more detail of core scheduling. This immediately produced an increase in throughput of 19 per cent because of the synchronous nature of the arrivals and departures of tasks, but no other major apparent effects. They attempted increasing the wait factor to 90 per cent for some cases with no major effects on the results except to allow the experimentation of more tasks receiving CPU attention in the same time interval. Scherr assumed that initial transients were unimportant to final results; they

discarded the statistical figure resulting from the execution of the first 5000 job steps and used the data resulting from the next 10000 job steps, a method suggested by Nielsen (Nielsen, 67). Their results indicate that the system never reached steady-state but did exhibit cyclical behavior, which they were able to explain. They explored the relationships between throughput and main storage capacity. As expected, as more main storage is available, throughput increases; however, the rate of increase decreases as more main storage is added by the law of diminishing returns, with a maximum throughput dependent upon CPU speed, which agreed with empirical data quite well. Turnaround time exhibited similar characteristics with respect to main storage size except, of course, that turnaround time decreased to a limit close to the empirical result as more main storage was available. They repeated Scherr's experiments concerning dynamic relocation with comparatively extremely large core requirements in each of the job steps. Dynamic relocation in this case produced substantially better maximum total job step delay times; though, as more main storage was added, the difference decreased quickly and eventually disappeared.

Marshall proposed two improvements in CPU allocation scheduling over that present in S/360/370/OS/MVT which improved throughput (Marshall, 69). First was a reward-penalty system which rewarded a task with a larger quantum of CPU time if it generated an I/O request during the last time quantum but with a smaller one if it did not. Second

was the use of the cumulative ratio of

(total elapsed time minus total CPU time)

divided by (total elapsed time)

as a component of a dynamic CPU task priority computation scheme. He found the first improvement ineffective, probably because the tasks were unable to use the additional CPU time, but the second highly effective, in his environment. He also simultaneously restricted maximum total consecutive quantum time to five seconds per task, which probably had as much effect (or more) as his dynamic priority calculation on computer system performance, as suggested by Baskett (Baskett, 70).

Wulf studied dynamic CPU and core scheduling on the B5700 and proposed an exponentially smoothed average of recent CPU instantaneous times as a CPU usage predictor. He noted that the use of this predictor as part of a dynamic CPU priority adjustment, along with the suspension of tasks which are apparently causing page thrashing, increased CPU utilization by 20-25 per cent and eliminated page thrashing as a problem. It is not clear what portion of the improvement was due to each modification made, nor that the actual changes made were exactly those specified, as no validation processes were performed as such. Wulf does indicate that resource allocation algorithms should be based on system performance measures, such as those used in the current research, and not solely upon attributes of individual tasks or the

system as a whole in order to attempt to optimize system operations as a whole, with respect to some preselected measure function (Wulf, 69).

Several authors have studied the use of round-robin scheduling techniques for certain restricted environments by comparing certain system measure values corresponding to varying quantum sizes. Several of these, including Baskett, have used queueing theory to predict that round-robin scheduling will produce the best throughput for the system when the CPU usage distribution is highly skewed toward less usage and the types of tasks being processed are unlike (Baskett, 70). Such information is interesting here as a comparative data source.

Several studies by Sherman, et al., have as their goal the comparison of various microschedulers proposed in the literature under an environment of information drawn from a CDC 6600 through a software probe with the use of trace-driven modeling, as suggested by Cheng (Cheng, 69, Schwetman, 70, Schwetman, 72, Sherman, 71, Sherman, 72). Using a preemptive scheduling policy and selecting the task expected to compute the longest before voluntarily giving up the CPU, they were able to derive what they considered was the worst case. Then, by defining a throughput measure as

((time for worst method) minus (time for subject method))

divided by (worst method's time),

they were able to compare the methods with respect to throughput increase. A summary of their results appears as follows:

Method	Improvement
Best (preemptive)	12.89 per cent
Exponential smoothing (512 ms bound)	10.93-11.53 per cent depending on alpha
Round-robin (8 ms quantum)	10.08 per cent
Complete history (512 ms bound)	9.63 per cent
Random guess (512 ms bound)	5.92 per cent
Best (non-preemptive)	3.06 per cent
FIFO	0.78 per cent
Worst (non-preemptive)	0.20 per cent
Worst (preemptive)	0.00 per cent

Cheng suggests that the trace-driven approach could assist in the synthesis of accurate predictive models through successive calibration steps (Cheng, 69). At each stage, the actions of the model could be compared with the measured results. Furthermore, if the real system were available for modifications, improvements could be made in it, and the cycle could be repeated. However, few such experiments have been reported in the literature, including Sherman's work.

Batson studied certain aspects of main and mass storage allocation and usage on the Burroughs B5700 (Batson, 70, Batson, 71). In one study, he used a software monitor to determine the distribution of main storage segment size under a normal workload. The most salient feature of his results is the large number of small segments (60 per cent of the segments in use contain less than 40 words). Another important result of his results is a refutation of the assumptions made by other researchers concerning the negative exponential distribution of segment sizes under a variable segment size system (Luderer, 72,

Randell, 69). In another study, he used a similar software monitor to study the changes in throughput associated with the support of a virtual memory system. He developed a set of standard benchmark programs and ran various combinations of them to derive various timing for (virtual memory load) versus (time to complete the set of jobs), using serial job processing times as a basis for the computation. He refers to Randell's study in which he notes that, under assumptions of negative exponential distribution, main storage fragmentation due to rounding of requests up to the next fixed-page size boundary is usually worse than that due to checkerboarding encountered in variable-paged systems (Randell, 69). If so, the fragmentation due to rounding may be very bad in some cases, for Batson and the current researcher found very bad checkerboarding in the B5700 main storage during their investigations. This class of study may become even more important, with the advent of large numbers of virtual memory computers, such as the IBM S/370 (IBM, 72).

Ryder proposed a method using the information gathered in the previous time quantum as a predictor on which he based the reordering of a queue of tasks waiting for the use of the CPU (Ryder, 70). This formed an exponentially-smoothed, priority-based CPU allocation scheme. As in other studies, tasks with a recent history of being I/O-bound are given preference in the use of the CPU, in order to maximize throughput. His algorithm was effectively heuristic and based upon six

task characteristics, including Marshall's scheme (Marshall, 69). This microscheduler adjusted its actions to some extent in an attempt to make better judgments and predictions of task and system behavior. It is not possible to segregate the improvements embodied in his algorithm into their individual effects on the system. He made some effort to increase CPU-I/O-overlap time, a subject of much interest in many of the other works discussed here, including Baskett and Schwetman (Baskett, 70, Schwetman, 70). He did measure the times associated with the system's use of the microscheduler in a set of benchmarks, and recorded results under the headings of run time, idle time, CPU time, and CPU-I/O-overlap time.

He noted that, for various benchmark environments composed of predominantly scientific, predominantly commercial, and mixed tasks, the microscheduler performed quite differently in each case. In fact, for his commercial benchmark on the S/360/65, the total run time increased by 1 per cent with his algorithm compared to run time without it. In almost all cases, idle time was decreased significantly, while run time was decreased little or none, indicating that his microscheduler was requiring much CPU attention compared to the older one. In most cases, CPU-I/O-overlap time was increased significantly, due in part to the increased CPU requirements partially induced by the microscheduler's overhead, as shown in his results.

Bernstein and Sharp report on a CPU microscheduler driven by

a policy function (Bernstein, 71). They note that the resource services received by a task may be characterized by the weighted sum of units of service of each resource in the system. Each class of tasks is characterized by a different vector of weights. They suggest that task priority should depend upon the difference between the resource service represented by the specific vector of weights for the task class and the service the task is actually receiving. They propose a microscheduling technique which attempts to maintain the weighted sum of resources for a task above that specified for its class at that type of execution.

The technique was implemented in an undesignated computer operating system and was compared with round-robin philosophy of CPU scheduling. For this case, the ratio of (average elapsed time) to (average CPU time) for batch tasks was improved from a range of (15 to 20) to a range of (10 to 15). Response time for interactive tasks was improved as per the following table:

microscheduler scheme	(response time in seconds) min.	avg.	max.
round-robin	4.5	10.8	102.4
policy-driven	0.5	1.7	3.8

The CPU utilization

(total CPU time) divided by (total elapsed time)

was improved by about 20 per cent under similar workload.

Several other changes were made in the system at the same time, seriously tainting the results. The time quantum was shortened

[illegible] process shown by other researchers

to 64 milliseconds, a process shown by other researchers to be highly effective (Marshall, 69, Baskett, 70). The round-robin microscheduler with a large time quantum was shown by others to be a rather poor microscheduler, so the great improvement is not surprising (Sherman, 71).

The concept of the policy function is remotely similar to that of the local systems effectiveness measures used in the current research, as described in the next chapter. Lacking in Bernstein's study is the concept of prediction of resource usage for the next time interval. The current research goes much beyond his study in that area. Their method could be considered a special, nonadaptive case of the method proposed in the current research, considering one class of scheduling philosophy and one measure.

Wilkes presents a technique for workload adjustment in time-sharing systems in terms of attempting to predict the maximum number of tasks allowable to maintain control of the system in the next small time interval (Wilkes, 71). He also suggests varying the number of users of the system (as represented by the number of concurrent tasks) to attempt to limit the number of tasks to that predicted earlier. A severe problem with this approach is system stability in terms of oscillation of the number of active tasks with respect to time. His paper illustrates a method by which a system may control its own actions through the scheduling of tasks. Although the method of control seems

unnatural, it is important as an example of a microscheduler somewhat related to that proposed in the current research. In an earlier work, he presented a model of main storage allocation in a time-sharing system which attempts to prevent main storage overloads in a manner similar to that presented to prevent system overloads (Wilkes, 69).

Blatny, et al., discussed a limited approach to the optimization of a time-sharing system using simulation (Blatny, 72). They considered only the measures of mean cost of delay and mean system cost per task, as suggested by others (Greenberger, 66, Rasch, 70). They used a simulation approach because the nonlinear cost curves arising from the measures and the existence of a finite main storage have been shown to pose quite awkward problems for analytical models. The only resources considered for optimization were a single CPU, a paged, finite main storage, and an infinite mass storage. Unfortunately for the generality of their results, they initially assumed that the main storage required is a truncated normal distribution with mean and standard deviation taken as multiples of negative exponential functions of CPU time. A multiple-priority-level round-robin feedback queueing discipline is assumed used by the CPU scheduler. The simulations were concerned with considering various cost curves, minimum time slice times, round-robin cycle times, maximum number of tasks allowed in concurrent execution, intra-process operating system overhead, size

of main storage, and the speed of the CPU. Their results agree quite well with Greenberger's and Kleinrock's analytical results for similar situations. They also noted that improvements in systems performance with respect to the measures being considered can be obtained by the use of variable time-slice techniques and by the selection of optimal round-robin cycle times.

Denning and Eisenstein surveyed and discussed statistical methods useful in computer system performance evaluation (Denning, 71B, Denning, 72). Their paper and his book review the principles and definitions which underlie estimation theory and develop the theory for application to the problems of the performance evaluation of resource scheduling, in terms of the estimation processes used in predictive microschedulers. The purpose of estimation theory as used in predictive microschedulers is to use historical information to predict resource utilization and use such predictions so as to assign the resources according to some scheme, such as least predicted utilization. If the information thus derived is not used to schedule resource request fulfillment, the algorithm is not a predictive microscheduler, but is only an estimator. Thus an estimator could be considered as being capable of driving a microscheduler. The authors of that paper analyze several sequential estimators in the form of stochastic approximators and compare them in terms of each algorithm's ability to determine the value of slowly changing signals in the presence of noisy measurements.

They note that present theory is far from complete, has serious shortcomings, but that actual present applications, such as in Denning's working set concepts, are advancing the theory and practice of the area (Alderson, 71, Anacker, 67, Belady, 69A, Belady, 69B, Brawn, 68, De Meis, 69, Denning, 68A, Denning, 68B, Denning, 68C, Denning, 68D, Denning, 69A, Denning, 69B, Denning, 70, Denning, 71B, Hatfield, 72, Margolin, 72, Morris, 72).

The simplest estimator, from a conceptual standpoint, is probably the mean. In this case all that is required is the size of the subset and the values of each of the elements of the subset. However, it is computationally inefficient and requires a large amount of storage in the case of post facto analysis. It is a poor estimator in terms of slowness to change in online analysis as new measurements are made and incorporated into the computation. This introduces the concept of responsive estimators, which are those which discount the effect of early observations on present estimations. One of the simplest responsive estimators is the exponential estimator. It has been used in many cases for such tasks as inventory control. It is parameter-driven and weighs less-recent measurements exponentially less than more-recent ones. If carefully controlled, this parameter can be used to make the method quite powerful, since it can be dynamically varied to make the method more or less responsive according to an indication of how well it is estimating future observations. If this parameter must remain fixed,

the method can give quite bad estimations, since, if it is responsive, it is quite sensitive to noise in the measurements, and if it is to be made less sensitive to noise, its responsiveness suffers.

Another estimator, suggested originally by Eisenstein, is called the learning or stochastic estimator (Eisenstein, 70). In this method, each new estimator is the sum of the previous estimate and the weighted difference between the previous estimate and the previous observed value. The weights are exponentially-smoothed differences of previous estimates and corresponding observed values, functioning as reward-penalty pairs. This estimator is designed to attempt to damp the oscillations in the differences between the estimated and the observed values. It has the same problems as the normal exponential estimation method, though, in certain cases, is reported to perform quite well. Partially upon his work, the decision was made to use second-order, rather than first-order, exponential estimation.

Denning suggests still another type of estimator, the moving window estimator, which he used in his working set model (Denning, 68A). In this method, all observations except the last T are discarded, and the estimate is taken to be the mean of the last T observations. T can be parametrically set, just as can the parameter of the exponential estimator, to make the estimator more (or less) responsive and less (or more) stable with respect to noise. Denning also discussed the problems of the construction of optimal estimators for given situations,

decision-making based upon estimators, and the cost-performance analysis of improving estimators.

It is upon the basis of the concept of estimation that the adaptive microscheduler of this research is founded. Based upon recently measured values of task and system parameters, the method estimates near-future values of these parameters. The next sections survey existing practical and prototypical microschedulers, some of which employ estimation techniques.

Special Cases

Special cases of internal resource allocation schedulers may be found by consulting the documentation describing actual implementation of existing microschedulers in current computer operating systems. These are of interest as describing the state of the art at the time they were constructed and as offering suggestions for the forms of the microschedulers and system effectiveness measures in this research.

The Burroughs B5700 Data Communications MCP (non-time-sharing) System microscheduler is best described as having a priority-based, nonadaptive, and partially round-robin CPU microscheduler (Burroughs, 71, Canning, 66). The CPU is always given to the task with the highest priority which is capable of continuing. If the CPU is taken from a task because of an external interrupt, its position in the task queue is left unchanged. If the CPU is voluntarily surrendered, its new position in the task queue is behind all others with the same

priority. Normally, all user tasks enter the mix with priority five, though the programmer can specify a different priority through control cards and the operator can modify task priority dynamically. All operating system tasks have priority zero, which is best. See also the discussion of Wulf's work concerning B5700 CPU allocation algorithms earlier in this chapter (Wulf, 69).

Main storage allocation is performed dynamically on demand. If sufficient main storage is not available for a task to continue, an attempt is made to overlay any overlayable storage which is of sufficient size for the request. If enough main storage is still not found, the task remains suspended until such time as enough main storage can be found for its request. In certain pathological cases, this may cause quite severe system degradation. Normal main storage allocation is FIFO; however, main storage allocation in a no-memory situation is priority-based. See the discussion of Batson's work concerning B5700 main storage allocation algorithms elsewhere in this current research (Batson, 69).

I/O channel allocation is FIFO in all cases. If enough channels are not available, the request is immediately fulfilled. If enough channels are not available, requests are queued for each device type on a FIFO basis. This method of I/O channel allocation is used by virtually all operating systems described in this section.

The internal resource scheduling techniques used in the B5700

MCP are so basic that many different approaches have been shown quite successful in drastically improving systems operations. As noted elsewhere in this research, Wulf and many others have reported fairly simple changes made to the MCP in the area of resource allocation and amplified results for the effort (Bowlden, 67, Wulf, 69).

The Burroughs B5700 Time Sharing System MCP uses a dynamic scheduling algorithm used in the validation and verification phases of this research (Burroughs, 72). It uses the following equations:

T = 8 times (C plus (4 times N) minus P) plus 208
E = T times J

where

T = CPU time slice limit in sixtieths of seconds
E = core quantum time limit in sixtieths of seconds
N = number of consecutive times task has used time slice (<7)
C = number of 1024 word main storage chunks used by task
P = declared external priority of task (0 = best, 9 = worst)
J = number of tasks currently swapped in.

Thus it dynamically computes the CPU time slice and core quantum limits to be used on the next time slice based upon (recent) history of execution of the task, similar to that described by Marshall as being ineffective, at least in the cases he investigated (Marshall, 69). The core quantum refers to the time of each occupancy in main storage.

This system has many of the same attributes as the B5700 Data Communications MCP described earlier. It differs primarily in its handling of main storage allocation. Tasks are initially assigned an amount of main storage equal to the next greater multiple of 1024 words

than the main storage estimate assigned by the compiler or user. Then the tasks execute out of this space until the space is needed for another task or the assigned space is exceeded in a no-memory situation. The task is then swapped out to disk, freeing that main storage for use by other tasks. At a later time, the task will be swapped back in, possibly with another 1024 words of main storage if it was in a no-memory condition, though with no relocation, and will be allowed to continue processing. If a task reaches the maximum amount of main storage available on the machine and is in the no-memory state, it is discontinued.

The Burroughs B6700 MCP uses a dynamic calculation of internal CPU priority with several interesting features (Burroughs, 69). It uses the following equation:

priority = (A1 times D) plus
(A2 times TR) plus
(A3 times TE) plus
(A4 times (TE-TP)/(TP plus 1)) plus
(A5 times C) plus
(A6/(TT-CT))

where

A1 through A6 are weight factors set by the operator or by default

D = declared priority
TR = time since last I/O
TE = elapsed time of task execution
TP = processor time of task
C = number of words of core used by task
TT = deadline (target time)
CT = current time (CT<TT).

This is a heavily-time-oriented and static method. It does not directly

modify its actions as the system becomes more or less loaded and hence does not have the adaptivity of the microscheduler in this study. It does implement a dynamic internal priority computation and application generally similar to that discussed by Marshall as being effective (Marshall, 69).

The main storage allocation scheduler of the standard B6700 MCP somewhat resembles that of the B5700 MCP. Dynamic memory allocation is made on demand. When necessary, least-recently-used segments of core are overlaid to enable the allocation of core when available, unused core is insufficient for task demands. Personnel at the University of California at San Diego have developed a variation of the standard B6700 MCP core allocation scheduler which includes an attempt to determine the current working set of pages of a task and keep such pages in main storage in preference to pages not in the current working set (Denning, 68A, Denning, 68C, Denning, 68D, Denning, 70, Denning, 72). Not enough data has been gathered on the effectiveness of this technique to decide in which circumstances it is superior to the standard main storage scheduler. Because of hardware and software design, relatively much more code and data are overlayable on the B6700 than on the B5700 for similar classes of tasks.

The Univac 1100 series Exec 8 operating system uses a dynamic microscheduler (Univac, 71). It is based upon a multiple level organization of tasks to be executed, known as the switch list, which is or-

dered by priority. Tasks at level L have priority over those at level L plus 1. Tasks at a given level are treated on a round-robin basis as far as CPU and core allocation are concerned. Quantum time Q is computed by the following formula:

$$Q = A \text{ times } (1 \text{ plus } P/F) \text{ times } T$$

where

A = time allocation factor assigned at system generation time
P = adjusted task priority
F = priority factor
T = 2 to the power L
L = switch list level.

If a task does not voluntarily release control of the CPU prior to Q units of time, it is moved to the next higher-numbered switch level, thus decreasing adjusted priority P but increasing switch level L. Tasks are returned to an initial switch level after completion of the condition for which they were voluntarily suspended.

The microscheduler allocates the CPU among the programs resident in core according to the structure of the switch list. The CPU is given to the highest priority task ready to run. Since the factors which determine the switch level of a task may change, a task may be allocated varying quanta for its execution during its lifetime. Once a task on a given switch level has lost control of the CPU, whether voluntarily or not, it will not be given control again until all other tasks on that level have been considered for further execution.

The microscheduler allocates main storage on a core quantum

basis. The core quantum is a function of task size, run priority, and computer usage. A main storage priority is kept somewhat analogously to the switch level of CPU usage. A batch task's main storage priority is never changed. When a remote task exceeds its core quantum, its main storage priority is dropped one level and its core quantum is doubled. Whenever any task exceeds its core quantum, it is marked as available for swapping to drum. Tasks are assigned to main storage in the order of their core priority. The initial main storage quantum is a function of the core priority level and run priority. The same task at level L plus 1 has twice the main storage quantum as the same task at level L. On the same level, a task with priority A (best) has twice the main storage quantum as one with priority Z (worst).

The microscheduler computes computer usage for main storage and CPU every six seconds. It then attempts to adjust CPU and main storage priorities so that a desired remote/batch usage ratio is more nearly reached. To do this, it uses an exponentially averaged history of usage for the past few minutes of operation and compares most recent usage values with historical usage. This is claimed to avoid instabilities in the microscheduler due to sudden, heavy demand or batch usage. This approach is similar to that suggested by Marshall as being effective in his research (Marshall, 69).

This class of microscheduler is similar to several of those found in the literature. Its handling of switch levels as related to voluntary

release of the CPU is almost identical to that of Ryder, who found it quite effective (Ryder, 70). Its handling of quantum time as related to voluntary release of the CPU is similar to that of Marshall, who found it not particularly significant (Marshall, 69). Baskett and Schwetman suggest that round-robin CPU allocation is most effective when CPU service time distribution is highly skewed toward zero and the tasks are unrelated, assuming zero task swap time (Baskett, 70, Schwetman, 70, Schwetman, 72). Because of the speed of the 1100 series CPU's, the CPU service time may very well be highly skewed in many cases of tasks not extremely CPU-bound, and thus round-robin CPU allocation may be quite effective in this case.

Prototypical Microschedulers

The following prototypical microscheduling techniques are considered in this research:

B5700 TSSMCP,
round-robin,
FIFO,
instantaneous exponential estimation,
complete history,
adaptive microscheduler.

They were considered since they are representative of many of the techniques used in existing operating systems and studied in the literature.

Following Schwetman and Sherman, all of the microschedulers investigated here which did not have a maximum CPU resource usage

burst time intrinsic to the methods were modified to include one such as that used by the B5700 TSSMCP microscheduler (Schwetman, 70, Sherman, 71). This improves some of the competing methods significantly, as noted by them and as discussed later in this section.

The first microscheduler philosophy considered was that of the B5700 TSSMCP, the same that was used in the validation process described earlier. It is based upon the following equations:

T = 8 times (C plus (4 times N) minus P) plus 208
E = T times J

where

T = CPU time slice limit in sixtieths of seconds
E = elapsed time slice limit in sixtieths of seconds
N = number of consecutive times the task has used up its time slice (≤ 7)
C = number of main storage chunks used by task
P = declared external priority of task (0 = high, 9 = low)
J = number of tasks currently swapped in.

Another microscheduler philosophy considered was round-robin CPU allocation with no priority. In this case, the system resources are visualized as being controlled by facilities with associated queues. Tasks' resource allocation requests are placed at the tail of the queue and are fulfilled from the head of the queue. If the resource allocation requirement is completely fulfilled at the end of some period of time, the task leaves that facility and joins the queue of another facility; otherwise its request is placed at the tail of the queue of the current facility. Variations on the basic round-robin scheduling technique were not considered because of time and resource restrictions on the research.

Another philosophy, similar to that of round-robin CPU microscheduling, is FIFO. In this situation, the facility-resource-queue arrangement is as in the round-robin case; however, when an item is to be placed into the queue of requests waiting for the use of a resource, rather than being placed at the tail of the queue, it is placed ahead of those of all other tasks which arrived in the system later than that task. Philosophies with results by other researchers are similar to that of FIFO and LIFO and random guess. The primary difference between those philosophies lies in the variance of the response time, with all having about the same mean response time; however, FIFO gives smaller variances than the others and hence is the one selected here.

Instantaneous exponential estimation was discussed in detail earlier in this research report as a philosophy of CPU microscheduling. The basic technique is to predict the length of the next resource service time for a task based on the task's immediate past behavior and assign the resource to the task which will probably use it the least amount of time before releasing it for use by other requesting tasks. Though Denning noted that instantaneous exponential estimation is quite prone to errors in measurement, Sherman showed that, in the case of his research, instantaneous exponential estimation was fairly effective as a microscheduling method, at least under a throughput measure of systems effectiveness, and the CDC 6600 environment (Denning, 71B, Sherman, 72). The parameter, ALPHA, of the estimator is fixed for

this case at 0.5, the value Sherman found best in his research, giving equal weight to new and historical data.

Still another microscheduling philosophy which has been proposed is the complete history technique as described earlier. It was shown by Sherman to be almost as effective as the round-robin technique in his research. In this case, the weight given to the historical portion of the estimator increases and that given to the instantaneous portion decreases comparatively. The last microscheduler philosophy considered was the adaptive microscheduler of the current research. It is described in detail later.

System Effectiveness Measures

In order to evaluate the performance of a computer system, system measures are required. Many studies in this area claim generality of results without specifying the system measure with respect to which the results were obtained. Usually, throughput of the system in terms of the average number of tasks completed over a unit of time is the implicit measure used (Sherman, 71). Others have designed measures which fitted the situation, ignoring other possible situations. It was decided to attempt to avoid these problems in this research by considering a set of prototypical system measures.

Since the adaptive microscheduler must be responsive to the nature of the system measure chosen, additional considerations are im-

posed in that case. The overall system measure, or global system effectiveness measure (GSEM), is too general to use in all cases as a guideline by which the adaptive microscheduler may function. Thus for each major GSEM type, a local system effectiveness measure (LSEM) must be defined. In certain cases, such as CPU utilization, the LSEM will be of a similar nature to the GSEM. In other cases, such as THROUGHPUT, it will be of a radically different nature. Though only prototypical system measure classes are considered, there is no logical reason that other combinations of measures than those considered here could not be used. For instance, a combination of THROUGHPUT and CPU utilization could be envisioned for a specific case. The only logical problems arise in the combination of quantities of unlike units, and with nonlinear response to changing resource usages. These problems may be resolved in specific cases, as noted by Hoffman (Hoffman, 71).

Several measures of system performance were used in the adaptive microscheduler evaluation process in order to base the comparisons. Representative samples of different classes of system measures were considered. They are defined and described in the following paragraphs.

THROUGHPUT is a measure many researchers have applied to computer systems and to computer operating system schedulers and microschedulers (Sherman, 72). It is defined to represent the number

of tasks which the system can complete in an interval of time. A good throughput measure is usually received by a system which is biased toward an environment consisting of smaller, shorter-running tasks. Thus the LSEM term for this case was

((projected I/O usage for next time interval for task i) divided by

(projected main storage usage for next time interval for task i)).

This was determined to be the best through experimentation, at least for the current environment. This LSEM favors the smaller tasks which require less CPU attention because of performing a relatively large amount of I/O activity, as used by Sherman.

CPU utilization is another measure which has been applied many times, both in theory and in practice, to computer systems (Kimbleton, 71A). It is not equitable to all types of tasks, as is the use of no single measure as an estimator, since a scheduler based solely upon it rewards CPU-bound tasks with more CPU attention than it awards others. A good CPU utilization is generally received by such a system which has a relatively heavily CPU-bound mix of tasks. Thus the LSEM term for the CPU case was

(projected CPU time for next time interval for task i).

REVENUE is another measure similar to CPU utilization which is sensitive to some combination of resource utilizations. Practically every computer installation uses a different measure of revenue. Properly interpreted, most of the measures considered in this study could

be used for revenue calculation purposes. The specific revenue measure structure chosen was that used on the B5700 at Georgia Tech, which is a multiple of

((3 times CPU time) plus I/O time).

Thus the form of the LSEM term for this case was

(3 times (CPU time projected for the next time interval for task i)
plus (I/O time projected for next time interval for task i)).

RESOURCE utilization is a measure that is sometimes used to measure how well a computer system is using its resources on an overall basis (Lucas, 71). It is a somewhat fairer measure than some of the others when used for estimation purposes, as it evenly favors tasks which use the various resources. The GSEM for this case is

(CPU time plus I/O time plus norm main storage usage).

Thus the form for the corresponding LSEM term was

((CPU time for next period for task i) plus
(I/O time for next period for task i) plus
(normalized main storage usage for next period for task i)).

I/O utilization is seldom used as a measure of computer system performance, but it is included in this research for completeness (Losapio, 72). Analogous comments apply to it as apply to CPU utilization. The LSEM term used for this case was

(projected I/O time for next time interval for task i).

LATENCY is a measure which has often been used in the form of

response time measures (Lassettre, 72). A microscheduler based upon this measure favors the extremely short-running tasks such as those normally encountered in time-sharing operations over the longer running tasks such as those normally encountered in batch operations. The GSEM for this case can be stated in several manners, but that used for this research is

(1 minus (effective overhead time due to queue handling)

divided by (swap in time)).

Its LSEM term can take one of several forms, depending upon the orientation of the policies of the management of the system, but the LSEM term used in this research was

((projected CPU time for the next interval for task i) divided by

(projected elapsed time for the next interval for task i)).

SYSTEM UTILIZATION is a measure used quite often when considering the use of hardware monitoring devices (Estrin, 67B). It measures the activity of the system in terms of the complement of the time when the system is idle. It is a fair and useful practical measure because if a computer system is being run a limiting elapsed amount of time, system utilization considerations can often increase the performance of the system significantly by reducing idle time. Several researchers have shown that this may be done in general by assigning a resource to that task which will use it the shortest amount of time before releasing it (Sherman, 72). Thus the LSEM term used in this

case was

(zero minus ((projected CPU time for next time interval for task i) plus
(projected I/O time for next time interval for task i) plus
(projected mass storage usage for next time interval for task i))).

SYSTEM COST NULLIFICATION is a measure suggested by researchers to directly attack the return-on-investment problem (Wald, 67). It is similar to simple resource usage except that the value of the measure is proportional to the sum of the values of each of the resources of the computer system used by the tasks. The coefficients were derived from 1972 B5700 price lists in consultation with Burroughs personnel. They represent the book price of B5700 computer system components, ignoring site preparation, air conditioning, shipping charges, maintenance, and operator salaries, which are not effected except trivially by the microscheduler. Main storage usage is normalized to take into account its dichotomous nature under the B5700 TSSMCP. Thus the form of LSEM term used in this case was

(3.14 times (projected CPU usage for next time interval for task i))
plus ((number of I/O paths)
times (projected I/O usage for next time interval for task i))
plus (8 times (normalized projected main storage usage for
next time interval for task i))).

In summary, the definitions of the various system effectiveness measures considered in this research are given in the following table.

Table 1. System Effectiveness Measures

Name	GSEM (Overall)	LSEM Term (Instantaneous)
THROUGHPUT	(tasks / unit time)	(I/O usage) / (core usage)
CPU	(CPU usage)	(CPU usage)
REVENUE	(3* (CPU usage) + (I/O usage))	(3* (CPU usage) + (I/O usage))
RESOURCE	((CPU usage) + (I/O usage) + (core usage))	((CPU usage) + (I/O usage) + (core usage))
I/O	(I/O usage)	(I/O usage)
LATENCY	(1 - ((queue handling time) / (swap in time)))	(CPU usage) / (swap in time)
UTILIZATION	(1 - idle time)	(- ((CPU usage) + (I/O usage) + (core usage)))
COST	((3.14* (CPU usage)) + (4* (I/O usage)) + (8* (core usage)))	((3.14* (CPU usage)) + (4* (I/O usage)) + (8* (core usage)))

CHAPTER III

THE ADAPTIVE MICROSCHEDULER

Operation of Microscheduler

The heart of this model lies in the use of an heuristic, dynamic, adaptive task scheduling technique which attempts to schedule the allocation of resources among the tasks at hand in a manner more conducive to semi-optimal system operation than embodied in other microschedulers in use today.

The general goal of the adaptive microscheduler of this research is to attempt to maximize the general systems effectiveness measure (GSEM) through maximizing the local systems effectiveness measure (LSEM) at each decision point. It does this through evaluating the predicted values of LSEM obtained by tentatively assigning the requesting tasks to the requested resources, then actually making the assignments which maximize the LSEM. The prediction of new LSEM values is based partially upon exponential estimation based upon previous values of system variables and partially upon the differences between the most recent predictions and actual values (the correction). At certain intervals the prediction process may be parametrically adjusted in order to adapt to changing conditions in the environment in

which the microscheduler is currently operating.

This microscheduler is intended to satisfy the design requirements of a heuristic, dynamic, adaptive, task-oriented internal resource allocation scheduler, which attempts to schedule resource allocation so as to maximize the productivity of the system. Succeeding chapters verify that it actually performs as expected, through the use of computer simulation, using data collected from a B5700 computer system running a selection of tasks drawn from the Georgia Tech environment.

Level of Detail

When preparing to model any system, the analyst must choose the model's world viewpoint carefully. If it is chosen too small, important characteristics of the model will be obscured by detail. If it is chosen too large, the characteristics will be obscured because of summary action. The viewpoint of a model's subcomponents must also be chosen carefully for essentially the same reasons.

The microscheduler of this research is assumed to function at the general level of detail and point of view represented by the following quantities:

- -task state vector
- -main storage segment
- -I/O transaction
- -CPU interrupt processing interval.

The following quantities are considered to be on a higher level and are

assumed to be included in the model parametrically or algorithmically:

- -operating system architecture (except microscheduler)
- -macroscheduling characteristics
- -workload characteristics
- -mass storage requirements
- -management policies and goals
- -system profit/cost functions (GSEM)
- -hardware configuration.

The following quantities are considered to be on a lower level and are assumed to be included parametrically or implicitly, or they are to be ignored:

- -multitasking interference in hardware
- -set of CPU and I/O instructions available
- -dynamic and static address relocation delays
- -exact structure of tasks and job steps.

The preceding representation allows for the concentration of attention on the microscheduler itself and attempts to avoid the consideration of subcomponents of computer systems not important here. The remainder of this section presents the model and associated concepts, following the level of detail and point of view just prescribed.

Representation of Tasks

Logically, tasks are represented as passive entities acted upon by the microscheduler. Each task has associated with it various quantities which may be interrogated by and/or changed by the microscheduler as real time proceeds. Some of the quantities are direct functions of time, such as deadline information. Most others are functions of assigned resource time, such as CPU and I/O time usage.

Actual task internal representation is transparent to the microscheduler just as is actual resource allocation internal representation. Thus the microscheduler is quite austere, manipulating the tasks and system under its control with respect to certain constraints and driving functions, without direct regard for their internal structure.

This approach is taken since, from the point of view of the microscheduler, the only important aspects concerning a task are its existence and its external attributes, not its internal structure. This is also a highly realistic viewpoint for an actual implementation of such a microscheduler as envisioned here.

System Effectiveness Measures

One measure of the productivity of a computer system is

((billable charge) per (real time unit)).

This is consistent and reasonable since increasing the productivity of the computer system increases the charge realized for a given time interval, since the system is capable of executing more jobs. Assuming that some constant, consistent accounting system is in effect and that it is of the nature that an increase in the use of a resource causes a corresponding increase in the charge, that measure will be used as one of the system objective functions, also called global system effectiveness measures (GSEM) as investigated in this research with respect to various philosophies of microscheduling.

This measure is too general to use directly as a decision tool for a microscheduler. Thus a local system effectiveness measure (LSEM) is required to be defined. It would ideally be of the nature that optimizing it at each point in time would be equivalent to optimizing the GSEM for that entire time period. This is, however, infeasible for a dynamic, unpredictable, nonanticipatory system because it is not possible at a given point in time to make the decision which is known to be correct at some later point in time (hindsight is always better than foresight). All that can be done is to make the apparently best decision at each point in time at which a decision must be made, hoping that worst-case situations will not occur, will be mollified by the revised decision process, or will not be severe enough to cause other than temporary degradation of the system so that the microscheduler can have time to take evasive measures.

The LSEM's used in this study are all defined to be the weighted sum of certain system and task parameters evaluated at each instant in time at which a decision must be made. By proper choice of the parameters and weights, optimizing an LSEM on an instantaneous basis may be made equivalent to a suboptimization of a given GSEM over a longer time interval. The better the decision function of the microscheduler is, the better the value of the GSEM which is realized, ignoring the possible additional resource load of the microscheduler.

The specific global (and local) system effectiveness measures

considered in this research were discussed earlier and were represented by the following list of measure classes:

-throughput
-CPU utilization
-revenue
-resource usage
-I/O utilization
-system utilization
-system cost nullification.

Note that the LSEM's are all dependent upon resource utilizations only. Thus they are all effectively computable from recent and historical information, and no future information such as task completion time is required. Other system effectiveness measures could be considered by defining the corresponding LSEM's. For instance, a new GSEM composed of equal weights of throughput and CPU utilization could be defined. Of course, nonlinear measure values and incommensurable units may complicate that process.

The values of the corresponding weight factors are determined from the corresponding terms in the GSEM from historical information, from policy decisions, and by heuristic means. Those terms known to have constant values and constant weight factors in a given environment should be dropped from consideration of terms of the LSEM. This class of GSEM does not consider line-connect charges nor unit charges, which are almost independent of computer performance, and corresponding terms will not appear in the LSEM.

The construction of the LSEM for a given environment is quite

important since it links the desired operation of the microscheduler to management's goals for the system. Certain goals may very well be conflicting. For instance, faster turnaround is in conflict with increased CPU utilization; the former is proportional to the decrease of instantaneous CPU usage whereas the latter is directly proportional to total CPU usage. Thus the relative coefficients of these terms must be adjusted to conform to management goals to resolve the conflict.

Other terms and their coefficients in the LSEM which correspond to terms in the GSEM must be chosen relative to resource utilization so that the GSEM is accurately represented in that portion of the LSEM. Terms and their coefficients in the LSEM which correspond to other system levels may also be chosen to correspond to management's goals, even though they do not have direct realizations as terms and coefficients in the GSEM. In given situations, it is up to a clever individual to construct the LSEM for a specific GSEM.

The decision, then, can be made in the manner described earlier in this section using the task LSEM predicted values for the next time interval to determine which requesting task is to receive use of each resource next.

The LSEM need be evaluated only in the cases in which a decision is required. This implies that the allocation of resources which are not scarce is simpler than the allocation of scarce resources. Thus in the case of multiplexed I/O paths, if fewer I/O operations of a

given class are in progress than there exist paths and another I/O allocation request is made by a task, there is no valid reason the request should not be immediately filled. If, however, all paths are in use, the LSEM would be evaluated to determine in which order requests by different tasks should be filled when paths do become available. Similar comments apply, of course, to CPU, main storage, mass storage, and other such possibly scarce resources.

Trends in System Performance

The microscheduler must be able to detect trends in system performance in terms of values of LSEM, its terms, and the attributes of the tasks in execution. It is necessary to minimize the resource requirements of the microscheduler in order not to negate the gains made by the improved microscheduler. Thus extremely sophisticated techniques, or very crude ones using much storage, may not be used. This also implies that large tables describing in detail many system and task variables may not be maintained. Thus prediction techniques must rely on small numbers of relevant variables to predict new values.

There is a long-term class of data, statistically describing the environment and current set of tasks. There is the short-term class, reflecting averaged values over some much shorter period of operation. Then there is the instantaneous class detailing the most current values of the system and task variables. These three classes of values enter

into trend detection and prediction in the microscheduler of this research.

Because, at a given instant in time, averaged values of system and task variables gathered more recently are better predictors of near-future values than are less recent values, short-term averages must be weighted so that recent values carry more weight than less recent values (Denning, 71B). The concept of weighted moving average (or window) is the basis for certain techniques which have been developed to track changing patterns of demands in inventory control and operating systems (Burroughs, 72, Denning, 71B, Wulf, 69). These techniques are suggested in this research as methods used to track changing patterns in resource requests and allocations. In particular, exponential weighted moving averages, trend detection, deviation, and mean absolute deviation (MAD) are used here (Denning, 71B). Experimentation indicated that exponential estimation was more effective in the investigated environment than other adaptive or nonadaptive methods.

Exponential estimation is a convenient class of weighted moving average for computation in that it does not require the keeping of large tables of historical data for each resource of interest in order to facilitate prediction. It has the capability to adjust to change, but its rate of response can be adjusted dynamically and parametrically. This parameter is normally referred to as ALPHA, but for the purposes of this research, it will be known as A. In this method,

new average = A times (new value) plus (1-A) times (old average)

where A is a scalar between zero and unity exclusively. As A approaches unity (zero), the average is more (less) weighted to the new values. Parameter A may thus be varied to cause the microscheduler to be more or less responsive to changing conditions. Bounds on A sharper than zero and unity may be parametrically specified to avoid the extreme cases.

It is known that the use of the basic exponential estimation technique will produce an estimate which lags behind data containing systematic trends. The amount of this lag may be determined from the mathematics of the exponential smoothing formula, and a trend correction may be applied to make the estimate more accurate. By the definition of trend,

current trend = (new average) minus (old average).

Because fluctuations in resource requests will cause fluctuations in the new estimate for resource requests, there is needed a method to estimate the trend present in the data to cause the dampening of this fluctuation. Thus

new trend = B times (current trend) plus (1-B) times (old trend)

is used to form a second-order exponential smoothing method on the estimates to detect trends, with parameter B as a scalar between zero and unity exclusively. A new parameter B was used here rather than A because the noise contents of the averages and trends are assumed to be

of different nature.

Since the objective is to produce a resource request prediction for the next time period, the new trend must be added to the new average and the trend lag correction. Thus predicted value is defined by the following equation:

predicted value = (new average) plus ((new trend) divided by (1-A)).

The deviation provides a simple means of computing the errors made by the prediction process. It is defined by the following equation:

deviation = (predicted value) minus (actual value).

It is useful in determining how and when to change the values of A and B and time interval between corrections T. If the value of the deviation is changing signs rapidly, outside of some small threshold, the system needs damping and A and/or B and/or T should be decreased. If it is increasing or decreasing consistently, A and/or B should be increased and T should be decreased. When the deviations are changing signs but are smaller than the threshold, the system is performing as expected and T may be increased, if it is not already at its upper bound.

The exact means by which A, B, and T are changed are given in Chapter IV. In particular, changing T had little effect as long as the formulae were applied at certain distinguished points, also as described there.

The mean absolute deviation is useful for determining the accuracy of the prediction process and is defined as follows:

MAD = (the sum of the absolute deviations of each measurement) divided by (number of measurements)

MAD is not used directly in this research but is used indirectly as a means of comparing certain variants of adaptive microschedulers.

One additional complication must be considered here. Since it is possible that the time between predictions is not constant, a correction must be made to the predicted value. Though many types of corrections were possible, linear interpolation was used successfully in this experimentation.

The formulae used in the predictor portion of the adaptive microscheduler may be summarized as follows:

1. AV(T2) = (A times V(T2)) plus ((1-A) times AV(T1))
2. CT(T2) = AV(T2) minus AV(T1)
3. NT(T2) = (B times CT(T2)) plus ((1-B) times CT(T1))
4. PV(T2) = AV(T2) plus (NT(T2))

where

T1 = previous time
T2 = current time
AV = average value
V = current value
CT = current trend
NT = new trend
PV = predicted value

and time between applications (T2-T1) is assumed constant.

To correct for variable time of application of prediction, equations 1 and 3 above must be corrected as follows:

F(T3) = MIN (1.00, MAX (0.00,
((F(T2) minus F(T1) times (T3 minus T1)
divided by (T2 minus T1)) plus F(T1))))

where F is AV or NT,

T3 = current time
T2 = such time that "interval of application" would have been constant
T1 = previous time.

Given that TO = time before previous time, then T1-TO = T2-T1. If T3 is between T1 and T2, the above equation for F forms a linear interpolation formula, and if T3 is greater than T2, it forms a linear extrapolation formula. Initially, times zero and the first time of application are taken as TO and T1, respectively. In case the extrapolation leads to a resource utilization not between 0.00 and 1.00, it is corrected to the nearer extremity.

These equations form a second-order, exponentially smoothed system which remains open to new information. The correction for variable time of application is not found in the literature, but significantly improves the predictive powers of the method. Assuming that values at some initial point in time may be specified, the equations can then be iteratively applied to provide new predicted values (AV(T)) and new trend values (NT(T)) at each later point in time at which a prediction is required.

In summary, this scheduler attempts to detect short-term trends in its monitored variables by comparing smoothed and trended values with most recently measured averaged values. By this means, a task which begins to request more or less of a resource may be lo-

cated and processed properly. Longer-term trends may be detected by comparing the statistical historical information with the smoothed and trended values, if necessary. It attempts to quiet much of the noise and inaccuracy of instantaneous estimation through the use of averages gained over somewhat longer periods of time, along with careful parameter variation. This avoids many of the problems mentioned by Denning with respect to exponential estimation (Denning, 71B).

Prediction and Correction Techniques

Static exponential estimation and trend detection in the microscheduler would be useful to some extent in a passive sense. However, to make the microscheduler a more active controller, a predictor is required. The predictor used here uses the same mechanism as in the trend detector. The actual technique used is that of extrapolation based on exponential estimation (Denning, 71A). The selection of each task in turn is simulated for each requested resource. The predicted LSEM is evaluated for each combination of tasks and requested resources. Then, taking into account the parallelism of each facility, the microscheduler can more properly select the tasks to which to allocate the resource requests for the next time interval than can other nonadaptive microschedulers.

This construction provides an almost built-in correction mechanism. If the predictor is consistently under- or over-estimating the

value of a variable (sum of deviations is not going to zero), a corrector may be applied to modify the parameters A and B of the exponential estimator to make the microscheduler a little more responsive or to decrease the time application interval to make it a little more accurate. In case the predictor is performing badly, over- and under-estimating values, the parameters A and B should be modified to make the microscheduler more stable. Thus the predictor-corrector portion of the microscheduler acts as a homeostatic stabilizer to the microscheduler, allowing the adaptivity to handle changing conditions, but also stabilizing the process.

The process as used in this research is quite simple, as it must be. Separate A's and B's are maintained for every task in the mix and for every facility of the system capable of being assigned to or used by the tasks. When a task enters the mix, its A's and B's are set to predetermined initial values, and the task's historical information is set to values to reflect the predicted initial normalized usage of the CPU and I/O facilities and the actual normalized initial usage of the main storage facility. In other cases, facilities such as those described earlier may be used. From a large number of experiments with various values for those initial parameters, it was determined in the environment considered in this research that good initial values for the A's were in the range of 0.4, for the B's were in the range of 0.3, and for the predicted initial facility usages were 0.5 for CPU and I/O. Thence-

forth, whenever the predictor-corrector process is activated to predict a new value for a task/resource usage combination, the historical information and more recent information are available to allow the effective prediction of a new LSEM value for a task. Of course, the historical information is updated each time the predictor-corrector portion of the scheduler is used. By noting the types of errors between the previous predicted task/resource usages and the current ones, the process determines in an algorithmic manner the corrections required, if any, to change A, B, and the time interval of application to make the next predictions of task/resource usages, and hence LSEM values, more accurate.

The exact nature of this corrector algorithm is environment-dependent; however, a large number of experiments were performed to determine the best manner in which to perform the correction in the environment of this research. It was determined that the best manner in which to perform the correction in this environment is represented by the following:

- compute difference between predicted value for task/resource usage and actual value.
- if difference is greater than threshold, check signs of current difference and previous difference.
- if they have different signs, the process is oscillating and must be made more stable by relying more heavily on historical information; if A is not already at its lower limit, it is decreased by DA (defined externally); otherwise, if B is not already at its lower limit, it is decreased by DB.

-if they have the same sign, the process is not responsive enough to new data and must be made more responsive by relying more heavily on more recent data; A and B are increased in an analogous manner to that described previously.

-using the most recently computed values for A and B, and regardless of other tests, predicted task/resource usage value is generated and used to compute a predicted LSEM value; since the times between applications are variable, linear interpolation is used to take this into account in the prediction process.

In further experimentation, it was determined that, in the environment considered here, the threshold for correction could be set system-wide at 0.1 with almost no loss in the overall precision of the prediction-correction process. Slightly better results may be realized in this environment by making DA a function of the error between predicted and actual values. It was also determined for this environment that the most effective times at which to apply the predictor-corrector were those at which a known major change in task or system status, such as a swap in or out or main storage expansion, occurred; this allowed the minimization of the overhead of the process while maintaining its accuracy. The concept of computed time of application is believed to be very important in systems not as main-storage-limited as in the B5700, and thus the explanation of the algorithm retains that concept. A flow-chart depicting the actions of the predictor-corrector portion of the adaptive microscheduler appears in Figure 1.

In summary of the operations of the predictor and corrector, an

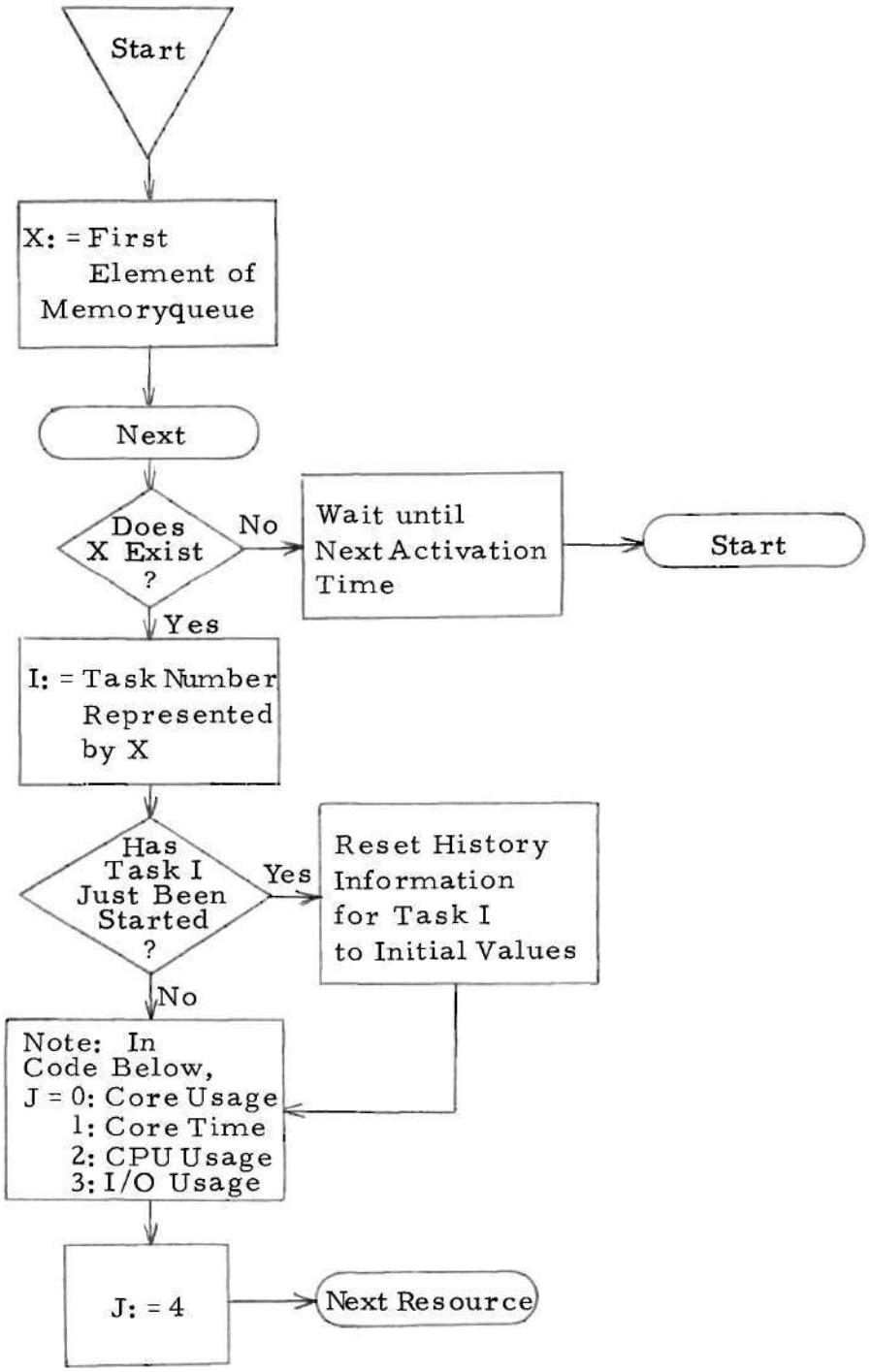

Figure 1. Flow Chart of Microscheduler.
(continued on next page)

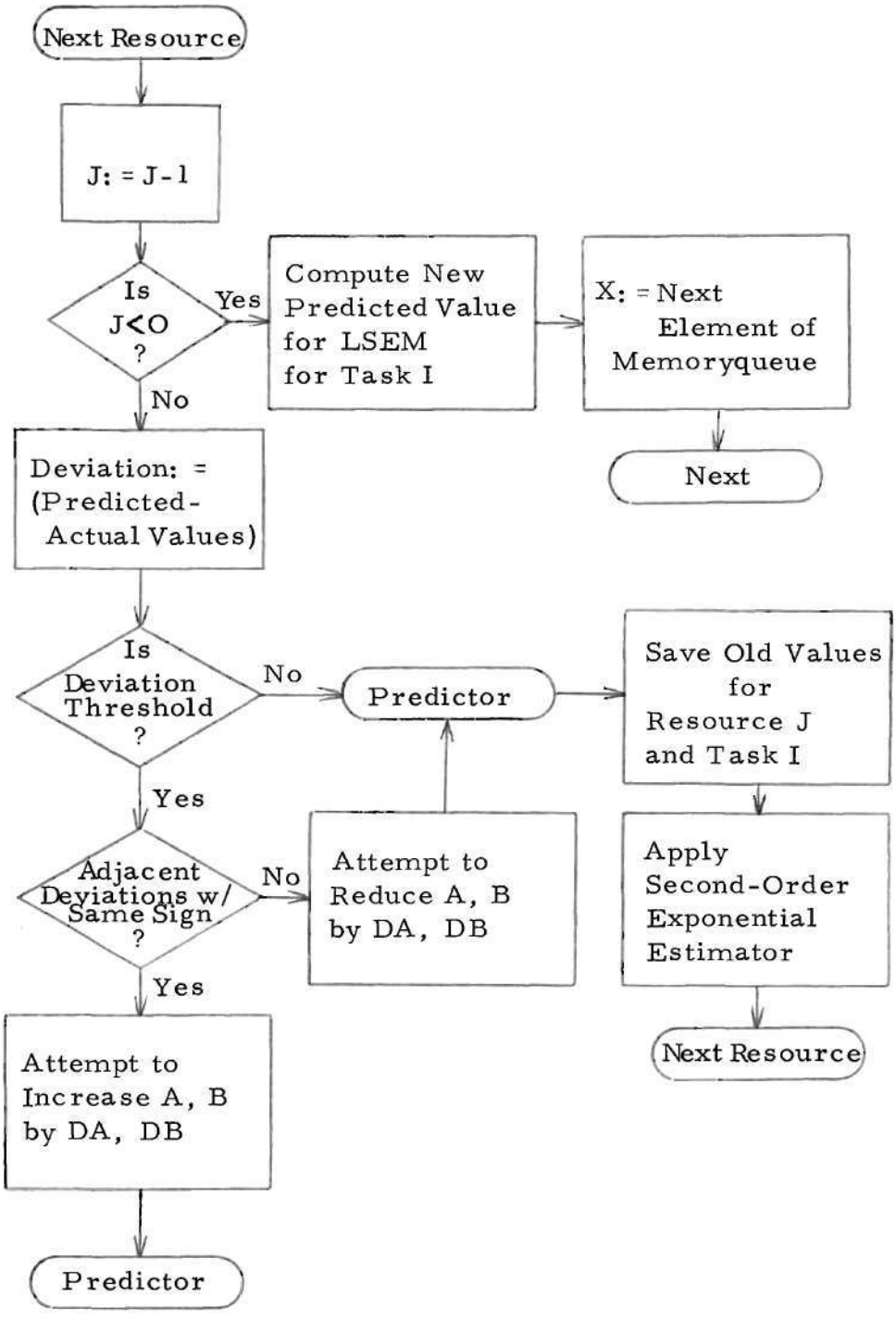

Figure 1. Flow Chart of Microscheduler. (concluded)

attempt is made to make the microscheduler homeostatic in the sense that the corrector subcomponent is capable of making the operations more or less responsive to changes in the environment in response to various stimuli.

Summary of Operation of Microscheduler

The microscheduler described here functions as the controller and comparator in an information feedback control loop. The scheduler operates on vectors of task attributes, attempting to optimally satisfy task resource requirements under the conditions imposed by a vector of available resources. It does this through the evaluation of the projected LSEM, assuming various combinations of task-resource allocations, and selects the tasks which will optimize the projected LSEM, whenever a decision must be made. The projection process involves using a second-order exponential estimation process, corrected by recent rates of change and recent trends. Certain parameters of the LSEM and of the exponential smoothing function are varied within limits to adjust to changing environmental conditions. It is experimentally verified in this research that such a microscheduler, in actual operation, improves the performance of a computer system in an environment obeying the general requirements discussed earlier, and such as that actually measured in actual operation through the use of software monitors, as described in the next chapter.

Comparison with Other Research

The design of the adaptive microscheduler of this research was motivated by the work of many other researchers. The primary references were discussed in the last chapter. This included the careful design and analyses of Denning, Ryder, Wulf, Bernstein, Wilkes, Sherman, et al., and the pragmatic design of various internal schedulers in use in actual operating systems. This study continues the task of the previous researchers in the search for better modes of internal scheduling. Because of its central-control position in an operating system, relatively small changes in the microscheduler may cause relatively large improvements in the performance of the operating system as a whole. Thus many researchers and manufacturers have expended much energy and time in the improvement of microschedulers.

In Chapter II of this report, the efforts of many researchers were discussed. Each of them reported studies of resource utilization in digital computers. In most cases, specific resource scheduling methods were considered, such as the following:

-complete-history (Stevens, 68, Sherman, 72)
-round-robin (Baskett, 70, Schwetman, 70, Sherman, 72)
-foreground-background (Rehmann, 68)
-reward-penalty (Marshall, 69, Eisenstein, 70)
-moving-average (Denning, 70)
-random-guess (Sherman, 72)
-FIFO (Sherman, 72)
-pre-emptive (Sherman, 72)
-policy-driven (Bernstein, 71)
-load-adjustment (Wilkes, 71)
-exponential-smoothing (Wulf, 69, Ryder, 70, Sherman, 72)

-stochastic-estimation (Eisenstein, 70)
-heuristic (Ryder, 70).

In a few cases, the researcher attempted to develop methods which would predict near-future resource utilization and, accordingly, schedule the tasks' resource allocation (Denning, 70, Eisenstein, 70, Sherman, 72, Wilkes, 71). All of those predictors were fixed-formula, nonadaptive, fixed-parameter methods, normally attempting to optimize the use of only one resource.

In contrast, the microscheduler of this research is adaptive in terms of changing its parameters in response to the error committed in the last prediction. Furthermore, this method is second-order in its exponential estimator, a technique chosen to make more accurate predictions and to be less susceptible to noise. Since the time of predictions will, in general, not be evenly-spaced, a linear correction is applied to attempt to predict more accurately. To this researcher's knowledge, this correction for uneven time steps is not considered in the literature but proves quite effective in this research. The improved prediction formulae contribute to the success of the adaptive microscheduler. Moreover, the three most critical resources, CPU, I/O, and core of the system and environment at hand are treated, rather than only one.

The literature lacks any general methodology for effecting microscheduler improvement for a variety of systems effectiveness

measures. This research provides a general method of using resource utilization predictions to attempt optimization with respect to various measures of systems effectiveness, even though a different formula using the predictions may be required for different measures.

CHAPTER IV

DESCRIPTION OF SIMULATION MODEL

Validation Techniques

Predicting and measuring operating system performance involves abstracting interesting features, obtaining values for system parameters, and analyzing their effects on numerical measures of performance. Models must be validated by comparison with measured performance, and measurements need models to give them meaning and generality (Johnson, 70). This has resulted in a spectrum of approaches, from purely mathematical models of interactive systems to practical measuring devices. At intermediate points are analyses of actual computer operating systems and synthetic task loads suitable for controlled experimentation (Baskett, 70, Sherman, 71, Sherman, 72).

Experimental validation to an absolute certainty is impossible. However, logical validation of a model or technique in an environment is possible to a desired degree of certainty. In the case of the current research, this means careful measurement and specification of the environment, followed by simulation of a controlled test case.

First, measurement must be considered carefully as a concept

in itself. Careful measurement of an environment is important, not only for the simulation which follows here, but for any real implementations of the proposed technique.

Second, the simulation must be formulated properly. If it is not, the validation and verification processes based upon it are questionable. If it is, its configuration may be used as an aid in actual implementation.

The sections following discuss the processes used to calibrate and validate the simulation model itself and to verify that the actions of the new microscheduler are as expected, in the simulated environment and workload.

Environmental and Workload Measurement and Modeling

Ferrari considered the problem of the accurate characterization and selection of an existing system's workload for the purpose of measurement, evaluation, and subsequent optimization of performance (Ferrari, 72). He noted that there are three following basic types of techniques for obtaining the characteristics of a workload: natural--using the real workload directly; artificial--constructing a workload supposedly statistically equivalent to the real workload; and hybrid--assembling a workload from parts of the real workload.

Measurements to obtain natural workloads must be long enough and representative enough that the workload is accurately portrayed.

Scherr collected data on CTSS for his dissertation during the prime shift, for about one month, on 47 different occasions, for a total of 112 hours (Scherr, 67). Bryan reported statistics taken from JOSS during 20,000 hours of operation (Bryan, 67). In the current research, several years of log data were available for general statistical characterization of the workload, though specific stereotypical task type analysis required less time than was required by Scherr. Another proponent of natural workload usage is Cheng, who proposed trace-driven modeling (Cheng, 69). In this case, software or hardware probes are used to trace the time and sequence of epochal events in the processing of various task types as found in a workload. The traces may then be used in simulation models directly or after further processing (Tetzlaff, 72).

Most of the system evaluation techniques developed and used are based on artificial workloads. Almost no efforts have been made to actually construct artificial workloads statistically equivalent to some real workload. Joslin has worked in this area with the goal of basing computer selection on benchmark results; however, the number of variables present in an actual selection process renders such a process questionable (Joslin, 66). Several manufacturers of programmable front-end data communications processors provide a special program which runs in the data communications processor and simulates user activity. These form true artificial workloads which are useful for debugging hardware and software components of the system, but are not

particularly useful for system performance optimization except in specific cases. Similar in concept to that is the use of scripts of time-sharing user-simulated input for driving time-sharing system simulators (Saltzer, 70).

Hybrid workloads were first introduced by Joslin under the name of application benchmarks (Joslin, 65). He suggested a task characterization into a number of specific slots. Then tasks falling into the same set of slots are related, and similarity class partitions of the workload may be performed. When constructing a hybrid workload, representatives of each of these classes can be used to characterize the real workload. Karush suggested a method similar to this except that the classes are not assumed known beforehand; the system's input and responses to it are recorded and played back during simulation studies (Karush, 69A, Karush, 69B, Karush, 70A, Karush, 70B). This method is not conceptually different from trace-driven modeling, but it does incorporate system input as an entity, and this is advantageous for time-shared systems (Lassettre, 72). It is quite closely related to the concept of a stereotype simulation model, as used in this research, and in the research by D'Angelo and Windeknecht (D'Angelo, 72). In the latter study, a simulation model was defined based upon the conjecture that a society of interacting stereotypes may retain properties of behaviors of real systems or components of systems. This approach is suggested rather than the more commonly used approach of specifying the mean and variance of

the parameters of the system, as it should be more accurate.

In order to validate the simulation and in order to allow the investigation of a variety of microscheduling techniques and system measures, careful measurement and statistical analysis of an environment is required. The environment chosen for this work was the Burroughs B5700, now a part of the laboratory of the School of Information and Computer Science at the Georgia Institute of Technology. This choice was made for several reasons, including the familiarity of this researcher with the hardware and software of that machine and the availability of it to him for testing.

Actual measurement work began with the construction of an interpreter and other programs specifically designed to allow the manipulation and summarization of data taken from the log accounting tapes. Information on these tapes described the workload of the B5700 starting in January, 1967, on a job step basis. Each job step entry contains the following information:

-job step name
-job step type (compile, execute, etc.)
-accounting information
-stop date
-start time
-stop time
-CPU time used
-I/O time used
-job step finish type (normal or error finish, syntax errors)
-main storage used
-mass storage used
-tape usage

Approximately 40,000 to 50,000 job step entries were recorded for each school quarter of operation. Twenty-two quarters of work were available for analysis, so about one million job step entries were recorded on tape and were candidates for use in this study. Such attributes as totals, averages, extremes, deviations, etc., were thus available on a job step basis to describe the environmental workload as a whole in terms of information derivable from the above data.

Other analyses of the data determined the relative percentages in the workload of each class of job step type and job step finish daily for each year of investigation. As could be expected, the structure of the daily workloads changed due to large numbers of factors. However, there are noticeable weekly, quarterly, and yearly cyclic factors due to the nature of the university environment of the computer system. Other major noncyclic changes were due to computer center policy changes, such as the change as of July, 1971, of offering preference to small jobs (those that require no more than two minutes each of CPU time and I/O channel time). These daily variations were too unreliable to use in the simulation to represent the overall workload, as expected, so yearly average data were used instead.

The preceding paragraphs have discussed how the workload was characterized as a whole, in terms of job step classes and the relative weights which should be applied to each case. As useful and necessary as this is, alone it is not sufficient. Descriptions of other quantities,

such as task characteristics, must also be derived and cannot be determined from job step summary information only. Thus a software monitoring system was developed specifically for this research by this author (Pass, 71). In its major form, it allows a B5700 user to dynamically monitor and analyze another job step under various conditions. It was used in the current research to assist in the derivation of most of the job step profile distributions required by the simulator from which the latter constructed the task descriptions. It has also been used on a practical basis by personnel at Oakland Army Base to analyze the performance of their B5700's through the characterization of problems in and improvements to their longest running and largest production programs. As of this writing, their work continues; however, in performance gains due to other similar systems such as LEAP and PROGLOOK for the IBM S/360/370 under OS/360, 10 to 20 per cent improvements are conservative estimates of improvement likely.

The portion of the software monitor which dynamically monitors other tasks is called SNOOPY. It is partially contained in the operating system itself. The driver provides the basic timing of the measurements, the identification of the job step being measured, the main storage address of an I/O buffer area in which the operating system can store results, and the logic to perform the I/O operations when required. The additional operating system module actually performs the analysis of the job step and returns the following items of interest into the buffer

area of the driver:

-name of identified job step (-1 if none)
-compiler flag, on if job step is compilation
-processor B flag, on if job step is running on processor B
-current segment number
-current address in segment
-time of day
-CPU time used
-I/O time used
-MCP overlayable main storage used
-MCP non-overlayable main storage used
-job step code main storage used
-job step data overlayable main storage used
-job step data non-overlayable main storage used
-other job steps' overlayable main storage used
-other job steps' non-overlayable main storage used
-time since last system failure.

The job step portion of the program treats this information placed into its buffer as blocked logical records and writes physical tape records when the buffer is full. Since the job step uses double-buffering and a proper choice of blocking ratio, tape I/O never interferes with the recording of data concerning the other job step. In actual operation, one tape I/O was done every 15 to 20 seconds. SNOOPY is always run at a higher priority than the job step being monitored. By the manner in which the normal B5700 CPU scheduling algorithm assigns the CPU to tasks, whenever SNOOPY is requesting a reading, it will be performed immediately before the other job step is reinitiated after an interrupt (Burroughs, 72).

By this process, much information may be gathered dynamically concerning a given job step, under various conditions. The interfer-

ence due to SNOOPY is so small it may be safely ignored. This was verified in several controlled experiments in which SNOOPY was instructed to continually monitor some large, long-running job steps, and no other job steps were allowed on the system. Job step times for the run-alone situation were in all cases practically identical with those for the SNOOPY situation. Core requirements for SNOOPY were constant at about 1300 words, once program code and I/O buffers had been brought into core initially. So little interference was noted in terms of CPU, I/O, and main storage that the results in the form of job traces should be accurate, with accuracy approaching that of hardware monitoring. Hardware monitoring was not possible in this case not only because of its expense and unavailability but also because the dynamic nature of main storage allocation on the B5700 would severely limit the types of data which could be gathered. In particular, segment and address information would probably have been impossible to gather with a hardware monitor, as would have been total main storage usage.

Once the trace has been produced for a given job step or set of job steps, the further processing required to actually use this information in meaningful forms becomes the next problem. Cheng suggests using the information almost in its raw form to facilitate trace-driven modeling (Cheng, 69). This approach is intended to solve the inherent dichotomous dilemma of simulation models of many assumptions on one hand or extreme detail on the other. In trace-driven modeling, the

workload and the responses of the system to the workload are supplied as input to the model. Trace-driven simulation eliminates or simplifies the tasks of specifying some system parameters and coding the details of system operation. He suggests the construction of a job profile library, containing the traces of many job steps of many different types. Then, the simulator can be instructed to select certain classes of profiles from the library for the simulation of an operating system running under that workload. This approach is enticing, for the modeler's work is reduced significantly, and the chances of having accurate simulation results are greatly enhanced. Unfortunately, Cheng's method has several operational flaws. One, of course, is that a trace-described workload is valid only on a given computer system configuration running under a given operating system, and thus the use of trace-driven simulation as a predictive, not descriptive, model is limited. To this researcher, there is a more serious flaw, however, in the loss of knowledge and insight into the actual internal operations of some of the algorithms represented by the simulated operating system. Rather than requiring the modeler to explicitly define many of the algorithms, the data itself is used to implicitly define their operations. Denning also notes that too often, an algorithm is considered a black box with input and output having mystical relationships, defined quantitatively, when a modeler really should first open the box and study its internal workings, even roughly, before settling for non-

qualitative methods (Denning, 71). Thus, unfortunately, Cheng's method was put aside but not forgotten.

Nielsen has also investigated the problem of accurate description of workload for the purpose of simulation modeling (Nielsen, 67A, Nielsen, 67B, Nielsen, 71). In an early simulator, he proposed the use of parametric instruction sequences to describe the workload. Each parametric instruction sequence was intended to capture the essence of the structure of a class of tasks and users. A special description language, designed as a set of instructions, was developed for the purpose of the expression of the description of the sequences of states and activities of the class of tasks and users. Eight instruction types were used to specify the desired behavior of a task during its simulated execution. These specified the following classes of task-oriented behavioral activities:

-inter-activity delay times
-page accessing
-terminal I/O interactions
-other I/O interactions
-accessing series of pages.

Six additional instruction types appeared in the descriptions of the classes of users. These controlled the formation and selection of the instructions comprising the pseudo-task description. This class of instructions was able to perform the following types of simulated activities:

-I/O device allocation and deallocation from pool of units

-specification of shared task pages
-construction of task descriptions according to given structures.

By the manipulation of a few parameters to the model describing the classes of users to be considered, the workload may be readily changed to reflect different proportions of types of use. This method looked quite good to this researcher, and Nielsen's simulator was actually run (with data supplied by Nielsen). However, the amount of work required to produce an accurate set of descriptions of a different environment was prohibitive and the simulator organization seemed inappropriate for general time-sharing system studies of non-fixed-page computer systems, such as the B5700. Hence, another possible solution method was put aside.

Nielsen, having apparently realized the same problems were hampering the use of his earlier simulator, redesigned and rewrote the original simulator under contract with Burroughs to study the proposed design of a time-sharing system for the B6700 computer system (Nielsen, 69, Nielsen, 70, Nielsen, 71). In this simulator he used a much more simple approach to task description. Distributions are collected reflecting the following:

-attributes of tasks
-execution time between I/O operations
-execution time between terminal I/O operations
-size of I/O operations
-think time of users at terminals
-total task execution time
-main storage size requirements
-nature of overlay requirements
-size of overlay segments.

Each job step is then characterized by a series of task descriptions, which are specified as successive attribute distributions of the foregoing classes. There can be as many distributions as desired. By numbering each distribution, they can be easily referred to for task description purposes and they can be shared for similar descriptions. When a task belonging to a particular job class type is to be originated, its specific nature is constructed by referring to the various distributions and parameters of the system and job step class. Thus even tasks from the same class will not be identical. This process produces roughly the same type of task description as does Nielsen's earlier simulator. This approach looked promising for the current research. A very early and basic copy of his SIMULA model was run on the B5700 with data describing B6700 conditions and appeared satisfactory as a very basic starting point, though a very substantial amount of work was required to convert it to a form acceptable to this research. In fact, the final simulator does not work like Nielsen's simulator, and the source listing only vaguely resembles it. Hence, this researcher feels that it is truly a new simulator.

Another program was then written to accept the trace output of SNOOPY and to convert it to a form from which the necessary distributions could be derived for each job step class. (The program was named LUCY, since LUCY often is the psychiatrist (analyst) of SNOOPY and the other characters of the PEANUTS comic strip.) LUCY is cap-

able of producing analyses of the SNOOPY traces in terms of job step profiles by any of the classes of data present in the trace records. By running SNOOPY on a sufficient number of representative job steps of interest and analyzing the results with LUCY, performance information was gathered. File I/O information not produceable by SNOOPY and LUCY was derived from observing the source listings and correlating them with LUCY output. Other distributions were derived by diverse means, such as the use of the STATISTICS options of the B5700 MCP and TSSMCP and the analysis of mass storage and tape usage entries present in the accounting log with respect to number of records and open time (Burroughs, 71). By these means the necessary distributions were gathered together for use in the simulator.

Much of the other environmental and task data for the simulator could be entered parametrically; however, description of the specific, necessary details of hardware and software structures on the B5700 required significant changes in the original simulator itself. For instance, modeled B6700 I/O multiplexor structure allows for several multiplexors with I/O peripheral controllers attached to as many multiplexors as desired and as many I/O devices as desired attached to an I/O peripheral controller. This modeled structure is not convenient for the modeling of the B5700, which has up to four I/O multiplexors, any of which can service any I/O device on a floating basis. Thus I/O structure in the simulator was changed to reflect that of the B5700.

CPU scheduling in the simulator was also changed to reflect that of the B5700 TSSMCP described earlier in this research (Burroughs, 72). Several other changes were made to transform the B6700 Time Sharing System simulator into a B5700 Time Sharing System simulator. Unfortunately, after these changes and many other major revisions were made, the speed of the simulator was distressingly slow and responded little to any attempts to speed it up. Whereas Nielsen reported times of five to ten minutes to simulate one minute of time, now sixty to ninety minutes were required (Nielsen, 71). Thus an entire rewrite was initiated, employing several major sets of revisions to the algorithms to speed them up and gather more information than was originally recorded and reported, and using U1108 SIMULA rather than B5700 SIMULA. When this effort was completed, the times (on the U1108) were quite significantly better. Whereas the B5700 simulator required twenty minutes to compile and sixty to simulate one minute of time, the U1108-based simulator required one minute to compile and collect (link-edit) and two-to-three minutes to simulate one minute of time. The simulator itself consisted of about 3350 source card images and occupied about 57,000 words of U1108 main storage out of 65,384 words possible maximum size. Data occupied over 375 card images, and each separate run required from one to fifty card image changes.

This improvement was immediately followed by the effort to verify that the new simulator did really model the B5700. This verifi-

cation was made using the job step profiles already gathered and such parameters that the simulation was organized as similarly as possible to the organization of the actual B5700 computer system when statistics and job step traces were gathered. The method by which the detailed simulated traces are reconstructed is discussed later in this chapter. Those attributes of the system which were measured by observation of the actual system and were displayed on the output of the simulator matched quite closely. The most important correlations appear in the list below.

Parameter	System	Simulation
-CPU utilization	12%	15%
-I/O utilization	44%	49%
-I/O 1	76%	80%
-I/O 2	22%	18%
-I/O 3	1.5%	2%
-I/O 4	0.5%	0%
-(disk I/O)/(all I/O)	78%	82%
-CPU/elapsed time	9%	11%
-system utilization	75%	80%
-average swapping rate	0.30/second	0.36/second
-main storage utilization	30%	35%

Other parameters displayed by the simulator but not easily measurable on the actual system were checked for reasonability. By these means, the operations of the simulator simulating the B5700 system were experimentally verified.

B5700 TSSMCP Characteristics

The B5700 Time Sharing System as modeled and analyzed in this

research is based on the B5700 hardware and Time Sharing System Master Control Program (TSSMCP), plus associated support programs. The B5700 hardware is quite different from most other hardware systems in several manners. Its main storage is limited to 32,768 words, each of which has 48 bits, divided into eight 6-bit characters. However, facilities in the hardware and software implement a variable-size-page-on-demand strategy, increasing the virtual main storage size to almost one million words per program. The I/O paths are composed of up to four multiplexors, any of which may be attached to any of the up to 32 different I/O devices on a fully floating basis. A memory exchange allows up to six simultaneous accesses to main storage, though only one simultaneous access per 4,096 word memory module is allowed. One or two central processing units (CPU's) are allowed, though one CPU only is assumed in this research. Preliminary investigations, including extra measurement and simulations, showed that the system could not effectively use a second CPU when running under the TSSMCP.

Main storage is partitioned into two primary areas, normally assumed equal in size. The lower area contains the resident and transient portions of the TSSMCP, the command-and-edit interface with remote users (CANDE), and peripheral handlers, such as printer backup and load control routines which spool printer/punch output and card input from and to disk or tape. The upper area contains the user tasks which may be swapped out of and into main storage to and from disk

storage. The division between these areas is termed the fence. The operator may change it, with some difficulty, though the fence is normally never moved from 16,384 words. After running a certain amount of time, a task is interrupted. If it has exhausted its CPU or core quantum limit for the particular time slice, or it attempts to perform a terminal interaction but cannot, or it cannot find enough space to bring in a required segment, it is eligible to be swapped to disk. At some later point in time, it may be swapped back from disk to main storage, perhaps with more main storage assigned to it. Main storage above the fence is allocated in 1,024 word sets, called chunks. The main storage space for a task is composed of a set of one or more contiguous chunks. When swapping is performed, information existing before the swap out is swapped back into the same addresses, though expansion in the number of chunks may occur in either increasing or decreasing memory addresses, unless the task is already occupying all of main storage above the fence.

The formulae for computing the CPU and core quantum limits for each time slice are the following:

T = (((N times 4) plus C minus P) times 8) plus 208
E = T times J

where T, E, N, C, P, and J are as follows:

T = CPU quantum limit in sixtieths of seconds
E = core quantum limit in sixtieths of seconds
N = number of consecutive times the task has been swapped for using up a quantum limit (≤ 7)

C = number of core chunks used by task
P = priority of the task (0 = high, 9 = low)
J = number of tasks currently swapped in.

As a consequence of the nature of the hardware and software of the B5700 operating under the TSSMCP, several items arise which may not be apparent from the preceding discussion. One major result is that normally only two or three tasks are present in main storage above the fence, thus limiting multiprogramming and making second processor usage ineffective in general. Another result is that the single I/O path to the disk causes the disk to be a limiting resource in many cases, comparable to the slowness of the CPU's and core and the small size of the main storage. Because of this, optimizing the I/O channel usage semi-optimizes the disk usage by the system.

The Simulator

General

Corresponding to that of the microscheduler which is the object of this research, the simulation's level of detail and point of view are represented by the following quantities:

-task state vector
-main storage segment and time-sharing chunk length
-I/O transaction
-CPU interrupt processing interval
-mass storage segment length.

The following quantities are at a higher level than the microscheduler and are represented in the simulation parametrically or algorithmically:

-operating system architecture (except microscheduler)
-macroscheduler
-workload characteristics (job step profiles)
-main storage requirements
-management policies and goals
-system profit/cost functions (GSEM)
-hardware configuration.

The following quantities are represented in the simulation parametrically or implicitly since they are on a lower level than that of the microscheduler:

-dynamic and static address relocation delays
-exact structure of tasks and job steps.

The following quantities are explicitly ignored in the simulation as having an implicit effect on the microscheduler in this case:

-multitasking interference in hardware
-exact set of CPU and I/O instructions available.

Main storage segment size is assumed variable and is computed from one of the task profile distributions when required. The time-sharing chunk size is set at 1024 words to correspond to that used by the B5700 TSSMCP main storage allocation algorithm. The basic CPU interrupt processing interval is assumed to be one millisecond, although that value is never used as such in the simulator; the millisecond is used only as a unit to simplify input and output of parameters.

The CPU, I/O, and main storage facilities were the ones monitored most carefully in this research for inclusion in the LSEM. In other cases, other facilities may be required to be carefully monitored for similar use.

Reconstruction of Task Characteristics

The simulator uses the information concerning each task stereotype to perform the macroscheduling and microscheduling of the workload in such a manner that certain statistical properties of a simulated environment may be carried over into the simulation. The workload was analyzed from log accounting data and decomposed into the following set of major classes:

-compilers and interpreters
-peripheral handlers
-student executions
-file maintenance
-system programming
-production.

Those major classes were then further subdivided into subclasses, producing a number of task stereotypes. Each of these was analyzed by the means described earlier to produce the following information for each task stereotype:

-distribution of time between file I/O's
-distribution of time between originator I/O's
-distribution of file sizes in records
-distribution of delay times for originator input operations
-distribution of task execution times
-distribution of task sizes in chunks
-distribution of segment sizes in words
-distribution of time between segment requests as function of ratio of task's current main storage size to current allocated size
-number of times to repeat operation queue before regeneration
-number of entries to place on operation queue on each regeneration
-for each I/O device pool, the number of I/O devices required and the percentage of task's file I/O requests which will refer to those devices

-priority associated with task stereotype.

The term "originator," as used in this research report, refers to a logical or physical device which presents tasks to the system to be executed and accepts documentary information pertaining to the tasks which have been executed. A send-receive remote terminal would satisfy the definition of originator.

The relative number of occurrences of the tasks from each subclass in the workload was entered to the simulator in the form of distributions for each case considered. Detailed distributions may be found later in this research. When a task is to be selected, this distribution is consulted to determine which of the task stereotypes is to be included in the simulated mix. Then the information described above is used to generate an initial operation queue for that task stereotype. By the method of generation, the operation queues of tasks from the same stereotype will be similar, but not identical. This operation queue is a reconstructed representation of the traces of the operations of tasks belonging to that stereotype, describing in detail the file I/O operations, originator I/O operations, computation periods, delay times, overlay operations, and main storage expansions to be performed by that task in the next few seconds of simulated time. The use of the queue is controlled by parameters of the task stereotype described above. This forms a variant of the trace-driven modeling as used by many other investigators in operating systems investigations (Cheng, 69, Noe, 70,

Noe, 71, Noe, 72, Sherman, 71). The correctness of this technique was proven in the phase of the investigation involved with the validation of the simulation model.

The following is intended to explain the means by which the traces represented by given task stereotype classes are regenerated, by means of an example. Once a task stereotype has been selected through external scheduling, the relevant parameters and distributions are consulted to generate an operation queue. Distributions are stated in terms of constant, increment, and weighting histogram. Assume the following sample data pertaining to the task selected:

-40 items to place on operation queue
-30 times to repeat operation queue before it is regenerated
-priority of task is 2
-half the explicit I/O operations of the task reference the card reader and the other half reference the printer
-card reader speed is 1400 cpm
-printer speed is 1100 lpm
-I/O speed is 1 word per 8.33 microseconds (maximum)
-distribution of time between file I/O's (in milliseconds):
    constant = 10; increment = 5;
    histogram = .2, .3, .6, .8, 1.0
-distribution of time between originator I/O's:
    constant = 125; increment = 75;
    histogram = .15, .30, .60, .80, .90, .95, 1.0
-distribution of file sizes (in records):
    constant = 0; increment = 150;
    histogram = .5, 1.0
-distribution of delay times for originator input operations:
    constant = 1000; increment = 500;
    histogram = .1, .2, .35, .65, .80, .90, 1.0
-distribution of task execution times:
    constant = 3500; increment = 1000;
    histogram = .2, .5, .8, 1.0

-distribution of task sizes (in chunks):
constant = 5; increment = 1;
histogram = .15, .35, .65, .85, 1.0
-distribution of segment sizes (in words):
constant = 50; increment = 50;
histogram = .2, .6, .75, .85, .90, .95, 1.0
-distribution of time between segment requests as function of ratio of task's current main storage size to current allocated size:
constant = 70; increment = 30;
histogram = .05, .30, .70, 1.0.

Using the first five distributions directly and the last three indirectly, the operation queue is constructed and might appear as follows:

-compute for 15 milliseconds
-read from card reader
-compute for 75 milliseconds
-read from terminal, delay 2.5 seconds
-compute for 30 milliseconds
-write on terminal
-compute for 45 milliseconds
-write on printer
.
.
.
-(repeat 30 times up to 40th item)
-(regenerate queue).

As noted previously, this closely resembles actual trace-driven output from other researchers' work (Noe, 71, Sherman, 71, Sherman, 72).

During experimentation, in order to determine how sensitive the results of the simulation were to small changes in the workload, the starting random number in the process used to select new tasks was effectively changed, giving a statistically equivalent but different workload. The results proved quite stable, varying only slightly when this was done, as is demonstrated in tabular form. When a different, batch-

oriented distribution was used, the results varied somewhat more, as demonstrated in the section describing the results of the experiments.

Verification of Microscheduler

In order to verify that the new microscheduler does have the potential of improving the performance with respect to at least one measure, of a computer operating system that employs it versus one that does not, the simulator described in the previous sections was structured to run a number of experiments using the task and system profiles and parameters derived as described earlier. With this microscheduler embodied in the simulator, comparisons were made between it and other prototypical microschedulers under several system effectiveness measures actually used in reality. The purpose of this section is to describe the experiments which empirically verified the effectiveness of the microscheduler.

One of the important items considered during final experimentation was the operating system overhead due to each microscheduling philosophy. Kernels were constructed for each scheduler in SIMULA in order to rank and quantify them. Generally, they were ranked as follows, with respect to increasing overhead, for each current task in the system:

-round-robin
-FIFO
-B5700
-instantaneous exponential

-complete history
-adaptive microscheduler.

SIMULA was chosen for the kernels since it resembles the B5700 ALGOL dialect ESPOL in which the B5700 TSSMCP was written. For other cases, similar kernels could be written and analyzed in assembly language of particular computer systems.

A large number of experiments were performed with the simulator as equipped with the various microscheduling philosophies and system effectiveness measures to explore the characteristics of the adaptivity of the new microscheduler under given workloads.

As noted earlier, some of these were involved with variation of initial values and bounds for A, B, and the time of application, and for initial historical information values, along with the determination of the exact nature of the predictor-corrector portion of the method, such as the method of changing A, B, and time of application, and of the optimal setting of the corrector threshold. It was determined that bounds on A should include the range (.35, .45) and B should include the range (.25, .35) for best operation in the measured workload. For a batch-oriented workload, the bounds on A should include (.40, .50) and on B should include (.30, .40). In either case, a threshold of 0.1 and DA of 0.1 and DB of 0.05 were found optimal for either workload in this environment. The fixing of A or B at a given value, or in a given range outside of those mentioned above, thus removing some of the adaptivity

from the method, caused the MAD to increase significantly for a run and the measure values to decrease correspondingly.

The threshold and DA, DB values were tuned by independent variation and experimentation. Making DA and DB functions of the error committed in the last time interval of application seemed intuitively promising, but showed no significant improvement over constant DA and DB, even under several different thresholds. In particular, the tables below demonstrate some of the results of a number of experiments investigating variable DA dependent upon the error committed. In both cases, the system measure used was THROUGHPUT. Also, in both cases, the threshold was set to 0.1, and the changes made were linear.

B5700 Workload	
DA	THROUGHPUT
.090	.255
.095	.257
.100	.259*
.105	.240
.110	.223

Batch-Oriented Workload	
DA	THROUGHPUT
.090	.0222
.095	.0223
.098	.0223*
.100	.0223
.105	.0221
.110	.0218

In comparison with fixed DA = 0.1, variable DA, all else constant, provided a THROUGHPUT measure at best slightly worse than that provided by a fixed DA. The best values provided by variable DA are marked in the table above with asterisks.

An important result for this environment was that it is not profitable to perform a predictor-corrector except at those points at which

it is known that the characteristics of the task mix are changing, such as on a storage swap in or out or main storage expansion. Separate experiments were performed to check this in which the time interval was varied from one millisecond to ten minutes; no significant difference was noted. This is fortunate, for it leads to lower overhead; such may not be the case in other environments.

The original version of the predictor-corrector portion of the adaptive microscheduler contained the assumption that the time between applications would be so nearly constant that interpolation would be unnecessary. When it was determined that such would not be the case, a linear interpolation algorithm was inserted into the predictor-corrector algorithm to make the predictions more accurate. Though it increased the overhead slightly, it improved the accuracy of prediction about 10 per cent for the current environment, and thus was well worth the small increase in overhead.

Other adaptive microscheduling philosophies were compared with the adaptive one of this research on an experimental basis. One of them was based upon Denning's nonadaptive moving-window approach to working set determination (Denning, 68, Denning, 71B). It was made adaptive by varying the size of the moving window depending upon the size of the error between predicted and actual task/resource usages. Another was based upon the nonadaptive stochastic estimator of Eisenstein which includes a penalty function which is intended to control

the prediction process in the presence of data with trends (Denning, 71B, Eisenstein, 70). It was made adaptive by varying the coefficient of the penalty function depending upon the size of the error produced between predicted and actual usages. After tuning experiments were performed, each performed about as well as the nonadaptive exponential estimator with ALPHA = 0.5, but far short of the performance of the adaptive method of this research.

One of the important aspects of the final experimentation was the question of adequate simulated run time versus cost of simulation runs. Since the run time of the nonadaptive simulation cases was about two-to-three minutes to simulate one minute of time and that of the adaptive cases was somewhat worse, the run time required for statistically significant results was a critical question because of possible budgetary limitations. Through consultation with a statistician and study of stochastic processes, ten-minute simulation runs were set up in which the measure function values were monitored every two seconds, with the first half minute of simulated run time discarded. After three minutes of monitored simulation time, the cumulative means of the measure function values varied insignificantly up to ten minutes of simulated time. Hence, three minutes of measured simulated time was used as a standard for subsequent final runs. The statistical basis for being able to perform one longer experiment (per system effectiveness measure-microscheduler combination), rather than many shorter ones, is

ergodicity.

The small problem of ensuring accuracy of the intermediate measurements is overcome by allowing the accumulation of partial completions of tasks. For resource utilizations, this presents no difficulties, since they may naturally be stated in percentages. For the throughput measure, the simulation has already selected a CPU time requirement for each task, and that is used as a basis for determining the completion factor of each task, in terms of

((CPU usage for task i) divided by (CPU limit for task i)).

Final experimentation consisted of simulation of each of the microscheduling philosophies with respect to each of the system effectiveness measures under two statistically different workloads. This entailed a quite considerable number of executions of the simulator, in terms of investigating and tuning runs. Each complete set of experiments required one run per nonadaptive microscheduler, plus one run for each system measure considered for the adaptive microscheduler. Budgetary and other restrictions prohibited the consideration of large numbers of other cases; however, the cases considered should be representative of many other interesting cases, and the amount of current experimentation is felt to be quite acceptable as far as accuracy of results is concerned.

The following tables present the results of these experiments. The first set of experiments was performed using the workload as ac-

tually measured on the B5700 at Georgia Tech, while the second used a batch-oriented workload. The first set of experiments was repeated with different initial random numbers in order to test the sensitivity of the results to the exact mix of task stereotypes; the percentage results held constant in all cases to within 3 per cent; the adaptive microscheduler was still superior in every case; and no major changes were noted in terms of intra-nonadaptive microscheduler comparisons. The batch-oriented task stereotype mix produced different results, notably an improvement in round-robin performance, as expected for a more CPU-bound system. By frequent monitoring, it was observed that the cumulative means of the system measure values by the end of three minutes of measured simulated time were very stable, as in the case of the other workload.

As expected, and as shown from preliminary results, the adaptive microscheduler's actions caused the simulated operating system containing it to receive somewhat better system objective measure values than did those containing the nonadaptive microschedulers. The results are scaled so that the values corresponding to the application of the adaptive microscheduler are unity, for ease of comparison. The actual values produced by the program are such that they are meaningful only as ratios.

Explanation of Workload Analyses

The following tables present the primary results of this research. The various microschedulers are represented by the columns of the tables and the various global system effectiveness measures (GSEM's) considered are represented by the rows. For each microscheduler/GSEM combination, there are two numbers. The first is (the value of the GSEM) divided by (the value of the GSEM for the AMS), and the second is the value of the GSEM. In the comparison column, the first column represents the improvement realized by the AMS over the best competing microscheduler, and the second represents the improvement over the worst.

Explanation of Workload Distributions

The first column presents the relative weights of the given classes of tasks, and the second column presents the cumulative weights. Ordering of the tables is irrelevant. Data for the first table was drawn from historical accounting data. Data for the second table was artificially generated to represent a batch-production-oriented workload.

Table 2. B5700 Workload Analysis

Name	B5700 TSS	Round Robin	FIFO	Inst Exp Est	Compl Hist	AMS	Comparisons Best MS	Comparisons Worst MS
THROUGHPUT	.79	.69	.69	.77	.73	1.00	21%	31%
	.21	.18	.18	.20	.19	.26		
CPU	.78	.76	.76	.81	.78	1.00	19%	24%
	.25	.24	.24	.26	.25	.32		
REVENUE	.81	.78	.80	.85	.81	1.00	15%	22%
	.21	.20	.20	.22	.21	.26		
RESOURCE	.86	.81	.81	.88	.85	1.00	12%	19%
	.22	.21	.21	.23	.22	.26		
I/O	.89	.77	.88	.90	.88	1.00	10%	23%
	.079	.068	.079	.080	.079	.089		
LATENCY	.90	.94	.96	.91	.87	1.00	4%	13%
	.83	.86	.88	.84	.80	.92		
UTILIZATION	.64	.85	.82	.64	.69	1.00	15%	36%
	.52	.69	.67	.52	.56	.81		
COST	.81	.80	.77	.87	.81	1.00	13%	23%
	.29	.28	.27	.30	.29	.35		

Table 3. B5700 Workload Analysis
(Different Random Numbers)

Name	B5700 TSS	Round Robin	FIFO	Inst Exp Est	Compl Hist	AMS	Comparisons Best MS	Comparisons Worst MS
THROUGHPUT	.80	.70	.70	.76	.73	1.00	20%	30%
	.21	.18	.18	.20	.19	.26		
CPU	.77	.76	.76	.80	.78	1.00	20%	24%
	.25	.24	.24	.26	.25	.32		
REVENUE	.80	.79	.80	.84	.81	1.00	16%	21%
	.20	.20	.20	.22	.21	.26		
RESOURCE	.87	.81	.80	.87	.85	1.00	13%	20%
	.22	.21	.20	.23	.22	.26		
I/O	.88	.78	.87	.90	.87	1.00	10%	22%
	.078	.069	.077	.081	.078	.089		
LATENCY	.91	.95	.95	.91	.87	1.00	5%	13%
	.83	.86	.87	.84	.80	.91		
UTILIZATION	.65	.85	.83	.64	.68	1.00	15%	36%
	.52	.69	.67	.52	.56	.81		
COST	.80	.80	.76	.87	.82	1.00	13%	24%
	.29	.28	.27	.30	.29	.35		

Table 4. Batch-Production-Oriented Workload Analysis

Name	B5700 TSS	Round Robin	FIFO	Inst Exp Est	Compl Hist	AMS	Comparisons Best MS	Comparisons Worst MS
THROUGHPUT	.65 .015	.78 .018	.65 .015	.74 .017	.80 .018	1.00 .022	20%	35%
CPU	.74 .17	.89 .20	.75 .17	.82 .19	.75 .17	1.00 .23	11%	26%
REVENUE	.75 .15	.86 .17	.75 .15	.80 .16	.76 .15	1.00 .20	14%	25%
RESOURCE	.78 .17	.89 .20	.82 .18	.86 .19	.79 .17	1.00 .22	11%	22%
I/O	.86 .076	.92 .080	.90 .078	.85 .074	.88 .076	1.00 .087	8%	15%
LATENCY	.85 .72	.91 .77	.88 .75	.85 .72	.86 .73	1.00 .85	9%	15%
UTILIZATION	.74 .37	.84 .42	.67 .33	.62 .31	.65 .32	1.00 .50	16%	38%
COST	.78 .22	.90 .26	.84 .25	.90 .26	.77 .22	1.00 .28	10%	23%

Table 5. B5700 Workload Distribution
(Number of Tasks)

Small ALGOL/GTL	.15	.15
Medium ALGOL/GTL	.05	.20
Large ALGOL/GTL	.01	.21
Small COBOL	.03	.24
Medium COBOL	.02	.26
Large COBOL	.01	.27
DYNAMO	.01	.28
Small FORTRAN	.10	.38
Medium FORTRAN	.05	.43
Large FORTRAN	.01	.44
Small Execute I/O	.10	.54
Small Execute CPU	.12	.66
Large Execute I/O	.02	.68
Large Execute CPU	.01	.69
Small File Maintenance	.01	.70
Large File Maintenance	.02	.72
Systems Programs	.02	.74
Tape/Disk-Printer	.26	1.00

Table 6. Batch-Production-Oriented Workload Distribution (Number of Tasks)

Large CPU Bound	.15	.15
Large I/O Bound	.15	.30
Large COBOL Compilations	.15	.45
Large FORTRAN Compilations	.15	.60
Large File Maintenance	.20	.80
Large ALGOL Compilations	.20	1.00

CHAPTER V

CONCLUDING REMARKS

Summary

The objective of this research was the development and verification of a new model of optimizing computer performance through the use of adaptive internal resource allocation scheduling techniques. The method of attack entailed the detailed design of such a technique, along with its verification in a given environment through the use of simulation. This technique is based upon the definition of a local system effectiveness function connecting the relevant variables of the system according to a policy describing the system objective function. This local measure is then used as a goal toward which the internal scheduler may work, using a parametric moving-window estimator in the form of a second-order exponential estimator, predictor-corrector trend correction, and variable-period correction application. The simulation was validated with respect to data drawn from an actual system under a real workload, and the new scheduling model was compared to several others under various system objective functions.

Although no general conclusions can be based on experimental results, it is felt that the results have shown that, in the cases tested,

the microscheduler did perform as expected against a variety of representative microscheduling philosophies and under a variety of system effectiveness measures. The only conclusions which can be drawn are concerning the validity of the model for those measures and in the specified environment. Further research and/or actual implementation would be required to base final general conclusions concerning the activities of this model, and these are the topics of the following sections.

Further Research

As with any detailed research in a field as rich and deep as this one, this research has opened several new questions and research topics. The question of the optimal tradeoffs between simplicity, sophistication and further optimization of microscheduling models is discussed in the next section. The effect of priorities, additional CPU's and data paths on the various measure values, especially that of system cost nullification, is complicated but would be quite interesting under the use of the microscheduler of this research. If a matrix of system cost nullification values for (number of CPU's) versus (number of data paths) versus (number of chunks of main storage) were investigated for reasonably small values of each, an optimum hardware configuration for those resources could be determined in specific cases. This type of research should be more practical, accurate, and general than that of a similar nature done by Wald with analytic modeling (Wald, 67). Another possible area of research would be the investigation of modifications to the

approach of the microscheduling of facilities with such multiple resources that the anomalies such as those mentioned by Graham could not occur (Graham, 71). These are pathological situations with respect to system performance in that adding more resources to be controlled by a facility actually decreases system performance with respect to throughput measures. Still another large area of research would be the development of more automatic means by which the software or hardware monitor trace output derived from a computer system running an actual workload could be used to determine the job step profiles for the type of simulator used here. One also wonders what classes of simulators or other analytical tools would be superior to that used in this research. In these respects a very significant amount of tedious manual and programming work was required to perform the tasks in this research of workload characterizations and job step profile characterization and analysis. Is there not a better way?

Other possibly-fruitful areas of further research lie in the improvement in the predictive powers of the adaptive scheduler without increasing its overhead or sophistication and in the increase in the generality of the environment in which the scheduler is assumed to operate. These areas are highly interrelated since changes in the environment will cause changes in the algorithms in the areas referenced previously, such as variable time of application, optimum ranges for the parameters of the predictor-corrector process, configuration of the algo-

rithm itself, etc. The simulator in its current form is somewhat firmly attached to the B5700 Time Sharing System philosophy, though certain areas of interest could be investigated with some amount of difficulty. This includes the use of adaptive CPU and main storage quanta and other main storage allocation strategies, rather than that used by the actual system, as is currently represented in the simulator. Other core-swapping systems, such as that represented by OS/360/MVT-TSO and U1108 EXEC 8, could be simulated with that simulator only with considerable difficulty, and paging systems would be even more difficult to simulate with that simulator, though other simulators are available for the analysis of paging systems, such as Nielsen's original one (Nielsen, 67). There is some intuitive evidence that systems with very large main storage, thus allowing a large number of tasks concurrently in execution, may benefit to a large extent from adaptive microscheduling techniques.

In a somewhat different area of extension, Denning's working set techniques could be implemented with adaptive techniques similar to those discussed here. Some work in that region has been done already by Denning and others (Denning, 72, Morris, 72). Early implementations of the working set concepts have shown its worth but have not generally been successful except in specific situations (Burroughs, 69).

As noted several times earlier, the system effectiveness mea-

sure function is assumed optimized to the extent that the proper choice is made by the microscheduler at the largest possible number of decision points. A tempting extension to the model would be to improve the microscheduler's decision function, perhaps in some radical fashion, such as periodically running a linear programming analysis of current tasks versus resources. Such extensions must be carefully planned and analyzed, for the productivity gained by so much difficulty may be more than lost through an improper microscheduler.

The law of diminishing returns applies to the improvement of schedulers (Denning, 71B). With a reasonable amount of sophistication and a small amount of resources devoted to it, a microscheduler can be constructed which displays a considerable improvement over a basic microscheduler. Each increment of improvement after the first requires disproportionately larger investments in sophistication, time, and resources. Thus any improvements in an already good decision function may be singularly difficult to make. Ryder's paper is a relevant warning to those who would use extremely sophisticated microschedulers (Ryder, 71). As noted previously, his microscheduler algorithms required much of the CPU resource; so much, in fact, that in one environment, a commercial one, the system ran more slowly in elapsed time with his scheduler than without, for the same mix, a reduction in throughput. Such problems cannot be foreseen before actual implementation or careful simulation. Denning notes that the relation-

ship of sophistication to cost of microschedulers is almost unexplored and may be quite treacherous (Denning, 71B).

Methods of Implementation

Those means used to actually produce a microscheduler which controls an operating system must be carefully chosen. The design of the implementation must be executed in coordination with expected or measured environmental data. Normally, only the most critical resources of a system should be optimized; this may usually be done by optimizing the CPU main storage, and I/O resource allocation techniques, for these are almost always the scarcest and most expensive resources of the system.

The configuration of the LSEM must be carefully considered. Some of its terms should have correspondences in the system objective function. Others may be chosen to bias the system toward specific user habits such as smaller tasks. The bounds, and increments of the exponential estimator parameters A and B must also be considered. For A or B too near 1, the microscheduler may be unstable, while for A or B too near 0, it may be unresponsive to changing conditions in the environment. For this research, certain initial values, bounds, and increments for A and B and values for the threshold were used with success. In other environments, other values may be required.

In summary, the implementor of this microscheduling technique

in a real situation must attempt to maintain simplicity, while also attempting to accurately portray the attributes of the environment in which it is expected to function. Many current microschedulers implemented in operating systems are so poor in actual operation (even though they may be very complex) that even a microscheduler embodying only some of the concepts discussed in this research will probably succeed admirably in improving computer system performance (Sherman, 71, Sherman, 72, Wulf, 67).

BIBLIOGRAPHY

ADR. Systems Analysis Machine--Introductory Guide. Applied Data Research, April 1970.

Abate, J., & Dubner, H. "Optimizing the Performance of a Drum-like Storage," IEEE Transactions on Computers (November 1969), 992-997.

Abell, V. A., Rosen, S., & Wagner, R. E. "Scheduling in a General Purpose Operating System," FJCC (1970), 89-96.

Adiri, I. "Computer Time-Sharing Queues with Priorities," JACM (October 1969), 631-645.

Adiri, I., & Avi-Itzhak, B. "A Time Sharing Queue with a Finite Number of Customers," JACM (April 1969), 315-323.

Alderson, A., Lynch, W. C., & Randell, B. "Thrashing in a Multi-programmed Paging System," International Seminar on Operating Systems Techniques, August 1971.

American Standards Association. American Standard Vocabulary for Information Processing, 1966.

Amiot, L., Natarajan, N. K., & Aschenbrenner, R. A. "Evaluating a Remote Batch Processing System," Computer (September-October 1972), 24-30.

Anacker, W., & Wang, C. P. "Performance Evaluation of Computer Systems with Memory Hierarchies," IEEE Transactions on Computers (December 1967), 764-773.

Anderson, H. A., Jr., & Sargent, R. G. "A Statistical Evaluation of the Scheduler of an Experimental Interactive Computing System," Statistical Computer Performance Evaluation, Academic Press, 1972, 73-98.

Apple, C. T. "The Program Monitor--A Device for Program Performance Measurement," Proceedings of Twentieth National ACM Conference (August 1965), 66-75.

Arbuckle, R. A. "Computer Analysis and Thruput Evaluation," Computers and Automation (January 1966), 12-15, 19.

Arden, B., & Boettner, D. "Measurement and Performance of a Multiprogramming System," Proceedings on the 2nd ACM Symposium on Operating System Principles (1970), 130-146.

Armstrong, J., Ulfers, H., Miller, D. J., & Page, H. C. "SOLPASS--A Simulation Oriented Language Programming and Simulation System," Proceedings Third Conference on Applications of Simulation (1969), 24-37.

Arndt, F. R., & Oliver, G. M. "Hardware Monitoring of Real-Time Computer System Performance," Computer (July-August 1972), 25-38.

Arndt, F. R., & Oliver, G. M. "Hardware Monitoring of Real-Time Computer System Performance," Digest of 1971 IEEF International Computer Society Conference (1971), 123-125.

Auerbach. Auerbach on Time Sharing. Auerbach, 1970.

Baer, J., & Caughey, R. "Segmentation and Optimization of Programs from Cyclic Structure Analysis," SJCC (May 1972), 23-26.

Baer, J. L., & Sager, G. R. "Measurement and Improvement of Program Behavior under Paging Systems," Statistical Computer Performance Evaluation, Academic Press, 1972, 241-264.

Bagley, P. R. "Two Think Pieces: (Item 2: Establishing a Measure of Capability of a Data Processing System)," CACM (January 1960), 1.

Bairstow, J. N. "A Review of Systems Evaluation Packages," Computer Decisions (June 1970), 20.

Baldwin, F. R., Gibson, W. B., & Poland, C. B. "A Multiprocessing Approach to a Large Computer System," IBM Systems Journal (September 1962), 64-76.

Bardy, Y. "Performance Criteria and Measurement for a Time-Sharing System," IBM Systems Journal (July 1971), 193-216.

Baskett, F. "Dependence of Computer System Queues upon Processing Time Distribution and Central Processor Scheduling," SIGOPS (June 1972), 109-113.

Baskett, F. Mathematical Models of Computer Systems. Ph.D. Dissertation, University of Texas at Austin, 1970.

Batson, A., & Brundage, R. "Performance Measurements on a Virtual Memory Computer System in a Batch-Processing Environment," ACM Workshop on Systems Performance Evaluation (April 1971), 214-226.

Batson, A., Ju, S.-M., & Wood, D. C. "Measurements of Segment Size," CACM (March 1970), 155-159; and Proceedings of Second ACM Symposium on Operating Systems (October 1969), 25-29.

Beilner, H., & Waldbaum, G. "Statistical Methodology for Calibrating a Trace-Driven Simulator of a Batch Computer System," Statistical Computer Performance Evaluation, Academic Press, 1972, 423-460.

Beizer, B. "Analytical Techniques for the Statistical Evaluation of Program Running Time," FJCC (1970), 519-524.

Belady, L. A., & Kuehner, C. J. "Dynamic Space-Sharing in Computer Systems," CACM (May 1969), 282-288.

Belady, L. A., Nelson, R. A., & Shedler, G. S. "An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paged Machine," CACM (June 1969), 349-353.

Bell, C. G., Gold, M. M., Steadry, A. C., Linde, R. R., & Chaney, P. E. Time Sharing, AD666730, 1967; or American Data Processing, 1967.

Bell, C. G., & Newell, A. Computer Structures--Readings and Examples, McGraw-Hill, 1971.

Bell, T. E. "Computer Measurement and Evaluation," ACM SIMULETTER (October 1972), 52-58.

Bell, T. E., Boehm, B. W., & Watson, R. A. Computer Performance Analysis--Framework and Initial Phases for a Performance Improvement Effort, RAND, R-549-PR, August 1971.

Bell,T. E. Computer System Measurement and Analysis, RAND Corporation, R-584-NASA/PR, January 1971.

Bemer, R. W., & Ellison, A. L. "Software Instrumentation Systems for Optimum Performance," IFIP (August 1968), C39-C42.

Bernstein, A., Detlefsen, G., & Kerr, R. "Process Control and Communication in a General Purpose Operating System," Second ACM Symposium on Operating Principles, Princeton, N. J. (October 1969), 60-66.

Bernstein, A. J., & Sharp, J. C. "Policy-Driven Scheduler for a Time-Sharing System," CACM (February 1971), 74-78.

Blake, K., & Gordon, G. "Systems Simulation with Digital Computers," IBM Systems Journal (January 1964), 15-16.

Blatny, J., Clark, S. R., & Rourke, T. A. "On the Optimization of the Performance of Time-Sharing Systems by Simulation," CACM (June 1972), 411-420.

Blevins, P. R., & Ramamoorthy, C. V. A Classification and Survey of Computer Performance Evaluation Techniques, University of Texas, Austin, April 1970.

Block, W. R. Paper presented at CUBE (Cooperating Users of Burroughs Equipment) meeting, Spring 1971.

Blunden, G. P., & Krasnow, H. S. "The Process Concept as a Basis for Simulation Modelling," ORSA presentation, 1965.

Boehm, B. W. Computer Systems Analysis Methodology, RAND Corporation, OR-520-NASA, September 1970.

Boehm, B. W., & Mobley, R. L. A Computer Simulation of Adaptive Routing Techniques for Distributed Communications Systems, RAND Corporation, RM-4782-PR, February 1966.

Bonner, A. J. "Using System Monitor Output to Improve Performance," IBM Systems Journal (November 1969), 290-298.

Bookman, P. G., Brotman, B. A., & Schmitt, K. L. "Measurement Engineering Tunes Systems," Computer Decisions (April 1972), 28-32.

Bowlden, H. J. "Two-class Control Algorithm," Westinghouse R & D Memo, 67-1C4-COMP-M84, 1967.

Boyd, D. L., & Epley, D. L. "A Simple Model for Multiple-Resource Allocation in Operating Systems," Computers and Automata (1972), 293-304.

Braddock, D. M., & Dowling, C. B. Simulation Evaluation and Analysis Language, IBM 360 D15, 1.005, 1968.

Bradley, J. "Assessing Operating Systems," Data Automation Digest (September 1968), 15-21.

Brawn, B. S., & Gustavson, F. G. "Program Behavior in a Paging Environment," FJCC (1968), 1019-1032.

Bross, S. D. J. "Models," Management Systems (1968), 327-336.

Bryan, G. E. "20,000 Hours at the Console: A Statistical Summary," FJCC (1967), 769-777.

Bryan, G. E., & Shemer, J. E. "The UTS Terminal System: Performance Analysis and Instrumentation," ACM Second Symposium on Operating Systems Principles (October 1969), 147-158.

Buchholz, W. "A Synthetic Job for Measuring System Performance," IBM Systems Journal (1969), 309-318.

Budd, A. E. A Method for the Evaluation of Software: Volume 2, General Description Including Benchmark Considerations, and, Volume 3, Executive, Operating, or Monitor Systems, TR-197, MITRE Corporation, July 1967.

Burroughs. B5700 Master Control Program Reference Manual, 1971.

Burroughs. B5700 Time Sharing System Information Manual, 1972.

Burroughs. B6700 Information Processing Systems Reference Manual, 1969.

Burroughs. B6700 Master Control Program Reference Manual, 1969.

Burroughs. Narrative Description of the Burroughs B5500 Disk File Master Control Program, 1967.

Bussell, B., & Kaster, R. A. "Instrumenting Computer Systems and Their Programs," FJCC (1970), 525-534.

Buxton, J. N. "Writing Simulations in CSL," Computer Journal (August 1966), 137-143.

Buxton, J. N., & Laski, J. G. "Control and Simulation Language," Computer Journal (August 1966), 137-143.

Buzen, J. "Analysis of System Bottlenecks Using a Queueing Network Model," ACM SIGOPS Workshop on System Performance Evaluation (April 1971), 82-103.

CACI. SIMSCRIPT II.5 Reference Handbook, CACI, 1971.

CDC. CDC 6000 Series SIMSCRIPT Reference Manual, CDC Publication No. 60178300, 1968.

Calingaert, P. "System Performance Evaluation: Survey and Appraisal," CACM (January 1967), 12-18.

Campbell, D. J., & Heffner, W. J. "Measurement and Analysis of Large Operating Systems during System Development," FJCC Part I (1968), 903-914.

Canning, R. G. "Data Processing Planning Via Simulation," EDP Analyzer (April 1968).

Canning, R. G. "Improve Operation with Multi-Programming?" EDP Analyzer (March 1966).

Cantrell, H. N. "Designing for Measurement," Digest of 1971 IEEE International Computer Society Conference (1971), 125-126.

Cantrell, H. N. Presentation at Workshop or Models for Time-Shared Processing, IEEE Communications Technology Group and IEEE Computer Group (January 1967).

Cantrell, H. N. Time-Sharing Data GETIS Report R65CD12 (December 1965).

Cantrell, H. N., & Ellison, A. C. "Multiprogramming System Performance Measurement and Analysis," SJCC (1968), 213-221.

Chang, W. "A Queuing Model for a Simple Case of Time Sharing," IBM Systems Journal (April 1966), 115-125.

Chang, W. "Single-Server Queuing Processes in Computing Systems," IBM Systems Journal (1970), 36-71.

Chang, W., & Wong, D. "Analysis of Real Time Multiprogramming," JACM (October 1965), 581-588.

Chang, W., Paternot, Y. J., & Ray, J. A. "Throughput Analysis of Computer Systems," ACM SIGOPS Workshop on Systems Performance Evaluation (April 1971), 59-81.

Cheng, P. S. "Trace-Driven Modeling," IBM Systems Journal (December 1969), 280-289.

Chilcote, W. S. Paper presented at CUBE (Cooperating Users of Burroughs Equipment) meeting, Fall 1970.

Clementson, A. T. "Extended Control and Simulation Language," Computer Journal (November 1966), 215-220.

Codd, E. F. "Multiprogram Scheduling," CACM (March 1960), 347-350 and 413-418.

Codd, E. F. "Multiprogramming STRETCH: A Report on Trials," IFIP (1962), 574.

Coffman, E. G. "Analysis of a Drum Input/Output Queue under Scheduled Operation in a Paged Computer System," JACM (January 1969), 73-90.

Coffman, E. G. "Analysis of Two-Time Sharing Algorithms Designed for Limited Swapping," Journal of the Association of Computing Machinery (October 1968), 549-576.

Coffman, E. G. "On the Trade Off Between Response and Preemption Costs in a Foreground-Background Computer Service Discipline," IEEE Computer Transactions (October 1969), 942-947.

Coffman, E. G. Stochastic Models of Multiple and Time-Shared Computer Operations, Number 66-38, UCLA, June 1966.

Coffman, E. G. "Studying Multiprogramming Systems (with the queueing theory)," Datamation (June 1967), 47-54.

Coffman, E. G., & Denning, P. J. Operating Systems Theory, Prentice-Hall, 1972.

Coffman, E. G., & Graham, R. L. "Optimal Scheduling for Two-Processor Systems," Acta Informatica 2, (1972).

Coffman, E. G., & Kleinrock, L. "Computer Scheduling Methods and their Countermeasures," SJCC (1968), 11-21.

Coffman, E. G., & Kleinrock, L. "Feedback Queuing Models for Time-Shared Systems," JACM (October 1968), 549-576.

Coffman, E. G., & Muntz, R. R. "Models of Pure Time-Sharing Disciplines for Resource Allocation," Proceedings of the 24th National ACM Conference (1969), 217-228.

Coffman, E. G., & Ryan, T. A. "A Study of Storage Partitioning Using a Mathematical Model of Locality," CACM (March 1972), 185-190.

Coffman, E. G., & Varian, L. C. "Further Experimental Data on the Behavior of Programs in a Paging Environment," CACM (July 1968), 471-474.

Coffman, E. G., & Weissman, C. "General Purpose Time Sharing System," SJCC (1964), 397-411.

Coffman, E. G., & Wood, R. C. "Interarrival Statistics for Time-Sharing Systems," CACM (July 1966), 500-502.

Cohen, L. J. System and Software Simulator (S3), Vol. I-IV AD679269-AD679272, 1968.

Cohen, L. J. Systems Analysis and Design, Spartan, 1970.

Cohen, L. J. Theory of the Operating System, UCC Seminar notes, 1967.

COMRESS. Introduction to SCERT, COMRESS, 1967.

COMRESS. The SCERT Reports, COMRESS, 1969.

Constantine, L. "Integrated Hardware/Software Design," Modern Data (September 1968-April 1969).

Conway, R. W., Maxwell, W. L., & Miller, L. W. Theory of Scheduling, Addison Wesley, 1967.

Corbato, F. J., Daggett, M. M., Daley, R. C. "An Experimental Time-Sharing System," SJCC (1962), 335-344.

Critchlow, A. J. "Generalized Multiprocessing and Multiprogrammed Systems," FJCC (1963), 107-126.

Curry, G. L., et al. "Scheduling in a Multiprogramming Environment," Software Age (January 1968), 32-37.

Dahl, O. J., Myhrhaug, B., & Nygaard, K. "Some Features of the SIMULA 67 Language," Second Conference on Applications of Simulation (1968), 29-31.

Dahl, O. J., & Nygaard, K. "SIMULA--An ALGOL-Based Simulation Language," CACM (September 1966), 671-678.

Dahm, D. M., Gerbstadt, F. H., & Pocelli, M. M. "A System Organization for Resource Allocation," CACM (December 1967), 772-779.

Daley, R. C., & Dennis, J. B. "Virtual Memory, Processes, and Sharing in MULTICS," CACM (May 1968), 306-312.

D'Angelo, H., & Windeknecht, T. G. "An Approach to Modeling an Elementary School," Proceedings of the Third Annual Pittsburgh Conference on Modeling and Simulation, April 24-25, 1972.

DeMeis, W. M., & Weizer, N. "Measurement and Analysis of a Demand Paging Time-Sharing System," Proceedings of the 24th National ACM Conference (1969), 201-216.

Deniston, W. R. "SIPE: A TSS/360 Software Measurement Technique," Proceedings of the 24th National ACM Conference (1969) 229-245.

Denning, P. J. "On Modeling Program Behavior," SJCC (1972), 937-944.

Denning, P. J. "Effects of Scheduling on File Memory Operation," SJCC (January 1969), 9-21.

Denning, P. J. "Equipment Configuration in Balanced Computer Systems," IEEE Transactions on Computers (November 1969), 1008-1012.

Denning, P. J. Resource Allocation in Multiprocess Computer Systems. Ph.D. Dissertation, M. I. T., 1968.

Denning, P. J. "A Statistical Model for Console Behavior in Multiuser Computers," CACM (September 1968), 605-612.

Denning, P. J. "Third Generation Computer Systems," Computing Surveys (December 1971), 175-216.

Denning, P. J. "Thrashing: Its Causes and Prevention," FJCC, Part I (1968), 915-922.

Denning, P. J. "Virtual Memory," Computing Surveys (September 1970), 153-190.

Denning, P. J. "The Working Set Model for Program Behavior," CACM (May 1968), 323-333.

Denning, P. J., & Eisenstein, B. A. "Statistical Methods in Performance Evaluation," ACM Workshop on Systems Performance Evaluation (April 1971), 284-307.

Denning, P. J., & Schwartz, S. C. "Properties of the Working-Set Model," CACM (March 1972), 191-198.

Dennis, J. B. Automatic Scheduling of Priority Process. Ph.D. Dissertation, M. I. T., 1964.

Dennis, J. B. "Segmentation and the Design of Multiprogrammed Computer Systems," JACM (October 1965), 589-602; and IEEE Convention Record, Part 3 (1965), 214-225.

De Rivet, P. H. Data Traffic Models for the Performance of Modular Computer Systems. Ph.D. Dissertation, Case Western Reserve University, 1971.

Des Roaches, J. S. Survey of Simulation Languages and Programs, MTR-2040, MITRE, January 1971.

Dewan, P. B., Donaghey, C. E., & Wyatt, J. B. "OSSL--A Specialized Language for Simulating Computer Systems," SJCC (1972), 799-814.

Dijkstra, E. W. "A Class of Allocation Strategies Inducing Bounded Delays Only," SJCC (1972), 933-936.

Dijkstra, E. W. "Complexity Controlled by Hierarchial Ordering of Function and Variability," NATO Conference on Software Engineering (October 1968), 181-185.

Dijkstra, E. W. "The Humble Programmer," CACM (October 1972), 859-866.

Dijkstra, E. W. Notes on Structured Programming, EWD 249, Technical University of Eindhoven, 1969.

Dijkstra, E. W. "Solution of a Problem in Concurrent Programming Control," CACM (1965), 569.

Dijkstra, E. W. "T.H.E. Multiprogramming System," CACM (May 1968), 341-346.

Doherty, W. J. "Scheduling TSS/360 for Responsiveness," FJCC (1970), 97-112.

Donovan, J. J., Alsop, J. W., & Jones, M. M. "A Graphical Facility for an Interactive Simulation System," Proceedings of IFIPS Congress (1968), 593-596.

Dopping, O. "Test Problems Used for Evaluation of Computers," BIT (1962), 197-202.

Dorn, P. H. "How to Evaluate a Time-Sharing Service," Datamation (November 1969), 220-223.

Draper, M. "V1108 EXEC 8 Augmented Accounting Data," Technical Papers of USE Conference, October 12-16, 1970, 421-435.

Draper, M. 1108 EXEC 8 Performance Evaluation and Scheduling. Paper presented to Fall 1970 Univac User's Association meeting.

Drummond, M. E., Jr. "A Perspective on System Performance Evaluation," IBM Systems Journal (1969), 252-263.

Ebeling, D. G. A General Purpose Conversational Time Sharing Program for Probabilistic Analysis. General Electric Corporation, August 1969.

Eisenstein, B. A., & Shen, D. W. C. "Adaptive Stochastic Tracking of Quasi-Repetitive Processes," Proceedings of IFAC Kyoto Symposium, 1970.

Eisenstein, B. A. Decision-Directed Adaptive Estimators for Repetitive Processes. Ph.D. Dissertation, University of Pennsylvania, 1970.

Emshaff, J. R., & Sisson, R. L. Computer Simulation Models, MacMillan, 1970.

Erikson, W. J. "An Analytical Cost Comparison of Computer Operating Systems," SDC-TM-3525 or AD 661 983, June 30, 1967.

Estrin, G., & Kleinrock, L. "Measures, Models, and Measurements for Time-Shared Computer Utilities," Proceedings, 22nd National ACM Meeting (1967), 85-96.

Estrin, G., Muntz, R. R., & Uzgalis, R. "Modeling, Measurement, and Computer Power," SJCC (1972), 725-738.

Estrin, G., Hopkins, D., Coggan, B., & Crocker, S. D. "Snuper Computer: A Computer in Instrumentation Automation," SJCC (1967), 645-656.

Evans, T. G., & Darley, D. L. "On-Line Debugging Techniques: A Survey," FJCC (1966), 37-50.

Famolari, E. FORSIM IV User's Guide SR-99, The MITRE Corporation, February 1964.

Fano, R. M., & Corbato, F. J. "Time-Sharing on Computers," INFORMATION, Scientific American, September 1966.

Feeley, J. M. "A Computer Performance Monitor and Markov Analysis for Multiprocessor System Evaluation," Statistical Computer Performance Evaluation, Academic Press (1972), 165-226.

Fenichel, R. R., & Grossman, A. J. "An Analytic Model of Multiprogrammed Computering," SJCC (1969), 717-721.

Ferdinand, A. E. "An Analysis of the Machine Interference Model," IBM Systems Journal (1971), 129-142.

Ferrari, D. "Workload Characterization and Selection in Computer Performance Measurements," Computer (July-August 1972), 18-24.

Fifield, A. J. "A System for Timing Systems," The Computer Bulletin (November 1968), 256-260.

Fine, G. H., & Mac Isaac, P. V. "Simulation of a Time-Sharing System," Management Science (February 1966), B180-B194.

Fisher, D. A. Program Analysis for Multiprocessing, TR-67-2, Burroughs, May 1967.

Fishman, G. S., & Kiviat, P. J. "The Analysis of Simulation-Generated Time Series," Management Science (March 1967), 525-557.

Fishman, G. S., & Kiviat, P. J. "The Statistics of Discrete-Event Simulation," Simulation (April 1968), 185-195.

Flores, I. "Swapping vs. Paging," Modern Data (April 1970), 152-157.

Flores, I. "Virtual Memory and Paging," Datamation (August 1967), 31-34, and (September 1967), 41-48.

Foley, J. D. "A Markovian Model of the University of Michigan Executive System," CACM (September 1967), 584-588.

Forgie, J. W. "A Time and Memory Sharing Executive Program for Quick Response," FJCC (1965), 599-609.

Fox, D., & Kessler, J. L. "Experiments in Software Modeling," FJCC (November 1967), 429-436.

Fox, R. L. Optimization Methods for Engineering Design, Addison-Wesley, 1971.

Fredrickson, R. M. Estimates of Supervisor Overheads for the IBM S/360/67 Simulation. Stanford University Computation Center Memo, August 8, 1966.

Freeman, D. N., & Pearson, R. R. "Efficiency vs. Responsiveness in a Multiple-services Computer Facility," Proceedings, 23rd National ACM Conference (1968), 25-34.

Freiberger, W. Statistical Computer Performance Evaluation, Academic Press, 1972.

Friedman, C. J. "Constraint Algebra--A Supervisory Programming Technique," IEEE Transactions on Computers, July 1963.

GSA. "Restriction on Use of Simulation for Evaluating ADPE Performance," GSA Federal Property Management Regulations, TR E-23, June 6, 1972.

Gaver, D. P. "Probability Models for Multiprogramming Computer Systems," JACM (July 1967), 423-438.

Gaylord, C. V. "Multiprogramming and the Design of On-line Control Systems," Data Processing Magazine (May 1968), 26-38.

Geisler, M. A., & Markowitz, D. A Brief Review of SIMSCRIPT as a Simulating Technique, RM-3778-PR, Rand, 1963.

Gibson, J. C. The Gibson Mix. Technical Report TR00-2043, IBM, June 1970.

Gibson, C. T. "Time Sharing in the IBM S/360/67," SJCC (1966), 61-78.

Glinka, L. R., Brush, R. M., & Ungar, A. J. "Design Thru Simulation, of a Multiple-Access Information System," FJCC (1967), 437-447.

Gold, M. M. Methodology for Evaluating Time-Sharing Computer Usage. Ph.D. Dissertation, AD 668 084, M. I. T., 1967.

Goodroe, J. R., & Leonard, G. F. "An Environment for an Operating System," Proceedings of the 17th National ACM Conference (1964), pp. E2.3-1--E2.3-11.

Gordon, G. System Simulation, Prentice-Hall, 1969.

Gould, R. L. "GPSS/360--An Improved General Purpose Simulator," IBM Systems Journal (January 1969), 16-27.

Graham, R. L. "Bounds on Multiprocessing Anomalies and Related Packing Algorithms," SJCC (1972), 205-217.

Graham, R. L. "Bounds on Multiprocessing Timing Anomalies," SIAM Journal of Applied Mathematics (April 1969), 416-429.

Graham, R. L. "Bounds for Certain Multiprocessing Anomalies," Bell System Technical Journal (September 1966), 1563-1581.

Greco, R. J. Evaluation of Real Time Computer Systems. Internal Memorandum GITIS-69-11, School of Information and Computer Science, Georgia Institute of Technology, December 1969.

Greenbaum, H. J. A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System, Report No. MACTR-58, M. I. T., 1969.

Greenberg, S. GPSS Primer, Wiley, 1972.

Greenberger, M. "A New Methodology for Computer Simulation," Computer Methods in the Analysis of Large-Scale Social Systems (1965), 147-162.

Greenberger, M. "The Priority Problem and Computer Time-Sharing," Management Science (1966), 888-906.

Greenberger, M., Jones, M. M., Morris, J. H., Jr., & Ness, D. N. On-Line Computation and Simulation: The OPS-3 System, M. I. T., 1965.

Greenberger, M., & Jones, M. M. "On-Line Simulation in the OPS-3 System," Proceedings of the 21st National ACM Conference (1966), 131-138.

Grenander, U., & Tsao, R. F. "Quantitative Methods for Evaluating Computer System Performance: A Review and Proposals," Statistical Computer Performance Evaluation, Academic Press (1972), 3-24.

Grochow, J. M. "Utility Functions for Time-Sharing System Performance Evaluation," Computer (September-October 1972), 16-19.

Grochow, J. M. "A Utility-Theoretic Approach to the Evaluation of a Time-Sharing System," Statistical Computer Performance Evaluation, Academic Press (1972), 25-50.

Habermann, A. N. On the Harmonious Cooperation of Abstract Machines. Ph.D. Dissertation, Eindhoven Technological University, 1967.

Habermann, A. N. "Prevention of System Deadlocks," CACM (July 1969), 373-377.

Hansen, P. B. "The Nucleus of a Multiprogramming System," CACM (April 1968), 74-84.

Hansen, P. B. "Short Term Scheduling in Multiprogramming Systems," SIGOPS (June 1972), 101-105.

Harary, F., et al. Structural Models, Wiley, 1965.

Hare, V. C. System Design, 1967.

Harrison, M. C. "Implementation of the SHARER 2 Time-Sharing System," CACM (December 1968), 845.

Harrison, M. C., & Schwartz, J. T. "SHARER, A Time-Sharing System for the CDC 6600," CACM (October 1967), 659-665.

Hart, L. E. "The Users Guide to Evaluation Products," Datamation (December 15, 1970), 32-35.

Hart, L. E., & Lipovich, G. J. "Choosing a System Stethoscope," Computer Decisions (November 1971), 20-23.

Hartley, D. F., Landy, B., & Needham, R. M. "The Structure of a Multiprogramming Supervisor," Computer Journal (November 1968), 247-255.

Hastings, T. Operating Statistics of the MAC Time-Sharing System, MAC-M-280, M. I. T., December 8, 1965.

Hatfield, D. J. "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance," IBM Journal Research and Development (January 1972), 58-66.

Hatfield, D. J., & Gerald, J. "Locality: Working Set, Request String, Distance Function, and Replacement Stack," Statistical Computer Performance Evaluation, Academic Press (1972), 407-422.

Hauck, E. A., & Dent, B. A. "Burroughs B6500/B7500 Stack Mechanisms," SJCC (1968), 245-251.

Hauth, C. "Turnaround Time for Messages of Differing Priorities," IBM Systems Journal (April 1968), 103-122.

Havender, J. W. "Avoiding Deadlock in Multitasking Systems," IBM Systems Journal (April 1968), 74-84.

Hebalker, P. G. Deadlock-Free Sharing of Resources in Asychronous Systems. Sc.D. Dissertation, MAC TR-75, 1970.

Heller, J., & Legemann, G. "An Algorithm for the Construction and Evaluation of Feasible Schedules," Management Science (January 1962), 168-183.

Hellerman, H. "Complementary Replacement--A Meta Scheduling Principle," ACM Second Symposium on Operating Systems Principles (October 1969), 43-46.

Hellerman, H. Digital Computer System Principles, McGraw-Hill, 1967, 124-125.

Hellerman, H. "Some Principles of Time-Sharing Scheduler Strategies," IBM Systems Journal (April 1969), 94-117.

Hellerman, H., & Smith, H. J. "Throughput Analysis . . . . ," Computing Surveys (June 1970), 111-118.

Herman, D. J. "SCERT: A Computer Evaluation Tool," Datamation (February 1967).

Herman, D. J., & Ihrer, F. C. "The Use of a Computer to Evaluate Computers," SJCC (1964), 383-395.

Highland, H. J. "A Matter of Communication," ACM SIMULETTER (October 1972), 82-93, 100-106.

Hillegass, J. R. "Standardized Benchmark Problems Measure Computer Performance," Computers and Automation (January 1968), 12-15.

Hills, P. R. "SIMON--A Computer Simulation Language in ALGOL," Digital Simulation in Operation Research (1967), 105-115.

Hobgood, W. S. "Evaluation of an Interactive-Batch System Network," IBM Systems Journal (January 1972), 2-15.

Hoffman, J. M. A Pre-Scheduler and Management Model for a Class of Computer-User Systems. Ph.D. Dissertation, January 1971.

Holland, F. C., & Merikallio, R. A. "Simulation of a Multiprocessing System Using GPSS," IEEE Transactions on Systems Science and Cybernetics (November 1968), 395-400.

Holt, A. W., & Commoner, F. Events and Conditions (Parts 1-3), Applied Data Research, Inc., 1969.

Holt, A. W., Saint, H., Shapiro, R. M., & Marshall, S. Final Report for Information System Theory Project--Contract Number AF30 (602)-4211, AD676972, Applied Data Research, Inc., 1968.

Holtwick, G. M. "Designing a Commercial Performance Measurement System," ACM Workshop on Systems Performance Evaluation (April 1971), 29-58.

Hornig, D. F., et al. Computers in Higher Education, U. S. Printing Office: 1967-0-272-973, 1967.

Huesmann, L. R., & Goldberg, R. P. "Evaluating Computer Systems through Simulation," Computer Journal (August 1967), 150-155.

Hughes Aircraft Company. Simulation Models--A Tool for the Development of Real-Time Computer Systems, August 1970.

Huntable, D. H. R., & Warwick, M. T. "Dynamic Supervisors--Their Design and Constructions," ACM Symposium on Operating Systems Principles, preprint, 1967.

Hutchinson, G. K. "A Computer Center Simulation Project," CACM (September 1965), 559-568.

Hutchinson, G. K., & Maguire, J. N. "Computer Systems Design and Analysis through Simulation," FJCC (1965), 161-167.

IBM. IBM System/370 Principles of Operation, GA22-7000-2, July 1972.

IBM. "Functional Structure of OS/360," IBM Systems Journal (April 1966), 2-51.

IBM. GPSS II, IBM, 1963.

IBM. SIMULATION, FORM #520-1538-1, 1968.

IBM. Simulation Evaluation and Analysis Language (SEAL), System Reference Manual, January 17, 1968.

IBM. Time-Sharing System/360 Development Workbook, IBM Internal Document, 1966.

Ihrer, F. C. "Benchmarking vs. Simulation," ACM SIMULETTER (October 1972), 94-99.

Ihrer, F. C. "Computer Performance Projected through Simulation," Computers and Automation (April 1967), 22-27.

ISC. A Brief Introduction to Interactive Sciences Corporation, Interactive Sciences Corporation, 1968.

ISC. Computer Systems--Analysis and Simulation, Interactive Sciences Corporation, 1969.

Jackson, P. E., & Stubbs, C. D. "A Study of Multiaccess Computer Communications," SJCC (1969), 494-504.

Johnson, R. H., & Johnston, T. Y. PROGLOOK Users' Guide (Revision 3), COSMIC-02250 or SLAC User Note 33 or SCC-007, October 1971.

Johnson, R. R. "Needed--A Measure for Measure," Datamation (December 15, 1970), 22-30.

Johnson, R. R. Systems Performance Measurement, Prediction, Optimization--Part 3, presented at Fall 1971 meeting of Cooperating Users of Burroughs Equipment (CUBE).

Jones, M. M. Incremental Simulation on a Time-Shared Computer. Ph.D. Dissertation, MAC-TR-48, 1968.

Jones, M. M. "On-Line Simulation," Proceedings of ACM 1967 National Conference (1967), 591-600.

Jones, M. M. On-Line Version of GPSS II," MAC-M-140, March 1964.

Jones, P. O. "Operating System Structures," IFIP (1968), 29-33.

Jordain, P. B. Condensed Computer Encyclopedia, McGraw-Hill, 1969.

Joslin, E. O. "Application Benchmarks: The Key to Meaningful Computer Evaluations," Proceedings of the 20th National ACM Conference (August 1965), 27-37.

Joslin, E. O. "Cost-Value Technique for Evaluation of Computer System Proposals," SJCC (1964), 367-381.

Joslin, E. O., & Aiken, J. J. "The Validity of Basing Computer Selection on Benchmark Results," Computers and Automation (January 1966), 22-23.

Karr, H. W., Kleine, H., & Markowitz, H. M. SIMSCRIPT 1.5, California Analysis Center, 1966.

Karush, A. D. Benchmark Analysis of Time-Shared Systems, SDC SP-3347, SDC, June 30, 1969.

Karush, A. D. Definition and History of Regenerative Recording, TRN-24317/103/00, SDC, February 1970.

Karush, A. D. "Evaluating Timesharing Systems Using the Benchmark Method," Data Processing Magazine (May 1970), 42-44.

Karush, A. D. "Program Quality Assurance," Datamation (October 1968), 61-66.

Karush, A. D. Towards a Study of the Regenerative Recording of Time-Sharing Systems, TRN-24317/104/00, SDC, 1970.

Karush, A. D. "Two Approaches for Measuring the Performance of Time-Sharing Systems," ACM Second Symposium on Operating System Principles (October 1969), 159-166.

Katz, J. H. "An Experimental Model of IBM S/360," CACM (November 1967), 694-702.

Katz, J. H. "Simulation of a Multiprocessor Computer System," SJCC (1966), 127-139.

Kay, I. M. "An Over-the-Shoulder Look at Discrete Simulation Languages," SJCC, 791-798.

Keefe, D. D. "Hierarchical Control Programs for Systems Evaluation," IBM Systems Journal (April 1968), 123-133.

Kernighan, B. W. "Optimal Segmentation Points for Programs," ACM Second Symposium on Operating Systems Principles (October 1969), 47-53.

Kernighan, B. W. Some Graph Partitioning Problems Related to Program Segmentation. Ph.D. Dissertation, Princeton University, 1969.

Kerr, R., Bernstein, A., Detlefsen, G., & Johnston, J. Overview of R & DC Operating System, TIS Report 69-C-355, General Electric, October 1969.

Kimbleton, S. R. "Performance Evaluation--A Structured Approach," SJCC (1972), 411-416.

Kimbleton, S. R. "Role of Computer Systems Models in Performance Evaluation," CACM (July 1972), 586-590.

Kimbleton, S. R., & Moore, C. G. A Limited Resource Approach to System Performance Evaluation, TR 71-2 ISDOS, University of Michigan, 1971.

Kimbleton, S. R., & Moore, C. G. "A Probabilistic Framework for System Performance Evaluation," ACM SIGOPS Workshop on System Performance Evaluation (April 1971), 337-361.

Kiviat, P. J. GASP--A General Activity Simulation Project, Applied Research Laboratory, 1963.

Kiviat, P. J. "Introduction to the SIMSCRIPT II Programming Language," Symposium on Simulation Techniques and Languages, May 1966.

Kiviat, P. J., & Prittsker, A. A. B. Simulation with GASP II, Prentice-Hall, 1969.

Kiviat, P. J., Shukiar, H. J., Urman, J. B., & Villanueva, R. The SIMSCRIPT II Programming Language: IBM 360 Implementation, RM-5777-PR, Rand, 1969.

Kiviat, P. J., Villanueva, R., & Markowitz, H. M. The SIMSCRIPT II Programming Language, Prentice-Hall, 1968.

Kleinrock, L. "A Continuum of Time Sharing Scheduling Algorithms," SJCC (1970), 453-458.

Kleinrock, L. "Time-Sharing Systems: Analytical Methods," In Critical Factors in Data Management, Prentice-Hall (1969), 3-32.

Kleinrock, L. "Time-Shared Systems: A Theoretical Treatment," JACM (April 1969), 242-261.

Kleinrock, L., & Coffman, E. G. "Distribution of Attained Service in Time-Shared Systems," Journal of Computer and System Sciences (March 1967), 287-298.

Knight, D. C. "An Algorithm for Scheduling Storage on a Non-Paged Computer," Computer Journal (May 1968), 17-21.

Knuth, D. E. "Additional Comments on a Problem in Concurrent Programming Control," CACM (May 1966), 321-322.

Knuth, D. E. Fundamentals of Programming, Vol. 1, 2, & 3, Addison-Wesley, 1968, 1969, 1972.

Knuth, D. E., & McNeley, J. L. "A Formal Definition of Sol," IEEE Trans on Electronic Computers (August 1964), 409-414.

Knuth, D. E., & McNeley, J. L. "Sol . . . ," IEEE Trans on Electronic Computers (August 1964), 401-408.

Kolence, K. W. "A Software View of Measurement Tools," Datamation (January 1, 1971), 32-38.

Kolence, K. W. "Systems Measurement--Theory and Practice," Proceedings of Share XXXIV (March 1970), 510-521.

Koster, R. Low Level Self-Measurement in Computers, REPT, R-69-57, University of California, October 1969.

Kosy, D. W. "Experience with the Extendable Computer System Simulator," Proceedings of the Fourth Conference on Applications of Simulation, December 1970.

Krasnow, H. S. Highlights of a Dynamic System Description Language, TR-195, IBM, 1966.

Krasnow, H. S., & Merikallio, R. A. "The Past, Present and Future of General Simulation Languages," Management Science (November 1964), 236-267.

Krishnamoorthi, B., & Wood, R. C. "Time-Shared Computer Operations . . . ," JACM (July 1966), 317-338.

Lasser, D. J. "Productivity of Multiprogrammed Computers," CACM (December 1969), 678-684.

Lassettre, E. R., & Scherr, A. L. "Modelling the Performance of the OS/360 Time Sharing Option (TSO)," Statistical Computer Performance Evaluation, Academic Press, 1972, 57-72.

Lehman, M. M., & Rosenfeld, J. L. "Performance of a Simulated Multiprogramming System," FJCC (1968), 1431-1442.

Licklider, J. C. R. "Discussion on Simulation Models," Computer Methods in the Analysis of Large-Scale Social Systems, MIT (1965), 163-165.

Llewellen, R. W. FORDYN--An Industrial Dynamics Simulator, North Carolina State University, 1965.

Lock, K. "Structuring Programs for Multiprogram Time-Sharing On-Line Applications," FJCC (1965), 457-472.

Losapio, N. S., & Bulgren, W. G. "Simulation of Dispatching Algorithms in a Multiprogramming Environment," Proceedings of ACM 1972 National Conference (1972), 903-913.

Lowe, T. C. "Analysis of Boolean Program Models for Time-Shared, Paged, Environments," CACM (April 1969), 199-205.

Lucas, H. C. "Performance Evaluation and Monitoring," Computing Surveys (July 1971), 79-92.

Lucas, H. C. "Synthetic Program Specifications for Performance Evaluation," Research Paper 33, Graduate School of Business, Stanford University.

Luderer, G. W. "On Memory Usage under Multiprogramming," Computer (September-October 1972), 31-34.

Lundell, E. D. "GSA Restricts Simulation Use As Tool for Comparing Systems," Computerworld (June 28, 1972), 1-2.

Lynch, W. C. "Operating Systems Performance," CACM (July 1972), 579-585.

McArdell, R. T. "Design Fundamentals of Transaction-Oriented System," (Part I, II, & III), Control Engineering (June-August 1967), 110-114, 79-82, 81-84.

MacDougall, M. H. "Computer System Simulation: An Introduction," Computing Surveys (September 1970), 191-210.

McIssac, P. V. Time-Sharing Job Descriptions for Simulation, SDC-TM-2713, SDC, November 4, 1965.

McKeag, R. M. Burroughs B5500 MCP, report in series Investigation of Operating System Techniques, 1971.

McKinney, J. M. Optimization Problems Arising in Time-Shared Systems, AD 697 788, September 1969.

McKinney, J. M. "A Survey of Analytical Time-Sharing Models," Computing Surveys (June 1969), 105-116.

McNeley, J. "Simulation Languages," Simulation (August 1967), 95-98.

Maher, R. J. "Principles of Storage Allocation in a Multiprocessor Multiprogrammed System," CACM (October 1961), 421-422.

Manacher, G. K. "Production and Stabilization of Real-Time Task Schedules," JACM (July 1967), 439-465.

Markowitz, H. M. "Simulating with SIMSCRIPT," Management Science (June 1966), 396-409.

Markowitz, H. M., Hausner, B., & Karr, H. SIMSCRIPT, A Simulation Programming Language, Prentice-Hall, 1963.

Marshall, B. S. "Dynamic Calculation of Dispatching Priorities under OS/360 MVT," Datamation (August 1969), 93-97.

Martin, F. F. Computer Modeling and Simulation, Wiley, 1968.

Menon, M. V. "Problem Concerning a Central Storage Device Served by Multiple Terminals," JACM (July 1965), 350-355.

Merikallio, R. A., & Holland, F. C. "Simulation Design of a Multiprocessing System," FJCC (1968), 1399-1410.

Meyerhoff, A. J., Routh, P. F., & Troy, J. P. BOSS, Applications Manual, Burroughs, July 19, 1968.

Mills, H. "Top-Down Programming in Large Systems," Debugging Techniques in Large Systems (1971), 41-56.

Mittman, A. B. SPURT Users' Guide, Northwestern University, 1969.

Moberg, L. V. "An Executive System for On-Line Programming on a Small-Scale System," FJCC (1967), 243-254.

Morenoff, E., & McLean, J. B. "Inter-program Communications, Program String Structures, and Buffer Files," SJCC (1967), 175-183.

Morganstein, S. J., Winograd, J., & Herman, R. SIM/61 . . . . ," ACM SIGOPS Workshop on System Performance Evaluation (April 1971), 142-172.

Morris, J. B. "Demand Paging through Utilization of Working Sets on the MANIAC II," CACM (October 1972), 867-872.

Morse, P. M. Queues, Inventories, and Maintenance, Wiley, 1958.

Muntz, R. R., & Coffman, E. G. "Optimal Preemptive Scheduling on Two-Processor Systems," IEEE Transactions on Computers (November 1969), 1014-1020.

Nakamura, G. "A Feedback Queueing Model for an Interactive Computer System," FJCC (1971), 57-64.

Nemeth, A. G., & Rovner, P. D. "User Program Measurement in a Time-Shared Environment," CACM (October 1971), 661-667.

Nielsen, N. R. The Analysis of General Purpose Computer Time-Sharing Systems. Ph.D. Dissertation, Stanford University, 1967.

Nielsen, N. R. "Analysis of Some Time-Sharing Techniques," CACM (February 1971), 79-90.

Nielsen, N. R. B6500 Time-Sharing Design Study Data, ETM # 508, October 1970.

Nielsen, N. R. B6500 Time-Sharing System Simulator, User's Reference Manual, ETM # 329, December 1969.

Nielsen, N. R. "Computer Simulation of Computer System Performance," Proceedings of 22nd National ACM Conference (1967), 581-590.

Nielsen, N. R. ECSS: Extendable Computer System Simulator, RAND CORPORATION, RM-6132-PR, January 1970.

Nielsen, N. R. "The Simulation of Time-Sharing Systems," CACM (July 1967), 397-412.

Noe, J. D. A Note on the Representation of Operating Systems, TR # 70-09-03, University of Washington, 1970.

Noe, J. D. "A PETRI Model of the CDC 6400," ACM SIGOPS Workshop on System Performance Evaluation (April 1971), 362-378.

Noe, J. D. Systems Performance, Measurement, Prediction, Optimization--Part 2, presented at Fall 1971 meeting of Cooperating Users of Burroughs Equipment (CUBE).

Noe, J. D., & Nutt, G. J. "Validation of a Trace-Driven CDC 6400 Simulation," SJCC (1972), 749-758.

O'Connor, T. J. Analysis of a Computer Time-Sharing System--A Simulation Study. Ph.D. Dissertation, Stanford University, 1965.

O'Neill, R. W. "Experience Using a Time-Shared Multiprogramming System with Dynamic Address Relocation Hardware," SJCC (1967), 611-621.

Ophler, A. "Measurement of Software Characteristics," Datamation (May 1964), 42-45.

Oppenheimer, G., & Weizer, N. "Resource Management for a Medium Scale Time-Sharing Operating System," CACM (May 1968), 313-322.

Ossanna, J. F., Mikus, T. E., & Danten, S. D. "Communications and I/O Switching in a Multiplex Computing System," FJCC (1965), 231-241.

Parente, R. J. A Language for Dynamic System Description, TR-180, IBM, 1965.

Parente, R. J., & Krasnow, H. S. "A Language for Modeling and Simulating Dynamic Systems," CACM (September 1967), 559-567.

Parslow, R. D. "AS: An ALGOL Simulation Language," Simulation Programming Languages (1968), 86-100.

Parpudi, M., & Winograd, J. "Interactive Task Behavior in a Time-Sharing Environment," Proceedings of ACM 1972 National Conference (1972), 680-692.

Pass, E. M. "SNOOPY--A Software Monitor Device for B5700 Programs," paper presented at Fall 1971 meeting of Cooperating Users of Burroughs Equipment.

Patrick, R. L. "Measuring Performance," Datamation (July 1964), 24-26.

Penny, J. P. "An Analysis, Both Theoretical and by Simulation, of a Time-Shared Computer System," The Computer Journal (May 1966), 53-59.

Petri, C. A. Transitional Networks, 1962.

Petroni, L. "On a Simulation Language Completely Defined Onto the Programming Language PL/I," Simulation Programming Languages (1968), 305-318.

Pinkerton, T. B. "Performance Monitoring in a Time-Sharing System," CACM (November 1969), 608-610.

Pinkerton, T. B. "Program Behavior and Control in Virtual Storage Computer Systems," CONCOMP Project Report 4, April 1968.

Pomerantz, A. G. "Predict Your System's Fortune: Use Simulation Crystal Ball," Computer Decisions (June 1970), 16-19.

Poole, P. C. "Some Aspects of the EGDON 3 Operating System for the KDF 9," IFIP (1968), 43-47.

Prittsker, A. A. B. JASP: A Simulation Language for a Time Shared System, RM-6279-PR, Rand Corporation, June 1970.

Pugh, A. L. DYNAMO User's Manual, MIT, 1961.

Pugh, A. L. DYNAMO II User's Manual, MIT, 1970.

Purser, W. F. C. "System Timing for On-Line Computer Control," Instrumentation Technology (January & February 1969), 41-46, 51-56.

Raichelson, E., & Collins, G. "A Method for Comparing the Internal Operating Speeds of Computers," CACM (May 1964), 309-310.

Randell, B., & Kuehner, C. J. "Dynamic Storage Allocation Schemes," CACM (May 1968), 297-306.

Randell, B. "A Note on Storage Fragmentation and Program Segmentation," CACM (July 1969), 365-369.

Rasch, P. J. "A Queueing Theory Study of Round-Robin Scheduling of Time-Sharing Computer Systems," JACM (January 1970), 131-145.

Rawlins, W. H., & Weller, M. F. "Peripheral Optimization in Operating Systems," Honeywell Computer Journal (Summer 1968), 10-19.

Rehmann, S. T., & Gangwere, S. G., Jr. "A Simulation Study of Resource Management in a Time-Sharing System," FJCC (1968), 1411-1430.

Reingold, E. M. "Establishing Lower Bounds on Algorithms: A Survey," SJCC (1972), 471-482.

Richards, P. Parallel Programming, Report TD-B60-37, Technical Operation, Incorporated, 1960.

Rodriguez-Rosell, J. "Experimental Data on How Program Behavior Affects the Choice of Scheduler Parameters," SIGOPS (June 1972), 156-163.

Rodriguez-Rosell, J., & Dupuy, J. "Instrumentation for the Evaluation of a Time-Sharing, Page Demand System," SJCC (1972), 759-766.

Rosen, S. "Electronic Computers: A Historical Survey," Computing Surveys (March 1969), 7-36.

Rosen, S. Programming Systems and Languages, McGraw-Hill, 1967.

Rosenberg, A. M. "The Brave New World of Time-Sharing Operating Systems," Datamation (August 1969), 42-47.

Rosin, R. F. "Determining a Computer Center Environment," CACM (July 1965), 463-468.

Rosin, R. F. "Supervisory and Monitor Systems," Computer Surveys (March 1969), 37-54.

Roth, P. F. "The BOSS Simulator--An Introduction," Proceedings of the Fourth Conference on Applications of Simulation, December 1970.

Ryder, K. D. "An Heuristic Approach to Task Dispatching," IBM Systems Journal (September 1970), 189-198.

Saaty, T. L. Elements of Queuing Theory, McGraw-Hill, 1961.

Saltzer, J. H. Traffic Control in a Multiplexed Computer System. Ph.D. Dissertation, MAC-TR-30, MIT, July 1966.

Saltzer, J. H., & Giotell, J. W. "The Instrumentation of MULTICS," CACM (August 1970), 495-500, and Proceedings of Second ACM Symposium on Operating Systems Principles (October 1969), 167-174.

Sayers, A. P. Operating Systems Survey, Auerbach, 1971.

Sayre, D. "Is Automatic Folding of Programs Efficient Enough to Displace Manual?" CACM (December 1969), 656-660.

Scherr, A. L. "Analysis of Main Storage Fragmentation," Proceedings of IBM Symposium on Storage Hierarchy Systems, TR-00-1556 (December 1966), 159-174.

Scherr, A. L. An Analysis of Time-Shared Computer Systems. Ph.D. Dissertation, MAC-TR-18, MIT, 1965.

Scherr, A. L. "Time Sharing Measurement," Datamation (April 1966), 22-26.

Schrage, L. E., & Miller, L. W. "The Queue M/G/1 with the Shortest Remaining Processing Time Discipline," Operations Research (July-August 1966), 670-684.

Schrage, L. E. Queuing Models for a Time-Shared Facility. Ph.D. Dissertation, Cornell University, February 1966.

Schulman, F. D. "Hardware Measurement Device for IBM System/360 Time-Sharing Evaluation," Proceedings of 1967 National ACM Meeting (1967), 103-109.

Schwartz, E. S. "Computer Evaluation and Selection," Journal of Data Management (June 1968), 264-279.

Schwetman, H. D. A Study of Resource Utilization and Performance Evaluation of Large-Scale Computer Systems. Ph.D. Dissertation, University of Texas at Austin, 1970.

Schwetman, H. D., & Brown, J. C. "An Experimental Study of Computer System Performance," Proceedings of ACM 1972 National Conference (1972), 693-703.

Seaman, P. H., & Soucy, R. C. "Simulating Operating Systems," IBM Systems Journal (October 1969), 264-279.

Selwyn, L. L. "Computer Resource Accounting in a Time-Sharing Environment," SJCC (1970), 119-130.

Sewald, M. D., Rauch, M. E., Rodiek, L., & Wertz, L. A. "A Pragmatic Approach to Systems Measurement," Computer Decisions (July 1971), 38-40.

Sharpe, W. F. The Economics of Computers, Columbia University Press, 1969.

Shedler, G. S., & Yang, S. C. "Simulation of a Model of Paging System Performance," IBM Systems Journal (April 1971), 113-128.

Shemer, J. E. "Some Mathematical Considerations of Time-Sharing Scheduling Algorithms," Journal of the ACM (April 1967), 262-272.

Shemer, J. E., & Heying, D. W. "Performance Modeling and Empirical Measurements in a System Designed for Batch and Time-Sharing Users," FJCC (1969), 17-28.

Shemer, J. E., & Robertson, J. B. "Instrumentation in Time-Shared Systems," Computer (July-August 1972), 39-48.

Sherman, S., Browne, J., & Baskett, F. "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogrammed System," ACM Workshop on Systems Performance Evaluation (April 1971), 173-199.

Sherman, S., Baskett, F., & Browne, J. "Trace-Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming System," CACM (December 1972), 1063-1069.

Shoshani, A., & Coffman, E. G. Sequencing Tasks in Multiprocess, Multiple Resource Systems to Avoid Deadlock, TR-78, Princeton University, 1969.

Shoshani, A., & Coffman, E. G. Detection, Prevention, and Recovery from Deadlocks in Multiprocess, Multiple Resource Systems, TR-80, Princeton University, 1969.

Smith, J. L. "An Analysis of Time-Sharing Computer Systems Using Markov Models," SJCC (1966), 87-95.

Smith, J. L. "Multiprogramming under a Page-on-Demand Strategy," CACM (October 1967), 636-646.

Smith, J. M. "A Review and Comparison of Certain Methods of Computer Performance Evaluation," The Computer Bulletin (May 1968), 13-18.

Software Products Corporation. A Technical Description of the CASE System, Software Products Corporation, Virginia, 1968.

Statland, N. "Methods of Evaluating Computer Systems Performance," Computers and Automation (February 1964), 18-23.

Standeven, J., Bowden, K. F., & Edwards, D. B. G. "An Operating System for a Small Computer Providing Time Shared Data Collection, Computing, and Control Functions," IFIP (1968), 55-59.

Stang, H., & Southgate, P. "Performance Evaluation of Third-Generation Computing Systems," Datamation (November 1969), 181-190.

Stanley, W. I. "Measurement of System Operational Statistics," IBM Systems Journal (1969), 299-308.

Stanley, W. I., & Hertel, H. F. "Statistics Gathering and Simulation for the Apollo Real-Time Operating Systems," IBM Systems Journal (1968), 85-102.

Stevens, D. F. "On Overcoming High-Priority Paralysis in Multiprogramming Systems: A Case Study," CACM (August 1968), 539-541.

Stevens, D. F. "System Evaluation on the Control Data 6600," IFIP (August 1968), C34-38.

Stimler, S. "Some Criteria for Time-Sharing System Performance," CACM (January 1969), 47-53.

Stimler, S., & Brons, K. A. "A Methodology for Calculating and Optimizing Real-Time System Performance," CACM (August 1968), 509-516.

Stone, D. L., & Turner, R. "Disk Thruput Estimation," Proceedings of ACM 1972 National Conference (1972), 704-711.

Strachey, C. "System Analysis and Programming," Information (1966), 56-75.

Strauss, J. C. "A Simple Thruput and Response Model of EXEC 8 under Swapping Saturation," FJCC (1971), 39-50.

Teichrow, D., & Lubin, J. F. "Computer Simulation . . . . . ," CACM (October 1966), 723-741.

Teory, T. J., & Pinkerton, T. B. "Comparative Analysis of Disk Scheduling Policies," SIGOPS (June 1972), 114-121.

Tetzlaff, W. H., Cooperman, J. A., & Lynch, W. A. "SPG: An Effective Use of Performance and Usage Data," Computer (September-October 1972), 20-23.

Thompson, W. C. "The Application of Simulation in Computer System Design and Optimization," Second Conference on the Applications of Simulation (1968), 286-290.

Tocher, K. D. "Review of Simulation Languages," Operational Research Quarterly (June 1965), 189-217.

Tocher, K. D., & Hopkins, D. A. "New Developments in Simulation," Proceedings of Third International Conference on Operations Research (1963), 832-848.

Tocher, K. D., & Owen, D. G. "The Automatic Programming of Simulations," Proceedings of Second International Conference on Operations Research (1960), 50-68.

Tonik, A. B. "Development of Executive Routines, Both Hardware and Software," FJCC (1967), 395-408.

Torgerson, P. E., et al. "Introducing Queuing Concepts: A Simulation Approach," JIE (May 1967), 328-333.

Totaro, J. B. "Real-Time Processing Power: A Standardized Evaluation," Computers and Automation (April 1967), 16-19.

Totschek, R. A. A User-Oriented Priority Scheme for a Time-Sharing System, SDC-SP-211, SDC, June 14, 1965.

Totschek, R. A. An Empirical Investigation into the Behavior of the SDC Time-Sharing System, AD622003 or SP2191, SDC, 1965.

Trachtenberg, M. "Problems of Evaluating Operating Systems," Proceedings of 1968 DPMA Conference (1968), 21-27.

Turski, W. M. "SODA--A Dual Activity Operating System," Computer Journal (August 1968), 148-156.

Ullman, J. D. The Performance of a Memory Allocation Algorithm, Technical Report Number 100, Princeton University, 1971.

UNIVAC. "EXEC 8 Software Instrumentation," UNIVAC Systems Programming Fall USE Conference, pp. 7-53-7-73.

UNIVAC. UNS-Network Simulator Programmer's Reference, UP7548, UNIVAC, 1967.

UNIVAC. UNIVAC 1100 Series Executive Programmer Reference Manual, 1971.

UNIVAC. 1108 ALGOL Programmer's Reference, UP7544 Rev 1, UNIVAC, 1970.

UNIVAC. 1108 SIMULA Programmer's Reference, UP7556 Rev 1, UNIVAC, 1971.

Van Felder, M. K., & England, A. W. "A Primer on Priority Interrupt Systems," Control Engineering (March 1969), 101-105.

Van Horn, E. C. Computer Design for Asychronously Reproducible Multiprocessing. Ph.D. Dissertation, MIT, 1966.

Varian, L., & Coffman, E. G. "An Empirical Study of the Behavior of Programs in a Paging Environment," Proceedings ACM Symposium on Operating System Principles, October 1967.

Varney, R. C. "Process Selection in an Hierarchial Operating System," SIGOPS (June 1972), 106-108.

Wald, B. Throughput and Cost Effectiveness of Monoprogrammed, Multiprogrammed, and Multiprocessing Digital Computers, AD654384, U. S. Navy, 1967.

Walter, C. J., et al. "Impact of Fourth Generation Software on Hardware Design," IEEE Computer Group News (July 1968), 1-10.

Walter, E. S., & Wallace, V. L. "Further Analysis of a Computing Center Environment," CACM (May 1967), 266-272.

Warner, C. D. "Monitoring--A Key to Cost Efficiency," Datamation (January 1, 1971), 40-49.

Watson, R. A. Measurement and Analysis of Computer System Performance, RAND CORPORATION, R-573-NASA/PR, December 1971.

Weaner, D. G. "QUIKSIM--A Block Structured Simulation Language Written in SIMSCRIPT," Proceedings of Third Conference on Application of Simulation (1969), 1-11.

Wegner, P. Programming Languages, Information Structures, and Machine Organization, McGraw-Hill (1969), 56-58.

Weik, M. H. Standard Dictionary of Computers and Information Processing, Hayden, 1969.

Weil, J. W. "A Heuristic for Page Turning in a Multiprogrammed Computer," CACM (September 1962), 480-481.

White, P. "Relative Effects of Central Processor and Input/Output Speeds upon Throughput on the Large Computer," CACM (December 1964), 711-714.

Wichmann, B. A. "A Modular Operating System," IFIP (1968), 48-54.

Wickland, G. A. SIMQUEUE--A Queuing Simulation Model, WP Series 69-13, University of Iowa, July 1969.

Wiener, J., & De Marco, T. "Tuning for Performance," Modern Data (January 1970), 54.

Wilkes, M. V. "A Model for Core Space Allocation in a Time-Sharing System," SJCC (1969), 265-271.

Wilkes, M. V. "Automatic Load Adjustment in Time-Sharing Systems," ACM Workshop on Systems Performance Evaluation (April 1971), 308-320.

Wilkes, M. V. "The Design of Multiple-Access Computer Systems," (Part 1 and Part 2), Computer Journal (May 1967), 1-9.

Wilkes, M. V. Time-Sharing Computer Systems, Elsevier, 1968.

Williams, J. G. "Experiments in Page Activity Determination," SJCC (1972), 739-748.

Williams, J. W. J. "ESP--The Elliott Simulator Package," Computer Journal (January 1964), 328-331.

Williams, O., et al. "A Methodology for Computer Selection Studies," Computers and Automation (May 1963), 18-23.

Winograd, J., Morganstein, S. J., & Herman, R. "Simulation Studies of a Virtual Memory, Time Shared, Demand Paging Operating System," SIGOPS (June 1972), 149-155.

Wirth, N. "Machine Coding, Multiprocessing, and Machine Organization," CACM (September 1969), 489-498.

Wood, D. C., & Forman, E. H. "Throughput Measurement Using A Synthetic Job Stream," FJCC (1971), 51-56.

Wood, T. C. "A Generalized Supervisor for a Time-Shared Operating System," FJCC (1967), 209-214.

Wulf, W. A. "Performance Monitors for Multiprogramming Systems," Proceedings of Second Symposium on Operating Systems (October 1969), 175-181.

Wyatt, J. B., & Dewan, P. B. OSSL--Operating Systems Simulation Language--A User's Guide, University of Houston, November 1971.

Yeh, A. C. "An Application of Statistical Methodology in the Study of Computer System Performance," Statistical Computer Performance Evaluation, Academic Press, 1972, 287-328.

Youchak, M. S., Rudie, D. D., & Johnson, E. J. "The Data Processing System Simulator (DPSS)," FJCC (1964), 251-276.

Yourdon, E. "An Approach to Measuring a Time-Sharing System," Datamation (April 1969), 124-126.

Ziegler, J. R. Time-Sharing Data Processing Systems, Prentice-Hall, 1967.

Ziegler, J. R. "Time-sharing & Software," Data Processing Magazine (September 1966), 38-40.

Zunich, L. H. "Study of OS-360-MVT SYSTEM-S CPU Timing," SHARE Computer Measurement and Evaluation Newsletter, February 7, 1970.

VITA

Edgar M. (Bud) Pass was born September 13, 1945, in Atlanta, Georgia. In 1963 he was graduated from East Atlanta High School as valedictorian. In 1967 he was graduated from Georgia Institute of Technology with the degree of Bachelor of Science in Applied Mathematics. In 1968 he was graduated from the same institution with the degree of Master of Science in Information Science.

During the years of 1965 to 1972, he worked for the Rich Electronic Computer Center at Georgia Institute of Technology as a systems programmer primarily responsible for the maintenance of the Burroughs B5700 software. During those years and to the present, he also functioned as an independent consultant in the field of computer applications, developing such systems as credit card payment automation, motor carrier rate and mileage computation, the automatic conversion of ink-print text into Braille, a large online inventory system, and a large state tax-collection information system. Also, from 1968 to 1971 he taught graduate courses in the area of analysis and design of computer operating systems for the School of Information and Computer Science.