MULTIPROGRAMMING IN A PAGE ON DEMAND COMPUTER SYSTEM: PERFORMANCE MODELS AND EVALUATION

by

DAVID SCHWARTZ

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

in

ELECTRICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

JUNE 1973

Signature of Author _____
Department of Electrical Engineering, May 11, 1973.

Certified by _____
Thesis Supervisor

Accepted by _____
Chairman, Departmental Committee on Graduate Students

MASS. INST. TECH.
JUN 4 1973
LIBRARIES

MULTIPROGRAMMING IN A PAGE ON DEMAND COMPUTER SYSTEM: PERFORMANCE MODELS AND EVALUATION

by

DAVID SCHWARTZ

Submitted to the Department of Electrical Engineering
on June 1, 1973, in partial fullment of the
requirements for the Degree of Master of Science

ABSTRACT

The System Process Model (SPM) is used to represent behavior of the multiprogramming module of a computer system operating under a demand paging strategy. It models the system as a network of states and associated queues. The level of multiprogramming is varied by changing the average number of pages that users are permitted in main memory. Devices are explicitly characterized by parameters representative of their behavior, and sample programs run under CP-67 are parameterized in terms of I/O and paging activity for use in representing jobs in the system.

With the intention of using this model for performance projection, a potential problem characteristic of such virtual memory systems is discussed. The thrasing problem is defined as the less than optimal use of system resources resulting from excessive competition for primary memory. Novel definitions for pinpointing this degredation are proposed.

SPM is then used as a vehicle to analyze the operations of a CP-67 like system. Performance is predicted under full load for various levels of multiprogramming and optimal performance as well as thrashing degredation are quantified. Scheduling is shown to be a key technique for improving system performance as well as improvements in device technology. An interesting characteristic of the system, the law of diminishing returns to scale, is uncovered as a result of the investigations.

Analysis is performed with both analytical and simulation models. This permits comparative analysis of the methods based on some interesting and revealing criteria. The techniques, which are of both theoretical and practical value in the area of performance evaluation, are rated on their approach to modeling SPM.

ACKNOWLEDGEMENTS

I would like to express my gratitude to all of the sincere people at both MIT and Harvard that have been so helpful in the preparation of this manuscript.

At MIT, I extend first thanks to Professor S.E. Madnick, my thesis supervisor. He introduced me to the area of computer system modeling, thus providing the initial stimulus for this research. He also created the atmosphere of friendship and understanding so important to the overall effort. I would like to thank Professor J. Donovan for his insight and continuing support of the research. In addition, I would like to extend thanks to Professor D. Larson, my graduate advisor, for his assistance in the Markov-state analysis that was carried out.

At Harvard, Professor U.O Gagliardi and J.P. Buzen provided an excellent environment for the study of computer system performance. I would like to thank them for their constructive criticism as well as the interesting literature in the field that they introduced to me. I would especially like to thank Jeff Buzen for his assistance in analyzing the queuing network model.

Words meaningful enough to thank my wife, Sharon, for her lasting patience and assistance are difficult to find. I thank her especially for help in the final preparation of the paper. She has been my ultimate queuing state, and I dedicate this thesis to her.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SYNOPSIS AND ORGANIZATIONAL REMARKS

The abstract is provided as an overview of the contents of this thesis. The following outline gives a more indepth look into the chapters. This will facilitate selection of the most appropiate for the reader.

CHAPTER 1: motivation for computer modeling is provided as well as a brief introduction to current techniques being used in the field. The thesis objectives are stated.

CHAPTER 2: a review of the literature is given and the specific work of researchers that is relevent to this thesis is discussed in more detail, both in analytical and simulation modeling.

CHAPTER 3: an example multiprogrammed, demand-paged, virtual memory system is presented. A conceptual view of the computer called the System Process Model is proposed to represent the system in a manner conducive to performance evaluation.

CHAPTER 4: analytical models are designed and solved in an attempt to provide useful information about the system under study. This includes analysis of continuous time Markov-state models and a queuing network model which is another continuous time Markov model.

CHAPTER 5: a simulation model is developed and implemented to study the same system. Characteristics and problems of simulation are outlined, and then tackled in the GPSS model designed. Extensions of simulation analysis are discussed.

CHAPTER 6: the performance projection results are obtained and discussed at length. Thrashing is pinpointed, the optimal level of multiprogramming is found, and scheduling of critical devices is shown to improve system performance. The device technology of the paging channel is upgraded with encouraging performance results. The system as a whole is observed to obey the law of diminishing returns.

CHAPTER 7: the modeling techniques are compared on the basis of the level of detail describable, the number of parameters needed to describe the model, the ease of implementation, the ease and exactness of the solution, and the overall cost of the approach. In addition, areas of future research are suggested.

APPENDIX A: a PL/1 program,used to solve the continuous time Markov-state models, is displayed with an extended program listing and sample output.

APPENDIX B: a PL/1 program , used to solve the queuing network model, is displayed with an extended program listing and sample output.

APPENDIX C: the extended program listing of the GPSS model is presented.

Chapters 1 and 2 are review for anyone familiar with computer modeling. They are suggested for the reader interested in an introduction to this field.

Chapter 3 lays the ground work for the rest of the thesis and should be read in its entirety. The proposed computer system module is presented in detail so that the conceptual model, SPM, can be completely specified.

Chapters 4 and 5 are basically independent, presenting the analytical and simulation models used in analysis of SPM. However, the parameter-

ization of the GPSS simulation in Chapter 5 is based on the underlying path selection formulation of the queuing network model in Chapter 4. So, in fact, sections (4.4.1 and 4.4.2) which introduce the system representation for the network of queues are recommended prerequisites to Chapter 5.

Chapter 6 presents the results obtained for the performance projection performed and is essential to the understanding of the flavor of the models.

Chapter 7 is of interest to the reader that is about to select a modeling technique for system analysis. It provides insight into the approaches taken in this modeling effort.

CHAPTER 1

INTRODUCTION

1.1. The Need For System Modeling

The better part of a quarter century of research and development in the computer industry has brought us from the days of the original stored program computer to complex, time-shared, multiprogrammed, multiprocessor, demand-paged, virtual storage, emulated computers. Yet, when a group of interested computer bodies were probed by Dr. Phister of Xerox Data Systems on the industry as a phenomenon (P1), no one was willing to deny that the industry is in it's infancy. This means that the multipurpose systems we have now are in the very process of evolving, being dynamically reconfigured to meet new needs.

A variety of observations have been made about systems of this nature with regard to performance evaluation and modeling. Calingaert (C1), Lucus (L1), and Estrin (E1) have extensively surveyed and appaised the

current and relevent issues of this discipline. One general point that is made in these papers is that both design and maintainence of these complex systems is usually inadequate unless modeling and analysis complements the efforts. More fundamentally, it has been found that the cost of developing and maintaining efficient systems of this kind becomes prohibitive if care is not taken to understand the capabilities and limits of the utility. It is modeling then, that serves as the vehicle for system design and analysis by predicting behavior in an environment likened to the real system.

In some cases these models have provided detailed descriptions of systems by there self-documenting nature. Generally, they have provided valuable insight into system bottlenecks, demonstrated complex interdependancies, led to more intelligent measurements, and been the catalyst for improved system performance.

1.2. Approaches To System Analysis: Analytical And Simulation Models

The general approach taken in computer modeling and performance analysis involves isolating key parameters of system behavior; then, through their interaction in a system model, varying the parameters to determine subtle interrelationships among software and hardware resources. This approach can be part of a feasibility study as well as an evaluation of an existing system.

It is the actual modeling tools to which we now direct our attention. The modeling methodology relevent to work in this thesis can be classified into two categories: analytical modeling and simulation modeling. These techniques will be reviewed here as an introduction to the approaches that

will be used to study multiprogramming in a demand-paged computer system. Both of these modeling techniques will be discussed in greater detail when the literature review is presented in Chapter 2, when actual models are applied to problem solving in Chapters 4 and 5, and when the techniques are compared in Chapter 7.

1.2.1. Analytical Models

Analytical models are mathematical conceptions of a system as opposed to the digital computer stochastic representation discussed in the next section. These models utilize a variety of underlying approaches to formulate solutions for the proposed system and can be classified on this basis.

Continuous time Markov models form one group. They are characterized by their explicit representation of all possible states in the system, with the values of the states defined in a manner that permits insight into system operation. Representation of the system is based on the Markov property. Balance equations are usually written for the steady state and have been solved by a variety of techniques; sometimes approximation methods must be used and in some cases closed form solutions are available. These models have been applied to the study of a wide variety of waiting line situations, i.e. queing networks.

In some situations the system to be modeled is not really Markov in nature. In order to make the Markovian assumption, it is necessary to study what is known as the embedded chain. This approach permits analysis of the above nature to be performed and yields results about the original system. Examples of the use of this technique are

found in the study of the M/G/1 queing system and the study of priority queing networks.

Math programming models have also been applied in operating systems optimization problems as well as more conventional models based on the fundamental principles of probability theory. The former models represent the computer system in terms of a cost minimization or maximization problem and usually require resourceful manipulation of the system variables to ensure feasibility. The latter models break system operation down into simple operations that can be probabilistically represented and have provided valuable predictive information.

In general, more expertise is needed in this area than in simulation modeling in terms of model preparation. Also, it is often very difficult to determine the appropriate analytical model for a given system, since real world constraints often lead to situations where equations that are written are infeasible. Yet, inspite of these drawbacks, this approach has the advantage of lower cost.

1.2.2. Simulation Models

The alternative approach to analysis mentioned was simulation; in this approach, programs are written to execute the interations of a proposed system, characterized as an associated set of decision rules and related probability distributions. The classification of simulations into categories depends on both the programming language utilized as well as the system under study. The uniqueness of the model depends on the techniques used to integrate all of the inputs for analysis.

A categorization of simulation language has recently been made

by Kay (K2). His contention is that the principal of virtual time traced event execution complemented by a variety of statisical barometers is basic to each approach in implementation. His "photographs" of the family tree of languages specify the higher-level language family with simulations writtten in a "host" scientific language, the GPSS-family, with simulations based on transaction flow through blocks which activate routines maintaining timing by updating discrete event chains, and the SIMSCRIPT-family, with simulations written in a prietary syntax, running time shortened by a system state approach that advances time to the next value when a change of state is imminent, and necessary compiler facilities to handle the most flexible simulation requirements.

This language classification has been quite general. In fact, when a specific type of system is selected for study, another category must be added to our simulation repertoire. This is the class of simulations that describe their system with a parameterized set of inputs. Motivation for this approach to modeling will be provided by brief mention of a problem encountered in the field of communication systems. There it was found that the hybrid nature of the system (neither purely continuous nor purely discrete) posed problems in terms of modeling with the available simulation languages. General purpose, higher-level languages had been employed; however, simulations were prohibitively difficult for the communications engineer, not well versed in programming techniques. Consequently, a special purpose language was designed to permit the engineer more programming ease, while at the same time providing functional blocks to handle the special purpose

features required in the field (G7).

This example serves to demonstrate the feasibility of taylored simulation languages. Now, although the discrete, stochastic nature of computer behavior permits modeling by the whole gamut of available languages, the nature of the system is specialized enough to warrant a language that permits easier, more accurate descriptions. The benefits gained are a top down approach to modeling by computer people that are not as familiar with simulation techniques, flexible and valid building blocks, and less chance of errors by the analyist. Languages such as this were at one time felt to be prohibitive because of rapid changes in computer technology and the broad range of objectives of the simulation analyist (H3). However, current research and development in this area has proven to be quite profitable (D5,W3).

Whichever language is selected for use in analysis, the technique of simulation is the most well known and widely used tool. There are various reasons for this; for instance, simulation permits modeling various levels of detail depending on the goal of the analyist. And, it also lends itself to understanding by management, usually less versed in the actual implementation methods; this results in wider acceptance of results. Generally it has been said that simulation raises expectations about predictions more than any other modeling technique.

1.3 Thesis Objective

The topic of computer modeling has received wide spread attention in the literature during the past decade, with a variety of analysis techniques having been exposed that provide a means to an end. These

ends have taken on many shapes and forms such as selection evaluation, performance projection,and performance monitoring (L1). The means to achieve these ends have been as diversified as the analysists that have performed the studies.

It is the intention of this thesis to intergrate and further the work in this field to date. The goal of the modeling effort is to study the operation of sample programs in a CP-67 like multiprogramming environment and to project system performance under a variety of load conditions. Then, for the selected computer system configuration, both analytical and simulation techniques will be applied to gain insight into the problem of performance degredation in such a multiprogrammed, demand-paged, virtual memory system. Novel definitions for determiningifthe system is operating in what is known as the thrashing region are defined. This region of degraded performance is identifiable by the less than optimum utilization of system resources. Investigation of system performance under full load conditions is carried out, and both scheduling of device activities and improvements in device technology are shown to be key techniques for improving system performance.

The main intention in selecting this subject material is to lend insight into both the interdependencies that can arise in complex computer systems and the problems they can cause.

The approach taken in analyzing the problem can be termed a multipronged modeling effort: a representative group of modeling tools selected from a survey of the current literature is resourcefully applied to study performance of a multiprogramming system. This approach was

selected in order to permit a comparative analysis of current modeling techniques. It yields a variety of observations about their applicability and value.

CHAPTER 2

WHO'S WHO IN COMPUTER MODELING

2.1. Overview of the Target System

The introduction to modeling methodology in section (1.2) serves as a forerunner to the literature review which follows. It is here where the mathematicians and computer analysists that have effected the evolution within the modeling field will be discussed. In addition, the work of those authors that have done work relevent to this thesis will be presented in greater depth.

To facilitate this review let's begin with an overview of a contemporary computer. One simplified view of such a system is shown in figure 2-1. Here the standard complement of hardware components is displayed. The central processor is the traditional instruction interpreter responsible for job execution. The memory hierarchy can be viewed as consisting of main (content addressable) memory and

secondary storage (Drum, Disk), the I/O hierarchy as I/O buffers in main memory, channel control units, device control units, and the actual devices themselves. The cooridinator of all of these facili-

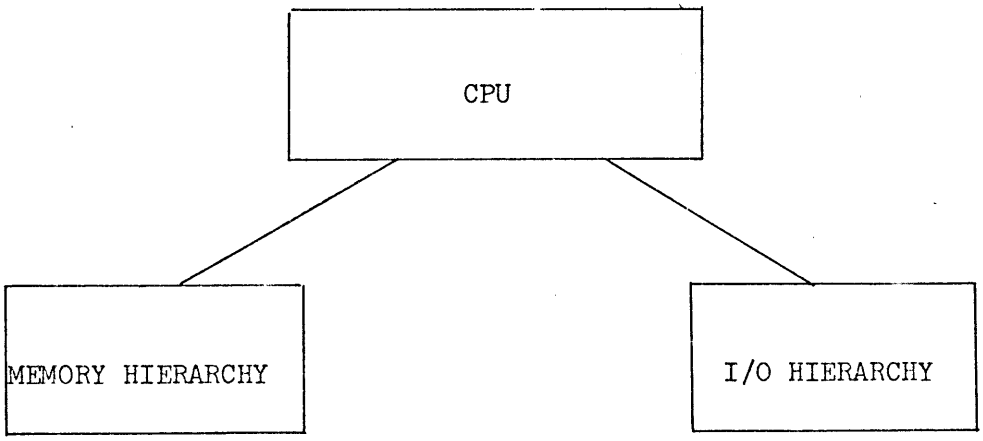


Fig. 2-1 Resource Partitioning in Operating System Overview

ties is the operating system; this key software resource is an open-ended set of programs that manages the system via its scheduling algorithms and system-wide data bases. This executive or monitor can be described as coorinating the following minimum set of functions (M2):

Memory Management
: This function is in charge of keeping a running account of memory, allocating blocks of the resource on the basis of user need via a proprietary algorithm, and reclaiming it when appropiate. This function is also responsible for supporting exotic virtual storage mechanisms when implemented.

Processor Management
: This function keeps track of the status of the central processor(s), assigning the resource to users ready to execute. It determines the time quantum, if multiprogramming, and reclaims the resource when control is relinquished or the

time slice is exceeded.

Device Management

This function keeps track of the resource paths in use, and determines efficient handling of devices for users requesting service. It specifies the device driver routines.

Information Management

This funtion keeps track of the information resource known as the file system. It provides protection and primitive functions for flexible use of the facility. It handles the logical aspect of information transfers.

This scenario can be most accurately described by the state transition diagram displayed in figure 2-2. Not only does this figure demonstrate job flow through the system, but it also provides insight into the interaction of the above mentioned operating system functions.

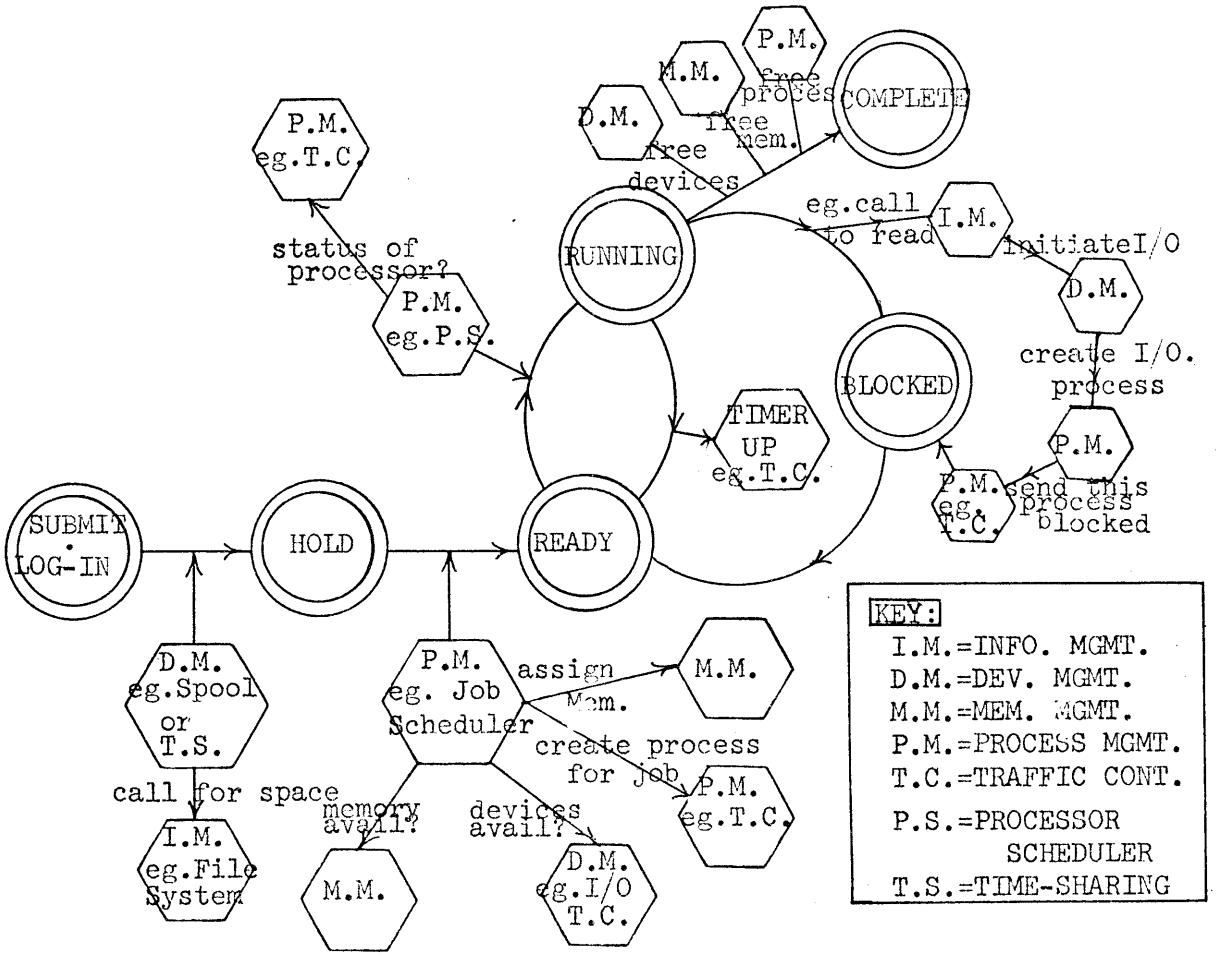


Fig. 2-2 Overview of Operating System Operations

Figure 2-3 shows the same basic diagram, now blocked off into modeling sectors. Thus, the state diagraph will be the means for presenting current modeling efforts, classifying the work of the analysists on the basis of the computer structures they have studied.

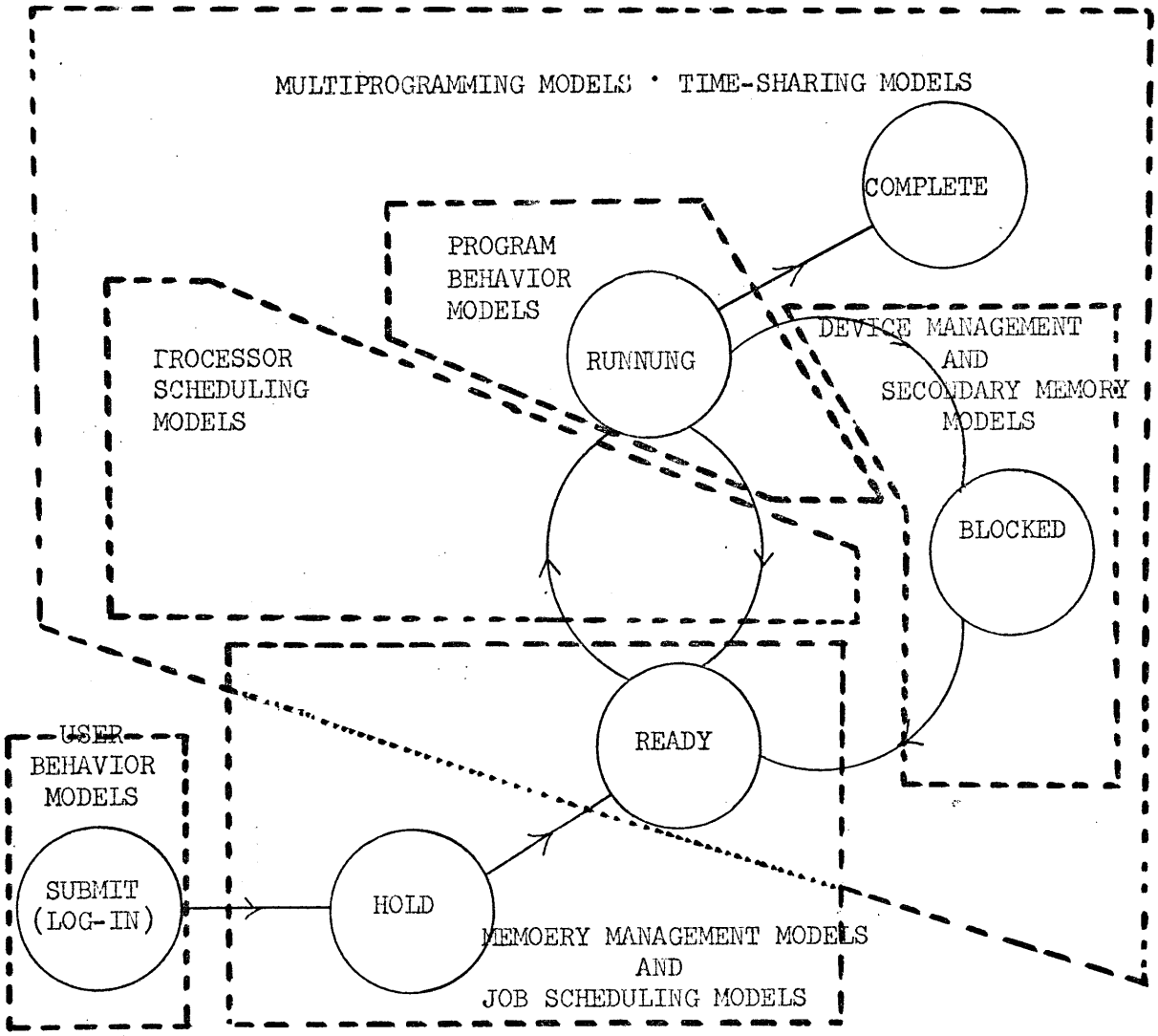


Fig. 2-3 Operating System Overview Partitioned By Models Used for Analysis

2.2. Analytical Modeling

In terms of analytical models, computer behavior has been viewed with a queuing theoretic eye. The development of queuing techniques to deal with waiting lines dates back to 1902 when Erlang used congestion theory to analyze the telephine traffic problems in Copenhagen (K3). Since that time the development of queuing theory has often been characterized as stemming from "solutions looking for a problem" (N1), as mathematicians such as Kolmogorov (F1), Feller (F1), Kendall (K3,K4), Takacs (T1), and Gordon and Newell (G5) approached the theoretical issues of queues and their underlying stochastic processes. Other mathematicians such as Jackson (J1,J2) were no less theoretical, but were motivated by practical problems. Special techniques have often been employed in analyzing queues because of the complexity of their stochastic processes. Consequently, when possible, analyists have made use of the theory of Markov chains developed by A. A. Markov (1856-1922) to simplify their work (F1,K3). In addition, arguments from renewal theory have also been found to be of value in faciliting mathmatical analysis (F1,D7).

Applications of the theory have been wide spread, with new horizons broadening with the increasing complexity of the world around us. In this context, the advent of practical digital computers [von Neuman 1947 (S8)] created a very rich area which lay ready to be opened to the application of queuing models. Interestingly enough, both areas of study remained fairly independent in the literature, maturing at their own pace for many years. Then, in the early 1950's Kendal (K4)

published his work on the use of imbedded Markov chains to analyze the stochastic problems of the theory of queues. This work did not immediately stimulate computer analyists to apply queing models in their analysis; however, it can be pointed to as a key break through which permitted significant work to be done in a variety of computer areas (C2, S3, T2). Similarly, the later work of Jackson, who began studying open networks of queues with poisson arrivals and exponential servers, and Gorden snd Newell, who expanded his work to closed networks, made advances which would be used quite extensively by computer men such as Buzen (B3), Moore (M7), and Rice (R2).

Summarizing, the analytical modeling approaches that have met with success in the analysis of computer systems utilize theory in the areas of continuous time Markov models, embedded Markov chain analysis, busy period arguments of renewal theory, math programming, and fundamental probability theory.

Specific work done in these areas is now characterized according to module of concentration, based on figure 2-3:

Time-Sharing:

Scherr,A.L.(1965)-S2
Coffman,E.G.(1966)-C6
Smith,J.L.(1966)-S9
Kleinrock,L.(1967)-K7,K8
Chang,W.(1968)-C4
Moore,C.G.(1971)-M7
Sekino,A.(1972)-S7

Processor Scheduling(waiting line management problems):

Graver,D.P.(1962)-G1
Chang,W.(1965)-C2
Schrag,L.E.(1967)-S3

Multiprogramming:

Smith,J.L.(1965)-S10

Wallace and Mason(1969)-W1
Buzen,J.P.(1971)-B3
Belady,L.A.(1966)-B2

Memory Management:
Belady,L.A.(1966)-B2
Denning,P.J.(1968)-D3
Corbato,F.J.(1969)-C11
Weizer,N.(1969)-W2
Denning,P.J.(1970)-D4

Device Management:
Chang,W.(1965)-C3
Denning,P.J.(1967)-D1
Coffman,E.G.(1969)-C7
Abate,J.(1969)-A1
Frank,W. (1969)-F2

Program Behavior:
Denning,P.J.(1968)-D2
Hatfield,D.J.(1972)-H1
Madnick,S.E.(1972)-M3,M4
Sekino,A.(1972)-S7
Saltzer,J.(1972)-S1

User Behavior:
Scherr,A.L.(1965)-S2
(usually approximated with a poisson process)

In retrospect, we see a trend that has resulted in a better understanding of computer design and behavior, as analytical modeling has be become a specialized field in its own right.

2.3 Simulation Modeling

The other school of thought that has also expended a great deal of energy to model and analyze computer systems is that of simulation. Simulation as a modeling tool has as intricate a history as analytical modeling, but is richer in development and broader in terms of accomplishments boasting applications in a wide spectrum of activities.

Since our primary concern is to look at computer modeling of computers, a detailed history of the post World War II development of the general techniques of this science will be sacrificed. A detailed description is presented by Martin (M6). Let it suffice to say that it was the advances made in computer technology that not only made simulation analysis feasible, but also made it necessary for the introspective aspect of this modeling. For, it is the complexity of our computers that has resulted in the intense study of computers with simulation.

The interesting nature of this subject, not to mention the predictive value of well implemented models of this sort has resulted in the development of a simulation folklore(G12).The approach to presenting some of the more current work in this field will be different than the one employed to discuss analytical modeling. First, the role of simulation in computer analysis will be reviewed by discussing one paper from a group of selected articles on this subject. Then some representative efforts will be categorized by language, and again, one paper from each group will be discussed in order to give a flavor for the type of work done.

To open the discussion, Seaman(S5), Huesman and Goldberg(H3), Humphrey(H4), and Maguire(M5) address digital simulation in general terms and attempt to determine the role of simulation modeling in the computer field. Issues such as the advantages of the technique, the need for higher-level special purpose languages, and the desire to design "self-monitoring and self-adjusting systems" are brought to light. To expand in depth the most recent of these presentations, Maguire, in a very read-

able presentation, developed the rational and methodology of simulation. He explained the reasons for the wide spread use of this tool and cautioned against the ill equipped, over-zealous analysist as one of the only deterrents to even more wide spread use in modeling. He expanded upon each of his reasons with an eye toward the decade upon us; the discussion included recognition of simulation's advantages, advancements in the techniques for producing models, increased availability of generalized models, increased application of experimental design techniques, and increased availability of relevent model data.

Continuing with the review, the intention is to now present the categroization of modeling by language; a sampling of the activities in each group is included with each category.

The classification of simulation languages has already been given in section (1.2). The work done in modeling has drawn on all of these, with selection based on factors such as programmer language background, languages supported within reach, complexity of the system (detail to be modeled), and the time constraints on producing a finished product.

In the area of algebraic languages, the work of Scherr(S2), Nielsen (N2), Macdougal(M1), and Lum et al (L2) are worthy of mention. The approach taken in these efforts was to develop special purpose routines in the host language that facilitate a top down modeling approach. The work of Nielsen in the development of a general purpose time-sharing systems simulator is representative of this group.

In order to develop an efficient and widely usable model, he programmed his system in a common subset of Fortran. In order to make the model

flexible and general, pains taking effort was taken to survey characteristics of existing systems and predicted future developments. He then chose a level of detail that adequately reflected system performance. He managed to keep coding down as well as execution time short. The model was run for a proposed single processor IBM 360/67 TSS.

The next category includes studies implemented in GPSS, the General Purpose Systems Simulator (G4,G6,S4). Examples of the application of this language are the work of Seaman(S5) and Holland and Merikallio(H2).

Seaman used this language to study the design issues of teleprocessing systems. His effort concentrated on predicting quantatively, the delays resulting from requests queuing for time-shared facilities. This resulted in the ability to determine the amount of storage needed to hold the queues for the processes. Demonstrating the modularity of GPSS models viz a vi its block structure, he presented the logical design of a simulation for a 360 model 30 information processing utility.

Models programmed in Simscript(D6) form the next category. It is noted that this language has permitted the most detailed analysis yet available in a special purpose language. Examples are work by Katz(K1) and work done at TRW systems(T4).

Katz's simscript program involved the study of a direct-coupled, multiprocessor operating system. The model represented the optimal level of detail that permitted accurate descriptions, an important characteristic of a good simulation. The language was demonstrated as capable of handling complex, adaptive scheduling algorithms and able to show the changes in system throughput as a function of priority schema in the

various queuing disciplines.

The last area is special purpose parameterized languages. Examples of such languages are CSS, which was developed as a result of lack of success in implementing complex computer models in GPSS(S6). Lomus II was developed as a Simscript based, generalized simulation system as well (H3). OSSL is a more recent example of such a language. This Fortran based computer simulation language deserves further mention (D5,W3).

Programs model computer systems of varying complexity, permitting macroscopic, yet detailed, description of state of the art features such as exotic memory management algorithms and complex job mixes. Configuration of the system is facilitated by hardware block descriptions and programming is eased by writing taylored procedures. This language is currently being used to teach a course in computer engineering at Harvard (W3) and provides an excellent top down approach to system analysis.

This list of languages and models was not meant to be exhaustive. It is hoped that qualification of work done in this area and the presentation of sophisticated studies in perspective has generated an implicit belief in the value of the simulation methodology.

2.4 Work Relevent to this Thesis

As has been demonstrated above, the range of research done in the area of modeling has been extensive. In this thesis, a selection will be made of those approaches that can be used to evaluate the performance of virtual memory computers viewed as the System Process Model.

The analytical work done in this area draws on established techniques

of queuing theory, making use of the Markov assumption to solve the models proposed. The specific models that are used derive from continuous time Markov models. The authors credited with work in this area are mentioned below in terms of the work done here, and they are discussed at length in the following sections.

Smith (S10) and Sekino (S7) are well known for their use of continuous time Markov processes in research on demand paging systems. In this thesis these processes are also proposed as feasible models. The technique used to develop equations for their solution is attributed to the work of Feller (F1). An approximate method for their solution in the steady state can be found in Drake (D7).

Buzen (B3), Rice (R2), and Moore (M7) have recently completed doctoral work on the use of queuing network models for studying computer behavior. The work to be done here uses the analysis performed by Buzen on closed queuing networks to solve the proposed model. The result of this work is a closed form solution that yields a wealth of performance information.

In the area of simulation, work of a more general nature has been found to be of value to the author. Schreiber (S3) has presented a thorough discussion of IBM's General Purpose System Simulator. The techniques he presents for system modeling are useful in analyzing SPM.

2.4.1 The Work of J.Smith

Smith studied multiprogramming under a page on demand strategy. He modeled a subsystem of a time-shared, multiprogrammed computer

that produced information on average CPU utilization. The system included a representation of a drum-I/O channel. Both of these system components were modeled with continuous time Markov processes. Within the multiprogramming module, three programs, maximum, were permitted. In addition, execution intervals were exponentially distributed and a priority system was implemented which determined selection between two job classes. A utilization factor, ρ, was defined for the drum-I/O channel in terms of program and hardware characteristics. It was found to adequately reflect system performance and worked well as a gauge for multiprogramming. The model was solved with RQA-1 (W1) and statistics were used from SDC. No definite conclusions were drawn, but insight was provided into multiprogramming operation.

2.4.2. The Work of A. Sekino

This doctoral presentation is likely to be viewed as one of the most complete analytical analyses of an existing multipurpose computer system. With an eye toward validating his model on the MULTICS system, Sekino developed a general model for a time-shared, multiprogrammed, multiprocessor, demand-paged, virtual storage computer.

He viewed the overall system as a hierarchical set of models, each of which contributed a key parameter to his final analysis. Specifically, Markov models were used to study program behavior, memory effects on paging behavior, and multiprogramming in a multiprocessor environment. The result was an analysis of the systems "percentile throughput" (a barometer found to be as valuable as system throughput) as well as a determination of the user's response time distribution.

The analysis delved into many phases of computer operation, taking into consideration issues such as data-base lockout for shared writable information, memory cycle interference resulting from multi-processor operation, program paging behavior as a function of memory size, and overhead times resulting from page faulting in a demand paging environment.

2.4.3 The Work of J.P.Buzen, D.Rice, and C.Moore

Buzen developed a conceptual model of multiprogramming called the Central Server Model which, in fact, is the same as the System Process Model expoused here. The mathematical model selected for analysis was a continuous time Markov model which studied the proposed system as a network of queues. He derived basic properties of program behavior in his system, such as the distribution of the number of processing requests per program and the total processing time per program. These were found useful when applying the model to specific performance evaluation problems. Independent of Gordon and Newell, he formulated and solved in closed form, balance equations for the equilibrium probabilities. A most notable result was his discovery of an efficient computational algorithm that made evaluation of the steady state probabilities and a variety of interesting marginal distributions very easy. Analysis of specific computer system problems proceeded with the hypothesis of the meaning of system bottleneck. Defined in terms of a sensitivity analysis, it was suggested that if a small increase in the performance of the component in question produced a large increase in overall system

performance, that component was responsible for degrading operations. Equipped with this methodology, he analyzed problems such as buffer size determination, peripheral processor utilization, and page traffic balancing. Each analysis was facilitated by a mathematical formultion of optimization equations based on a measure defined for system processing capacity.

The work of Rice also delved into an analysis of a queuing network model of multiprogramming. His basic model differed slightly from that studied by Buzen. The nature of job exit and entry permitted a variable level of multiprogramming with a maximum bound of N. Steady state solutions were provided by applying an approximation technique and were found to yield accurate results. A variety of system performance measures were derived for the overall model. In addition, an approach to modeling an I/O hierarchy as a subsystem composed of queue dependent servers was found to be useful for representing a disk system.

Moore's work is yet another example of research into queuing network models of computer behavior. His analysis was dedicated to time-sharing system structure and resulted in a more intricate net of queues than that studied by Buzen and Rice. The reason for this was his effort to represent more complex task routing that occurs in such systems. Observing as did Rice, that queue dependent servers could be used to model a disk subsystem, he included this feature in his model to make it more realistic. He was also able to represent other more realistic service distributions by matching exponentials. Validation of his model for MTS was effected with real system measurements.

2.4.4 The Work of T.Schreiber

In the area of simulation analysis, the work of Schreiber will be discussed. His work is in the implementation issues and modeling capabilities of GPSS. The presentation is available as a very comprehensive and readable manuscript which he uses to teach a course in simulation at the Michigan business school. The case study approach was effectively applied to build up modeling techniques, and quite complex situations were studied. This work deserves credit in light of all of the material available to learn GPSS, for included are very realistic examples and an affectionate concern for all the details and subtle shortcomings of the language.

CHAPTER 3

MULTIPROGRAMMING IN A PAGE ON DEMAND COMPUTER SYSTEM AND A CONCEPTUAL MODEL FOR PERFORMANCE EVALUATION

3.1 Page On Demand Virtual Memory Systems

Virtual memory systems have been developed to satisfy basic needs of multiprogramming and time-sharing systems. Their value in these applications expounded on by Denning (D4), includes such advantages as the ability to load programs into spaces of arbitrary size, the ability to run such partially loaded programs, and the ability to vary the space allotted these loaded programs. Of course, the benefits to the user include the illusion of a nearly unlimited address space.

Although advances in memory storage techniques may one day obviate the need for the complex operating systems needed to manage such virtual memory computers, it is not clear that this will occur even in the distant future. Thus, we can say with assurity that virtual systems are evolv-

ing as the new state of the art, and feel confident that analyzing such systems is a valuable way to spend energy. It is hoped that such analysis will lead to insight into operating characteristics and will result in improved system performance.

An appropriate point of view to take from the outset which will facilitate analysis, is to think of the operating system as a manager of resources. This points up the important question of resource allocation in virtual memory systems. What key resources must be available for the system to operate efficiently and how are they utilized?

To begin to answer these questions, the behavior of programs in the system must be characterized; for, it is to their service that the system is dedicated. To better understand how a system manages resources for this important task, let us look at the underlying aspects of a real virtual memory system that operates under Control Program-67. This discussion is based on a recent paper by Bard (B1).

The system is multiprogrammed and operates under a demand paging strategy. User jobs execute until one of the following events transpires:

(1) A page fault occurs: A program requests execution of a location in its address space that is not currently in executable memory.

(2) An input/output request is issued to a peripheral device.

(3) The executing program exceeds its allotted time quantum.

(4) A wait PSW is loaded.

Upon occurrence of one of these interruptions, program execution is terminated, and execution is initiated in the next users address space. A program encountering either (1) or (2) is said to be blocked and cannot

receive CPU service until the occurrence of an associated external event denoting service completion. Reschdeuling for CPU service the occurs in the same manner as for programs interrupted by timer runnout or by the loading of a wait PSW. Thus, it is evident from this discussion, that important resources are "real" memory for program execution and peripheral devices that can be used for servicing paging and I/O requests.

Further explanation of the operation of this system in the event of a page fault is of value. It is this aspect of the systems operation that reveals the memory management algorithm, an important function essential to the understanding of program characterization: the paging behavior of a program in a demand paging environment.

The system operates under a single bit LRU removal algorithm. This function , called upon whenever a program needs a new page, maintains a pointer to a cyclical list of all main memory pages. Each page has an associated usage (reference) bit which is turned on when the page is referenced by a user program. The algorithm moves the pointer around this list of pages and inspects the reference bit of the page selected for examination. If the bit is off it removes it, otherwise it turns the bit off and moves on to check the next page. The reference bits of a users pages are also turned off when the user is removed from in-queue status. Thus, a global removal algorithm is implemented by the cycling pointer.

One can invision, then, a user program that has a time dynamic allotment of main memory based on his demand for pages and the removal of pages by the system algorithm. Bard calls this allotment of memory the

users resident set. Changes in its size can occur when a user acquires a new page due to a fault or when pages are removed while the program is in the interrupted state. Thus, we can see that based on executable memory available and system activity, system paging can vary greatly. This may give rise to an interesting phenomenon that will be investigated in this thesis: Thrashing. For, when the system is paging rapidly due, say, to a high level of multiprogramming, it is quite possible to have system performance deteriorate (the users average resident set size decreases as the level of N increases resulting in an increase in the number of page faults suffered per user).

Given this system description, there are a variety of open questions to which we might address ourselves. For instance, it would be of value to specify a quantative definition of when thrashing is in progress from the point of view of performance projection. Also, determination of the optimal level of multiprogramming for a given job mix as well as expected system throughput for the load is of interest. Then, from a resource management point of view, it would be of interest to identify utilization of system components in the hopes of finding any potential system bottlenecks.

The logical approach to answering these questions is to develop an operable, conceptual model of the system that will yield statistical data and performance information. Before doing this, a more detailed discussion of Thrashing follows in order to further motivate the need for a quantative model for analysis.

3.2 The Thrashing Problem: Novel Quantitative Definitions

The analysis of multiprogramming in a page on demand environment has received much attention (S10,D3,W1). A potent problem that designers of systems of this nature face is called thrashing: excessive competition for primary memory resulting in degredation of system performance. Denning (D3) describes the less than optimum use of resources in this situation as follows, "Thrashing inevitably turns a shortage of memory space into a surplus of processor time."

The basic reasons why thrashing occurs have been isolated. They are (1) the large times required to access pages stored in auxilliary memory and (2) the availability of limited real memory coupled with address space acquisitions that are uninsulated because of a global removal algorithm.

It is intended to show how these two factors do, in fact, result in this severe performance degredation in the demand paged system studied here. However, it is necessary to define the term "degredation" more explicitly before we can hope to quantify an actual thrashing point for the system. Two views for pinpointing this operation are now presented (M2).

3.2.1 The Multiprogramming Cost of Job Execution

The first view revolves around associating a cost factor with a job executing in a multiprogramming environment. The cost function is determined for a job under given load conditions by charging him for the amount of non-overlapped processor time that is spent during his execution, i.e. this accounting system charges the user for any time that the

CPU cannot be shared. The factor has three components that can be described as follows:

(1) Each program requires CPU processing time to complete. There is no overlapping of CPU time while the job performs its computations. Call this time T_{CPU}.

(2) Each program issues a number of I/O commands. When such a request is issued by a program, charge him for the idle CPU time that results because there are no other programs available to utilize the CPU. Call this time $T_{I/O}$.

(3) Each program suffers page faults depending on the level of N. Charge him for idle CPU time that results when he issues a page request and there are no other programs available to utilize the CPU. Call this T_{PAGE}.

Then, for a program whose paging and I/O activity are viewed as occuring essentially in separate systems, the entire cost for running in a multiprogramming environment can be determined:

$$T_{TOTAL} = T_{CPU} + T_{I/O} + T_{PAGE}$$

Based on this cost approach, it is clear that such a program should experience a decrease in T_{TOTAL} as the level of multiprogramming increases, since the possibility of overlapping the execution time of one of the jobs with the waiting time of another increases. However, can one get too much of a good thing? Yes! And this is where the threshold definition for Thrashing arises. In a monoprogrammed environment, T_{TOTAL} for the sample program can be determined. Granted, this program doesn't make efficient use of system resources. However, we can definitely say that we would not want an increased level of multiprogramming (N) to result in as poor use of the resources, i.e. we would not want increased N to result in associating a cost factor as large as the monoprogrammed case with a user job under that system load. It is clear that at this point the

system is loosing. Define this as the Thrashing point. This approach to determing the bounds on multiprogramming is of pedgagocial value because each factor causing the thrashing phenomenon can be observed individually.

3.2.2 Average CPU Utilization

Alternatively, view two uses the steady state fraction of CPU busy time as a direct measure of Thrashing point. Here, one can determine this CPU utilization for the monoprogramming environment. Then, with an increase in the degree of multiprogramming, it is possible to determine the point when the utilization of this key resource degrades to the value found for the monoprogrammed case. This point is where Thrashing will be said to begin. This point of view is intuitively obvious for our system model, since we would like to optimize the use of our central resource for processing jobs. However, in general, one might question the motive for using this CPU utilization criteria as a system barometer. For, in a real system, the CPU might be 95% utilized and yet performing no useful work (80 % of the time is being spent as overhead in the supervisor). In such a situation, one would definitely be advised to campaign for an analysis that produced results regarding system throughput or turn around time. It will be shown for the conceptual model that is proposed in the next section, that system throughput is directly proportional to average CPU utilization, and so use of CPU utilization as a system barometer is justified.

3.3 The System Process Model: A Conceptual Model of Computer Behavior

The System Process model (SPM) is a means to describe the operations of a multiprogrammed computer utility, characterizing the activity of the processes in terms of program behavior parameters and the interaction of the collection of processes in terms of occupancy of state positions in a directed graph of the system.

I observe with Denning (D2), Kimbleton (K5,K6), and Sekino (S7) that the time trace of these processes finds them circulating among the following three states:

i. active: receiving CPU service

ii. waiting: queued for or receiving service from a peripheral device, i.e. blocked waiting for the completion of an associated external event.

iii. ready: queued for CPU service

In the work of the above authors, the system under study was a virtual memory system operating under a demand paging algorith, and consequently, processes were characterized solely by their paging behavior. In this model, a process will be characterized not only by its paging behavior but by its I/O behavior as well. The inclusion of I/O activity makes the analysis consistent with the description of the CP-67 multiprogramming module. The wait PSW event will not be included in this analysis, and it will be assumed that a job in execution issues an I/O request or suffers a page fault before a timer interrupt occurs. This assumption makes it possible to use the analytical modeling techniques that will follow. It is not overly restrictive for operation of CP-67 like systems which actually demonstrate this behavior.

3.3.1 Sample Program Behavior

Consideration of the activity of program behavior in this slightly simplified multiprogramming environment is now appropriate. View a memory partition with a user's pages as an executable address space that presents the CPU with processing requests, I/O requests, and page requests for new executable memory space. Let a partition be loaded from the peripheral paging device and assume that the initial processing activity required to partially page in a user for execution is part of the general paging activity of that program. Observe, that the degree of multiprogramming is determined by the number of pages allotted to each users memory partition. Varying the level of N implies that that a user's allotment of "real" core has been altered by the system.

It can be seen that a request for service from one of the shared resources (CPU, drum,disk) might not be satisfied if the server is busy. Consequently, the executing process stands to suffer queuing delays while in the system. His life cycle then consists of alternating sequences of active,blocked, and ready intervals, with possible queuing delays. This is the basic nature of the System Process Model.

Now it will serve us well to explicitly describe the program behavior so that the appropriate parameters can be evaluated for model analysis.

Sample Program Characteristics:
(this program is an assembler run under CP-67 (M2))

(1) Page size is 4K-Bytes

(2) 160K-Bytes of virtual memory is required by the entire job

(3) The program runs for 2,000,000 instructions, i.e. it requires 2 sec. of CPU time to complete execution.

(4) The total amount of I/O time experienced by the job when run in a stand-alone environment is 3 sec. (This job is a class C job as specified in OS-9-10: Job parameter statements). In other words, if a request is serviced in 50 msec, this job issues a total of 60 I/O's.

(5) The paging behavior of the program will be a function of the amount of "real" core memory available for it's executable pages. A static parachor curve for the sample program is shown in figure 3-1.

The static parachor curve predicts the number of instructions that are executed between page faults when a job runs in a fixed partition in core paging against itself. Figure 3-2 then shows the functional relationship between the number of page faults and the degree of multiprogramming; the level of N was determined as described above for 256K-Bytes of total core available to the users.

From our previous discussion, it might appear that what one really wants is a measure of the dynamic behavior of programs in the multiprogramming environment, i.e. a dynamic parachor that predicts page faults as a function of average resident set size. Certainly, this data would more accurately represent the dynamic character of the resident set (B1). However, for the following two reasons, the static parachor will be quite appropriate for our use:

(1) Our ultimate goal is to predict a bounds on the level of multiprogramming and to determine system performance for that load. Since the static parachor tends to over-page for a given memory size, it is certain that the slack in the actual values is on the side that we can afford to live with.

(2) The queuing models that will be utilized in analyzing SPM view the operating system as managing multiprogramming among fixed memory partitions.

Fig. 3-1

SAMPLE PROGRAM PAGING BEHAVIOR AS A FUNCTION OF REAL MEMORY AVAILABLE, 4K-BYTE PAGES, LRU

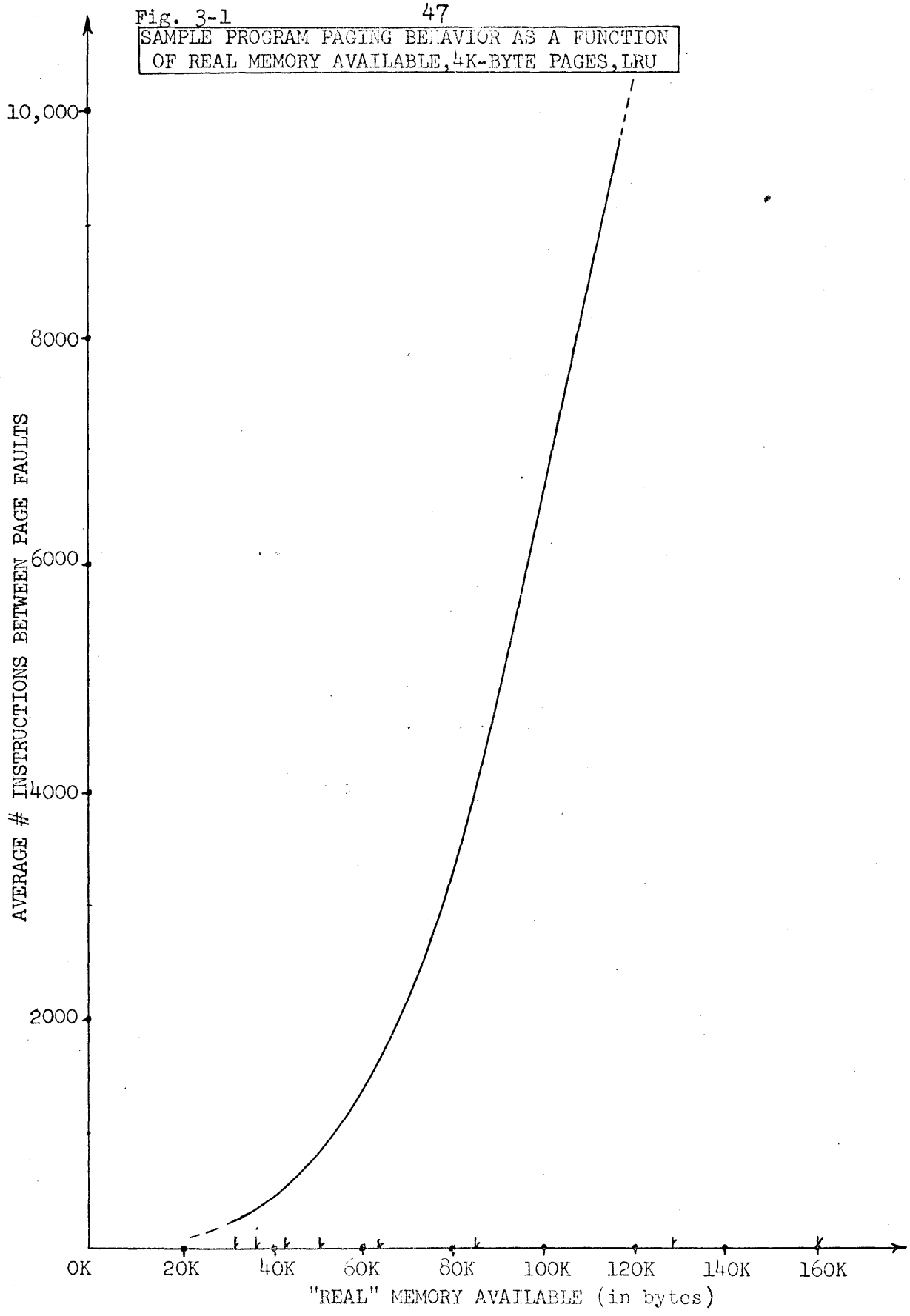
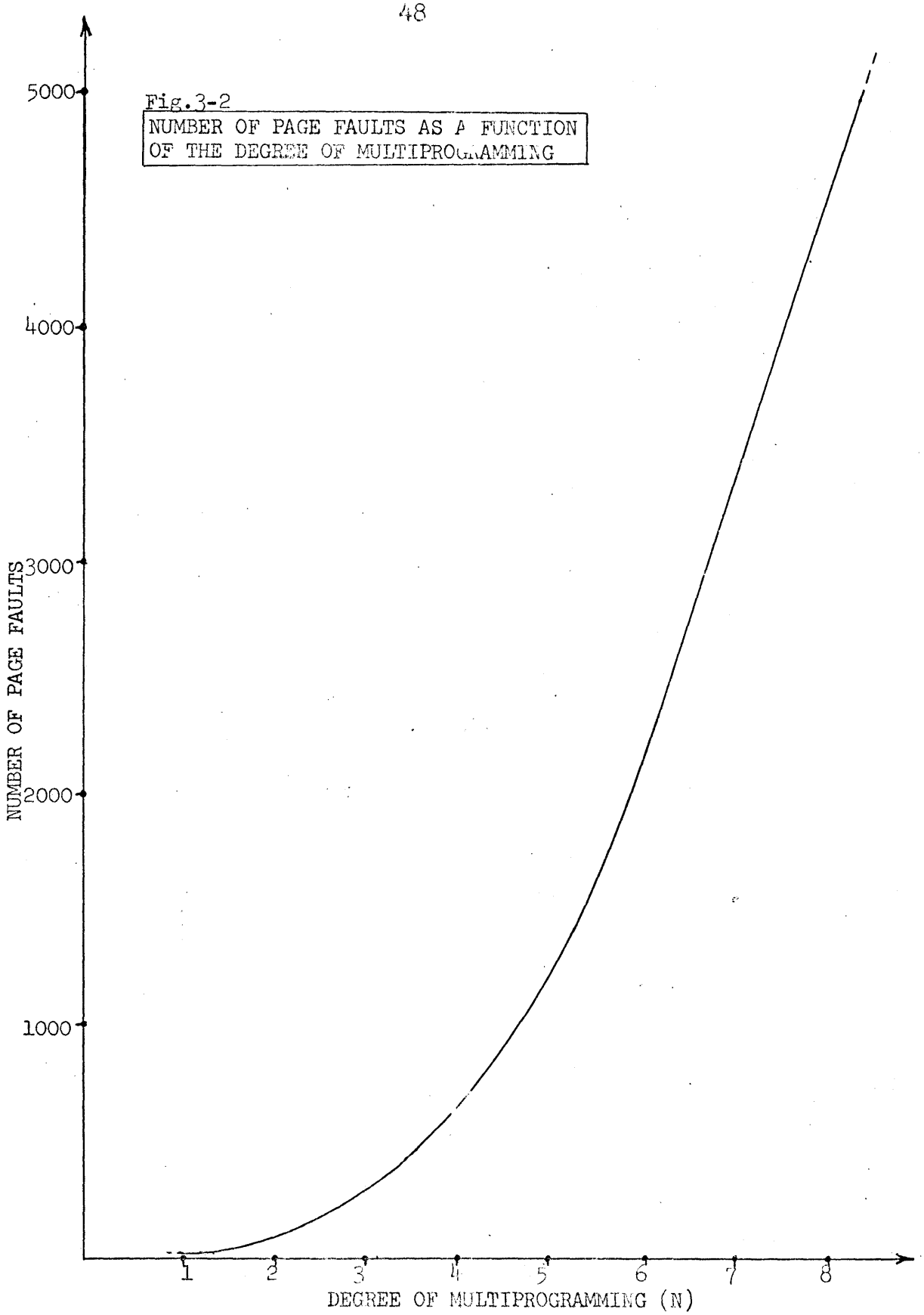


Fig.3-2
NUMBER OF PAGE FAULTS AS A FUNCTION OF THE DEGREE OF MULTIPROGRAMMING



3.3.2 The Input/Output Subsystems

It yet remains to characterize the peripheral devices that service the page faults and I/O requests described above. The intention is to specify the device characteristics in a manner that will provide realistic values for device services; the use of available models for these components is taken advantage of where appropriate.

3.3.2.1 The Paging Drum

The page handler will consist of a high speed magnetic drum. Models for such systems have been analyzed in detail in the literature (D1, C7, F2, A1); For the purposes of this discussion, Denning's FCFS model with no request scheduling will be used to initially represent the device service mechanism. An exponential distribution will be assumed with mean:

$$E(\text{service 1 request}) = \frac{N-1}{2N} \cdot T + \frac{T}{N}$$

where N is the number of sectors
T is the time for one revolution
note: the first term can be thought of as the latency time and the latter, the constant transfer time.

The service is for single page requests in the demand paged system. It is assumed that requests for transfer are uniformly distributed among the N sectors on the drum. It is also assumed that a useful model need only treat the transfer of pages from the drum to the high speed memory (page output from core to drum creates no delay in the input of pages from drum to core) (S10).

Selection of drum rotation rates is based on current and advanced technology. The following values will be used in the analysis:

Drum Rotation (msec)	Expected Service Time (msec)
(Slow) 40.00	25.00
(Medium) 16.70	10.94
(Fast) 8.00	5.00

3.3.2.2 The I/O Disk

The I/O handler is a disk device. Using characteristic values for a 2314 unit (GI-21 Rev.2), service will be assumed to be exponentially distributed with mean 50 msec. This figure includes seek, rotational delay, and transfer time for a standard I/O from the sample program.

3.3.2.3 Scheduling the I/O Subsystems

The FCFS expected service times above involve long queuing delays and do not reflect, very effectively, the performance of the devices under queued loads. At best, they provide a means for determining a lower bound for system performance by operating in the slowest manner expected for the particular device. In order to demonstrate the effect of scheduling as well as to indicate a manner for determining the upper bound of performance for the given system configuration, the following two views will also be taken of the subsystem hierarchies:

(1) Service by the device is dependent on the load of requests in the device queue, i.e. the mean service time for a request depends on the length of the queue. This view of the server is equivalent to including scheduling into the channel. Included in this view is the effect of saturating the server. For example, as requests queue for service in the drum, the sector queues might receive individual requests which could be effectively serviced at the same time. However, as more requests filled the queues, it would be likely that a request is already in a subque. A typical distribution for

this sort of activity is as follows:

let Ms = the mean time to service a request

R = the service rate of that device

where R defines the negative exponential distribution:

prob. service interval $\leq t = 1 - e^{-R \cdot t}$

then $Ms = 1/ k \cdot R \qquad k = 1 \text{ to } j \qquad 1 \leq j \leq N$

$Ms = 1/ j \cdot R \qquad k \geq j$

This discipline is referred to in later analyses as scheduling.

(2) Each request receives instantaneous service with the expected service rates selected in sections (3.3.2.1 and 3.3.2.2). There is no queuing for a sector or a track, and each memory partition thinks it has its own service device. This view results in better than realistic service and yields an upper bound for multiprogramming performance. This service distribution is as follows

$Ms = 1/ k \cdot R \qquad k = 1 \text{ to } N$

and is referred to in later analyses as No Queuing or Optimal(Opt).

3.3.3 Basic Modeling Assumptions

Based on the above discussion the following collective observations will be made:

i. The N jobs existing in the system are uniformly characterized as described and are statistically identical

ii. The service disciplines are each characterized by an exponential distribution.

iii. The processor and peripheral devices are switched instantaneously among the jobs in the ready queues.

iiii. The system is assumed to be fully loaded at all times.

Assumption 'i' corresponds to the test situation in which measurements were made for a job in it's own fixed partition, but it is re-

strictive in terms of real job mixes. With regard to 'ii', the resourceful use of exponential holding times has succeeded in modeling general physical systems many of which don't strictly conform to this behavior (S2,M7). This assumption also makes possible the analytical analysis performed. Assumption 'iii' can be relaxed quite naturally with simulation as can all of the above restrictions. It is also necessary to permit the analytical modeling that follows. Lastly, assumption 'iiii' permits consideration of job completions. It implies that there is always a new job waiting to begin processing, keeping the level of multiprogramming constant.

3.3.4 A Derivation of System Throughput for SPM

In section (3.2.2) it was ascerted that the average CPU utilization was an adequate monitor of system performance. In order to demonstrate that this is really true, the following derivation finds CPU utilization to be directly proportional to system throughput. A similar derivation was performed by Sekino (S7) when he analyzed the "percentile throughput" of his system model.

Define the following parameters for a single processor system:

hs: the headway made by the CPU in processing user jobs between service requests.

mtbs: the mean time between service requests

utiliz: the utilization factor of the CPU

also, define the following overhead terms one might associate with such a demand paging system:

tp: paging overhead time including supervisory functions required to handle a page fault

ti: I/O overhead time including supervisory functions required to handle an I/O request

tm: misc. overhead time corresponding to all the other supervisory functions.

In a system with overhead, we have:

$$mtbs = \overline{hs} + \overline{tm} + f(\overline{ti} + \overline{tp})$$

However, in the system under study, observe that:

$$mtbs = \overline{hs}$$

because of the assumptions of no overhead.

Now, the "percentile throughput(Θ)" of such a system, i.e. "the percentage of the systems computational capacity spent as the user's useful work"), can be expressed as:

$$\Theta = \text{utiliz} * \frac{\overline{hs}}{mtbs} = \text{utiliz}$$

Then for an arbitrary length of time, T(sec), the time that the single processor facility spends doing useful work for the user is $\Theta \cdot T$. Each user's execution time (2000msec) is represented as Te_i i=1 to K, for K user jobs completed in this time. Then, under full load, the system throughput, Th-Pt, can be calculated as follows:

$$\text{Th-Pt} = \lim_{T\to\infty} \frac{K}{T}$$

$$= \lim_{T\to\infty} \left\{ \frac{K}{T} \cdot \frac{\Theta \cdot T}{Te_1 + Te_2 + \ldots Te_K} \right\}$$

$$= \lim_{T\to\infty} \left\{ \frac{\Theta}{\left\{ \sum_{j=1}^{K} Te_j \right\} / K} \right\} = \frac{\Theta}{\overline{Te}}$$

$$\boxed{\text{Th-Pt} = \text{utiliz} / \overline{Te} \quad \text{Q.E.D.}}$$

CHAPTER 4

ANALYTICAL MODELS FOR PERFORMANCE ANALYSIS

4.1 Mathematical Model Selection

The issue to which we will now address ourselves is that of finding a resourceful approach to solving the System Process Model. Solving this multiprogramming model involves mapping a class of input functions into a relevent set of output functions. The input functions, as specified in section (3.3), were found to be process request rates for the system resources and expected device service times. The important output functions were presented in section (3.2) when the discussion of thrashing resulted in the selection of two factors that could be used to quantify system degredation: the cost of multiprogramming in non-overlapped processor time and/or the % CPU utilization (or, equivalently, the system throughput).

Three analytical models have been chosen as suitable to perform the analysis. Each model approaches the mathematical description of the

system differently, lending insight into its appropriateness for detailed system analysis. It is observed that a nice feature of this multi-pronged modeling effort is the ability to validate the predictive capacity of the models.

The formulation of the analytical models will include the design of three continuous time Markov models. In each application, the system representation will be presented, the mathematical analysis will be developed, and the technique for solution demonstrated. The results obtained for performance prediction of the CP-67 like multiprogramming module will be presented in a separate chapter.

4.2 Independent Continuous Time Markov Models

It has been stated that the total cost of running a job in the multiprogramming environment consists of the non-overlapped processor time resulting from CPU execution, paging activity, and I/O activity. Thus, our modeling approach must isolate these three characteristics so that their individual effect on the cost function can be observed.

4.2.1 System Representation

In order to effect the isolation of activities, it will be assumed that the I/O behavior and the paging behavior can be separated. This can be visualized in figures 4-1 and 4-2 where the programs are run on separate computers that monitor the above two activities. Consequently, for a given level of multiprogramming, one can think of a job executing and requesting input/output service or suffering page faults. As ex-

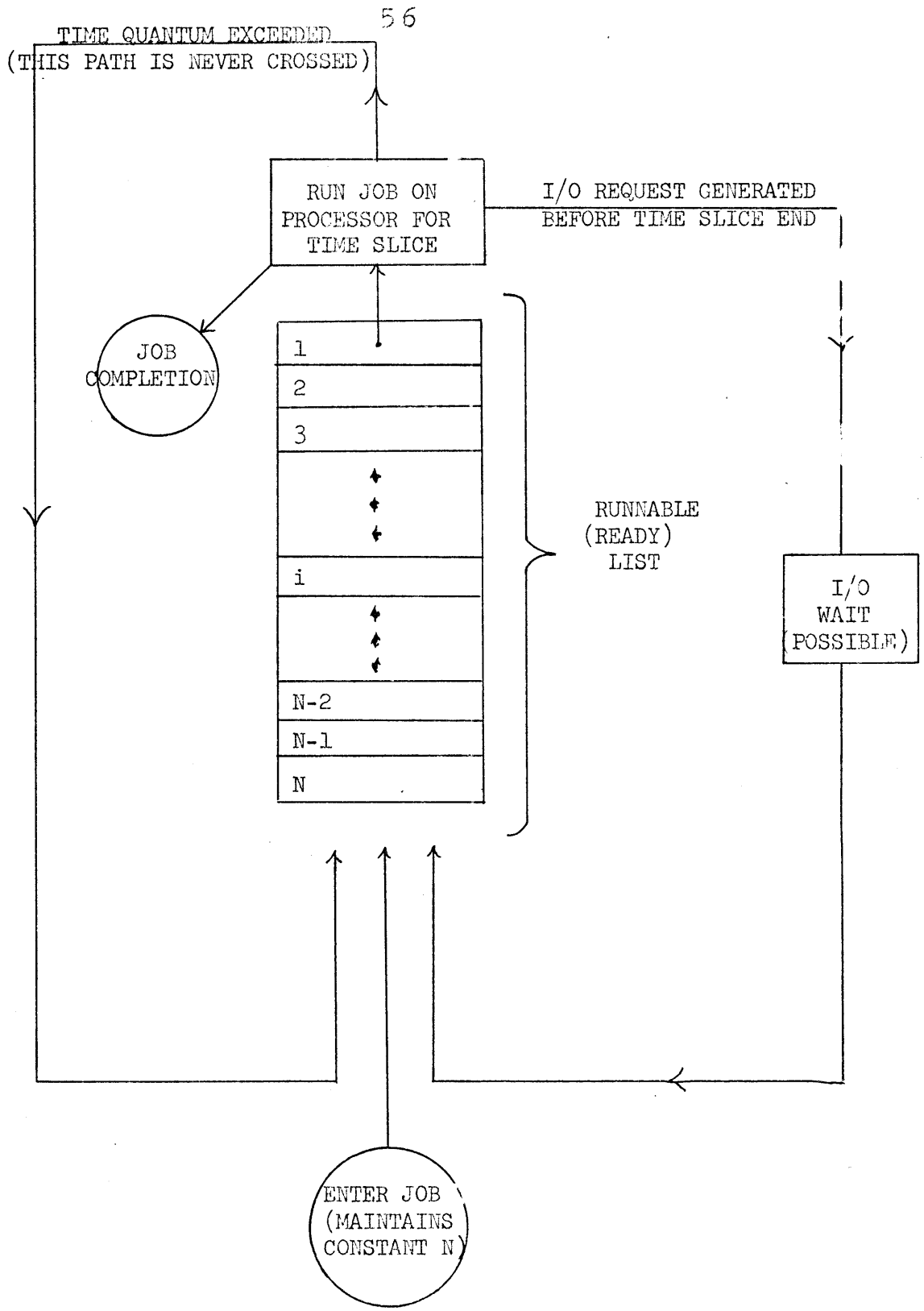


Fig 4-1 Conceptual Model Of Process Activity In A Multiprogramming Environment With I/O Wait

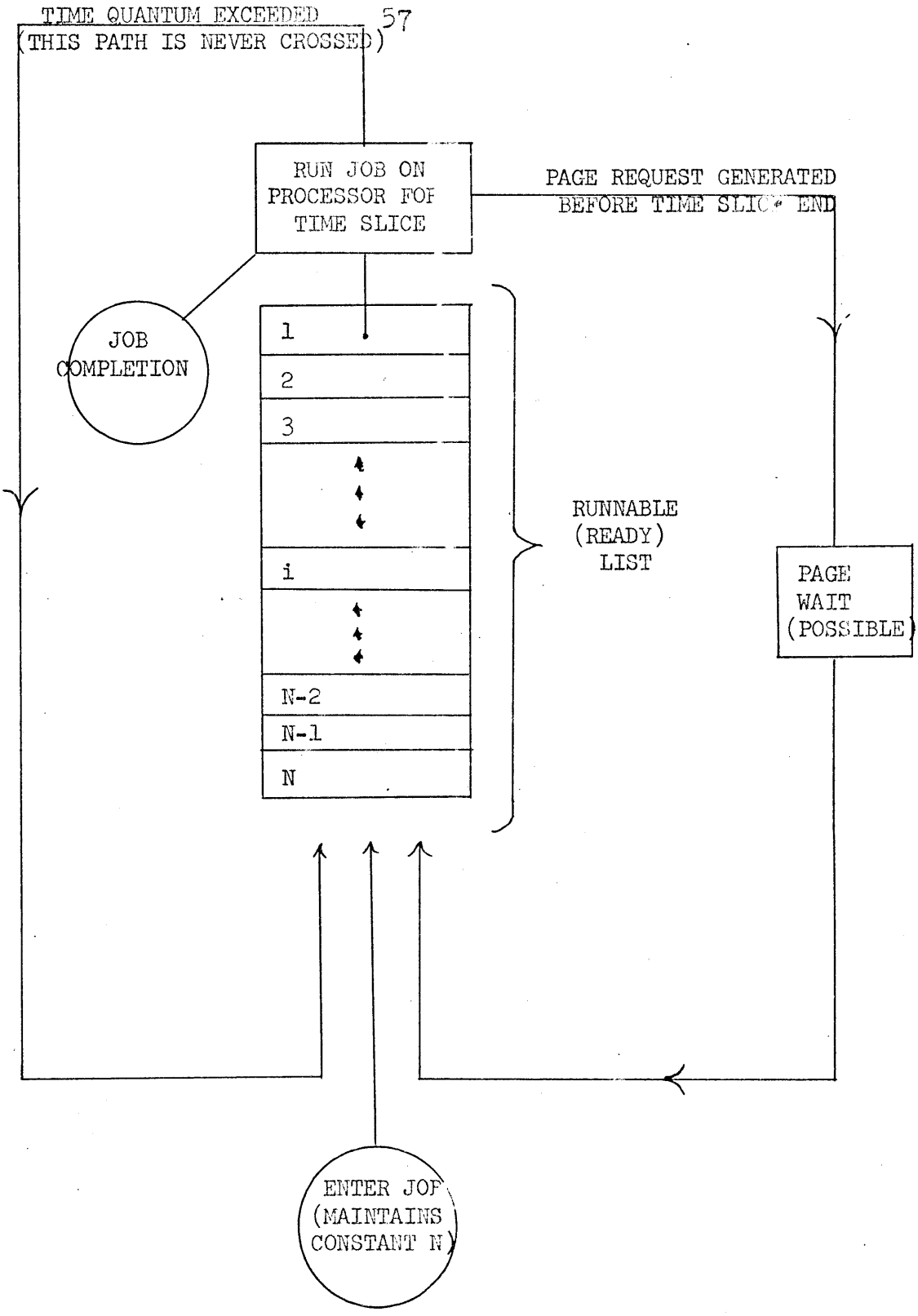


Fig. 4-2 Conceptual Model Of Process Activity In A Multiprogramming Environment With Page Wait

plained in section (3.3), the quantum exceeded path will not be traversed since it is assumed that a job issues a service request before such a time limit is reached. The block representing the wait state is intended to include any possible queuing for service as well as the actual time spent serving the request. When a service is complete, the associated process returns to the CPU ready queue for its next processing interval. Upon job completion, another job is asumed to be ready to take its place, maintaining the full load level of multiprogramming, N. Treating the activity independently is an abstraction one might prefer to live without; however, this restriction will be lifted in a subsequent analysis and it will be shown that the results obtained are accurate. It should also be mentioned that this approach does have pedgogical value in that the individual components of the composite function can be observed under varying conditions.

4.2.2 Specification of the Composite Cost Function

Calculation of the components of the T_{Total} cost function for a job in the multiprogramming environment will now be explained. The need will arise for continuous time Markov models to determine information about the system important to the final evaluation of these components.

T_{CPU} is constant for the job as one would expect. This is true because no sharing of the processor is allowed when it is dedicated to a jobs execution. Thus, T_{CPU} is 2 sec., since this is the amount of CPU time needed by each program to complete its computations.

$T_{I/O}$ is the expected cost in non-overlapped processor time that results from a jobs I/O activity. It is easiest to explain this calculation

by first observing the mono-programmed environment. In this regard, an important factor that must be evaluated in both of these remaining cost analyses is the time that the job expects to spend in the system without the benefits of multiprogramming. For the class C job explained in sect. (3.3.1), there are 2 sec. of CPU time and 3 sec of I/O wait time for a total of 5 sec. to quit the system. Note, the I/O wait time can be thought of as being generated by multiplying the 60 I/O's the job issues by .05 sec per service for a total of 3 sec. This interpretation will be of aid in the T_{PAGE} calculation that follows. The need for this factor is now evident. Given the steady state fraction of system idle time that results in the I/O environment, then $T_{I/O}$ for the job is simply:

$$T_{I/O} = 5 \text{ sec.} * \{\text{the steady state fract. of CPU idle time which is the probability of non-overlapped CPU time} = P_{N.O.}\}$$

For the mono-programmed case, if $P_{N.O.}$ is .6, then $T_{I/O}$ is 3 sec. Now, the cost of running a job in this environment decreases as the degree of multiprogramming increases, since there is a greater chance of overlapping the execution of a job in the CPU ready queue with one that has just gone blocked for I/O service. For example, we might find $P_{N.O.}$=.4 when N=2, with a corresponding $T_{I/O}$ of 2 sec. This means that 1 sec. of the time previously spent entirely in I/O wait has now been spent executing another job; this time is therefore, not charged to the users cost function.

It is also apparent from this discussion that when I/O requests meet execssive queuing delays in the I/O channel, the decrease in the cost function for the job will begin to level off as the benefits of multiprogramming are lost. In a real system where overhead for I/O set-ups is a time factor that plays a role in the cost of a job, one would certainly expect the cost function to begin to rise again as the devices were saturated and the steady state fraction of idle time for the system increased incr-

dinately.

In summary, the cost in non-overlapped processor time for a job in an environment where the only wait is for I/O service is:

$$T_{I/O} = 5 * (P_{N.O.})$$

where $P_{N.O.}$ is the steady state fraction of time that the CPU is idle in the multiprogrammed system of degree N.

T_{PAGE} involves a slightly more complicated calculation. The reason for this is that the expected time that a program would spend in the system to complete its operations is a function of the degree of multiprogramming; for, as the level of N increases, the number of page faults suffered increases, and so the time spent in the system for each job increases. Consequently, the time factor needed for the paging analysis that is analogous to the 5 sec. used in the I/O analysis (remember, in that situation, a job expected to spend 3 sec. of I/O time independent of N) must be re-evaluated for each level of multiprogramming.

In order to calculate this value, it is necessary to determine the number of page faults at the particular value of N from the static parachor. This number must then be multiplied by the time for each request to be serviced. The result is the time the user job would expect to spend for page waiting. Finally, adding 2000 msec to this gives the total time that the user will spend in the system to complete without the benefits of multiprogramming. This factor, when multiplied by $P_{N.O.}$ for the paging environment, yields the paging cost value for the user when run in the system.

Sumarizing this calculation for the stand alone environment is now done by example. Assume that a page service takes 25 msec and that for N=1, the static parachor has yielded 40 page faults for the job. So,

the total paging time is then:

$$\text{paging time} = 40 * 25 = 1000 \text{ msec.}$$

and, the multiplicative factor which is the total time spent in the system for that level of N, without the benefit of multiprogramming is:

$$\begin{array}{rl} \text{total time in system} = & 2000 \\ + & \underline{1000} \\ & 3000 \text{ msec.} \end{array}$$

then

$$T_{PAGE} = 3000 * P_{N.O.}$$

If $P_{N.O.}$ is found to be 1/3 then T_{PAGE} will be 1000 msec. In other words, the job would need 3 sec to clear the paging environment, 1 sec. of which would be time that the processor was not able to overlap its activities due to page faulting.

In this system, as the degree of multiprogramming increases, for a time there will be a greater chance to overlap execution of ready jobs with those of waiting jobs; however, it is also apparent that the load on the paging device increases as the jobs begin to compete for primary memory (the system will be paging at a higher rate as the level of N increases). Consequently, it will be possible to observe an increase in the paging cost function for a job as less and less CPU time can be spent executing in a useful mode. Thus, the benefits of multiprogramming are lost as the jobs are more likely to be found waiting for page service, and system degredation sets in.

The key value needed to evaluate the T components has been found to be the steady state probability of CPU idle time. The next logical step is to model these systems in order to determine this variable.

4.2.3 The Continuous Time Markov Models

It now remains to show how to calculate the fraction of CPU idle time that occurs in each multiprogramming environment. This will be done by modeling the behavior of the separate systems as continuous time Markov processes. Figures 4-3 and 4-4 show the state descriptions that are appropriate for the analysis. The value of the state is the number of processes queued for service at the peripheral device. Each state can be completely described by a two tuple (W,E), where W is the number of processes waiting to complete service on the device, and E is the number of processes executing. For this single CPU system, this number is either 1 or 0.

The parameters displayed in the figures that are essential to the analysis that follows have already been partially specified. It has been stated that the service disciplines will be characterized by exponential distributions. Thus, we have the average arrival rate of service completions:

let $E(t_c)$= the expected interarrival time for a service completion

then,

$$U = \frac{1}{E(t_c)}$$

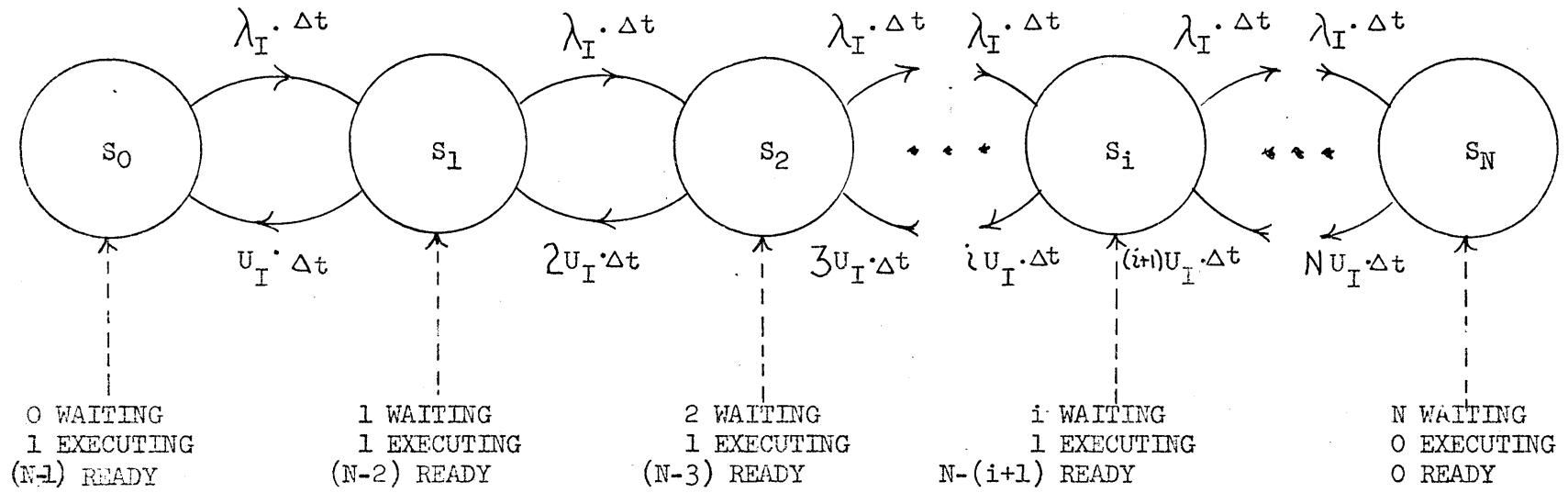
for the drum or the disk. This permits us to specify the transition probabilities for the return paths in the model. Adopting the point of view that arrivals occur at points on a continuous line, for suitably small values of Δt,

let c= a service completion arrival

then,

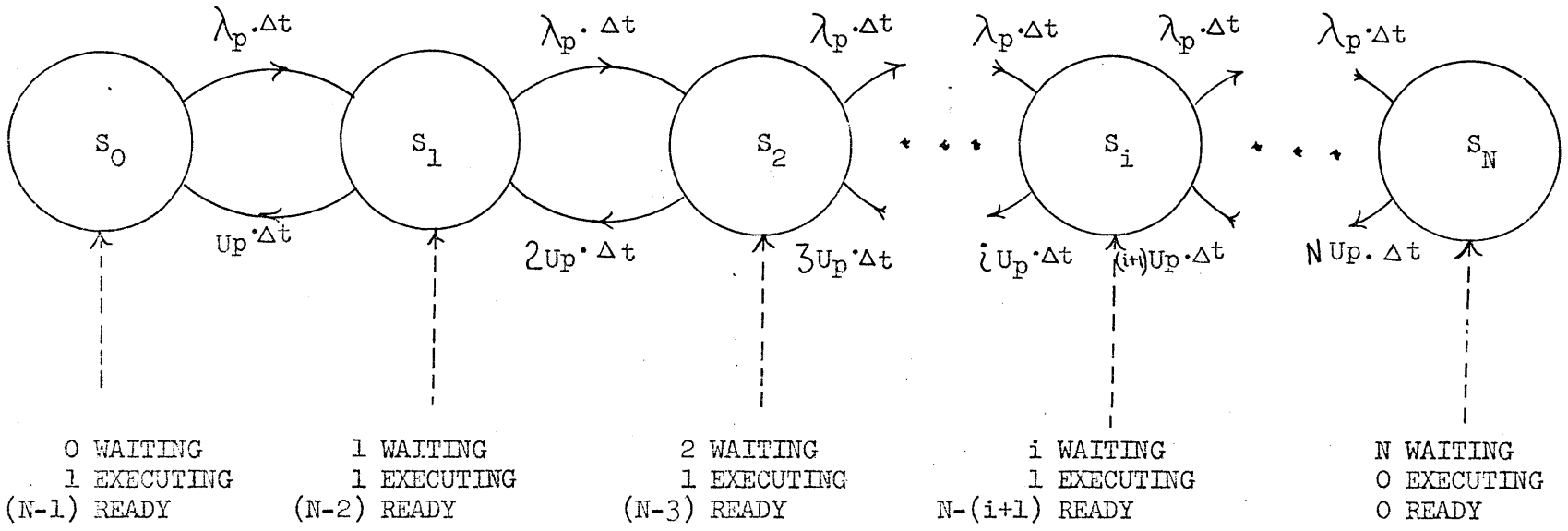
$$P(c/\Delta t) = U \cdot \Delta t$$

Now it remains to characterize the I/O and paging behavior of the jobs



WHERE: λ_I is the I/O REQUEST RATE
U_I is the I/O SERVICE RATE

Fig. 4-3 Markov Digraph For The State Transitions In A Multiprogramming Environment With I/O Blocking



WHERE: λ_p is the PAGE FAULT RATE
U_p is the PAGE SERVIC RATE

Fig.4-4 Markov Digraph For The State Transitions In A Multiprogramming Environment With Page Blocking

as transition probabilities of the Markov models. For the purposes of this discussion, it will be assumed that a program's requests for page or I/O service are also exponentially distributed. We have enough information about the programs to determine the expected interarrival times for requests. So it is then an easy matter to specify the foward transition probabilities.

let $E(t_r)$= the expected interarrival time for a service request
$E(t_{I/O})$=2000 msec./#I.O.'s=2000/60
$E(t_{PAGE})$=2000 msec./#Page Faults=2000/f(N)

then

$$\lambda = \frac{1}{E(t_r)} \begin{cases} \lambda_{I.O.} = 60/2000 \\ \lambda_{Page} = f(N)/2000 \end{cases}$$

for a page or I/O request. In the same manner as above, this permits us to specify the one-step transition probabilities for the foward paths. For suitably small values of Δt,

let r= a service request

then,

$$P(r/\Delta t) = \lambda \cdot \Delta t$$

Now, the description of the models is complete.

4.2.4 The Steady State Solution

The behavior of such closed-commuting chains are known to possess a steady state (F1,D7,C10,T3). Let π_i (i=0,1,2,...,N) be the probability of finding the system in state i after operation for a long time. These values can be found by one of the well known techniques for solving Markov models of this class. Birth-Death equations can be used (F1,D7,C10,T3), transform analysis can be performed (T3), or differential difference equations can be solved (F1,D7).

The closed form solution available from the first two suggested

techniques is known as the Erlang loss formula (C7):

$$\pi_n = \frac{1}{n!}\left(\frac{\lambda}{U}\right)^n \cdot \frac{1}{\sum_{i=0}^{n} \frac{1}{i!}\left(\frac{\lambda}{U}\right)^i} \qquad \text{Eqn.4-1}$$

The third technique will be used here to solve for the steady state probabilities. The value of using this method is that it affords the benefit of demonstrating the technique that will be applied to solving the dual facility continuous time Markov model in the next section (4.3), where no known closed form solutions exist. It also helps preserve the equality of the two studies.

Equations 4-2,4-3,and 4-4 show the equations used to solve this model. Eqn. 4-2 is the set of differential difference equations for the CTMP described above. Equation 4-3 represents the system of equations in the steady state. The derivatives that represent the change of the probability of being in a state with respect to time have been set to zero, an implicit condition of equilibrium. Eqn.4-4 is the matrix representation of 4-3 for the case N=5.

This set of equations lends itself to solution on a digital computer. The CPU idle time analyzer is a computer program that was written to solve the set of simultaneous equations generated for the general case of such models. A description of the program, an extended program listing, and sample output are available in Appendix A.

4.2.5 Applying the Solutions

Now that the steady state solutions are available, the evaluation of

$$\frac{dP(0,t)}{dt} = U \cdot P(1,t) - \lambda \cdot P(0,t) \qquad n = 0$$

$$\frac{dP(n,t)}{dt} = \lambda \cdot P(n-1,t) + (n+1)U \cdot P(n+1,t) - (\lambda+nU) \cdot P(n,t) \qquad 0 < N < N$$

$$\frac{dP(N,t)}{dt} = \lambda \cdot P(N-1,t) - N \cdot U \cdot P(N,t) \qquad n = N$$

Eqn. 4-2 Differential Difference Equations for Single Facility Continuous Time Markov Process (immediate service/req.=Optimal)

$$0 = U \cdot \pi_1 - \lambda\pi_o \qquad n = 0$$

$$0 = \lambda \cdot \pi_{n-1} + (n+1) \cdot U \cdot \pi_{n+1} - (\lambda + nU)\, \pi_n \qquad 0 < n < N$$

$$0 = \lambda \cdot \pi_{n-1} + N \cdot U \cdot \pi_n \qquad n = N$$

Eqn. 4-3 Equilibrium Equations for the Single Facility CTMP

π_o	π_1	π_2	π_3	π_4	π_5	
$-\lambda$	U					$= 0$
λ	$-(u+\lambda)$	$2U$				$= 0$
	λ	$-(2U+\lambda)$	$3U$			$= 0$
		λ	$-(3U+\lambda)$	$4U$		$= 0$
			λ	$-(4U+\lambda)$	$5U$	$= 0$
				λ	$-5U$	$= 0$

$$\text{s.t.} \quad \sum_{i=0}^{5} = 1$$

Eqn. 4-4 Example Matrix Description of CTMP : N=5, 1 CPU

the T_{TOTAL} cost function is possible. This can be done by utilizing the value of π_N, in fact, represents the steady state CPU multiprogramming idle time, $P_{N.0.}$. A point on the cost function is generated by "running" the model for a particular value of N, calculating T_{CPU}, $T_{I/O}$, and T_{PAGE} as explained above, and summing the three values. The total cost function is generated by"running" the model for various levels of N.

The results obtained from the application of this set of models to the analysis of the CP-67 like multiprogramming module are presented in chapter 6.

4.3 The Dual Facility Continuous Time Markov Model

As pointed out, the treatment of the paging and I/O activities as occuring in two separate computer environments is a reasonable differentiation to make in a pedgagocial setting; however, it is desirable to take a more reasiltic view of program activities in order to obtain more accurate performance projections.

4.3.1 System Representation

A conceptual view of the proposed system is shown in figure 4-5. Similar assumptions about the path selections and blocking functions hold for this dual facility model. The only difference in this approach is that the activities are assumed to occur within the same computer. So, it can be assumed that a job executing on the processor can suffer a page fault or request I/O service during his burst of processor time. The same rates of request and service apply that were used in the independ-

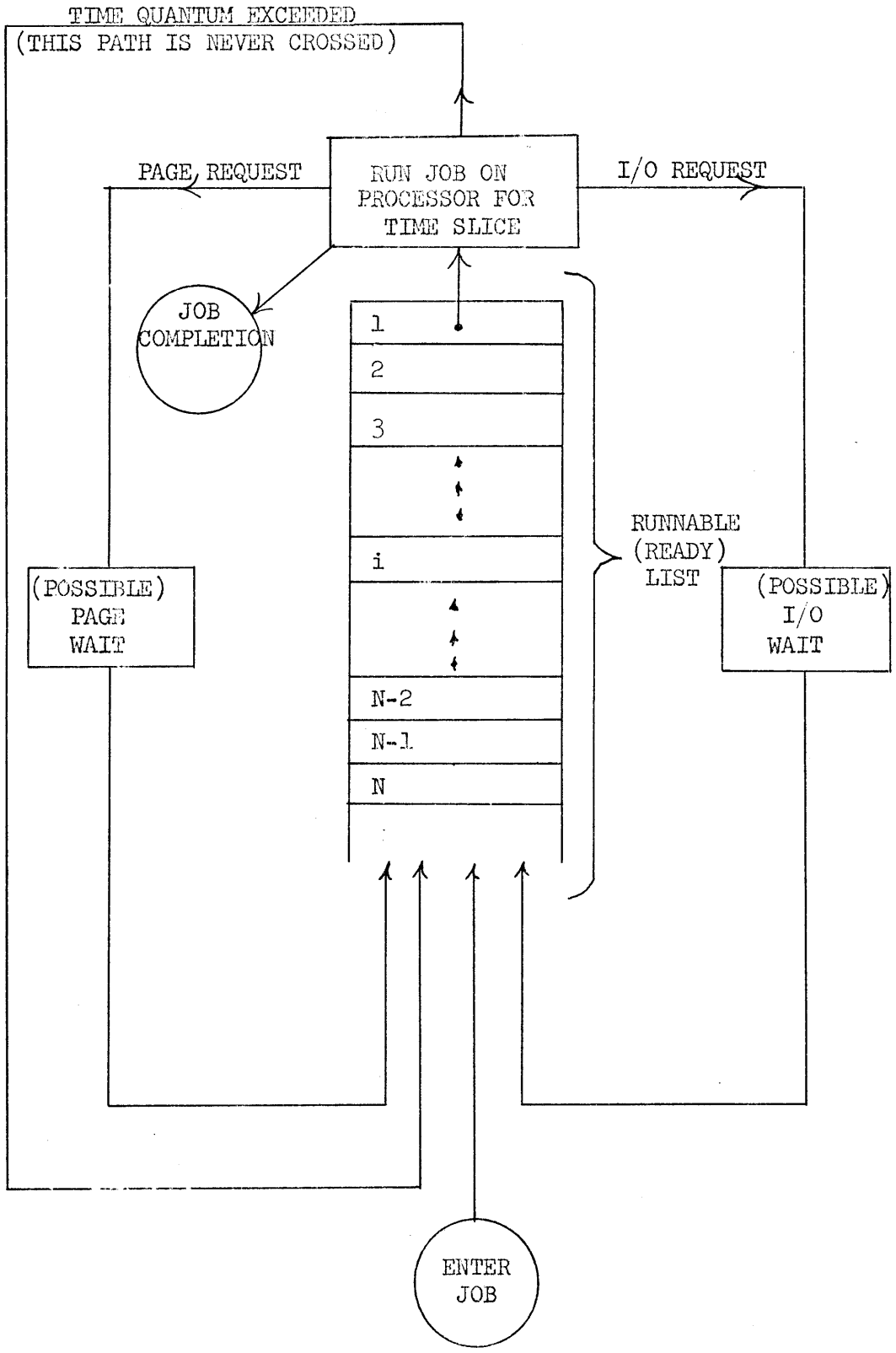


Fig. 4-5 Conceptual Model Of Process Activity In A Multiprogramming Environment With Page (+) I/O Blocking

dent study(sect.4.2.3). Thus, the one-step transition probabilities for the Markov digraph agree. The continuous time Markov model that represents this extended environment is shown in figure 4-6 . Here, a states value is the number of processes waiting for I/O service and page service. A three-tuple is required to describe a state completely. This vector is (W_I,W_P,E) where W_I is the number in the I/O wait state, W_P is the number in the paging wait state, and E is the number of processes executing.

4.3.2 The Steady State Solution

Determining the differential difference equations for this model is slightly more involved. There are seven conditions which must be included to specify the model. These include 3 end points, 3 boundaries, and 1 central condition. The specific equations are shown in Eqn.4-5. Their steady state representation is displayed in Eqn.4-6. The program written to solve the single channel models has been designed to include the general case of this double barreled Markov digraph as well. The same PL/1 subroutine, MLSQ, is used to solve the set of N simultaneous equations. Appendix A displays sample output obtained from application of the solution technique outlined above.

4.3.3 Applying the Solutions

Obtaining the steady state solutions for each system state is the first step of the analysis. A variety of marginal distributions can then be determined that are useful for the study of the multiprogramming system. For example, summing the probabilities over all

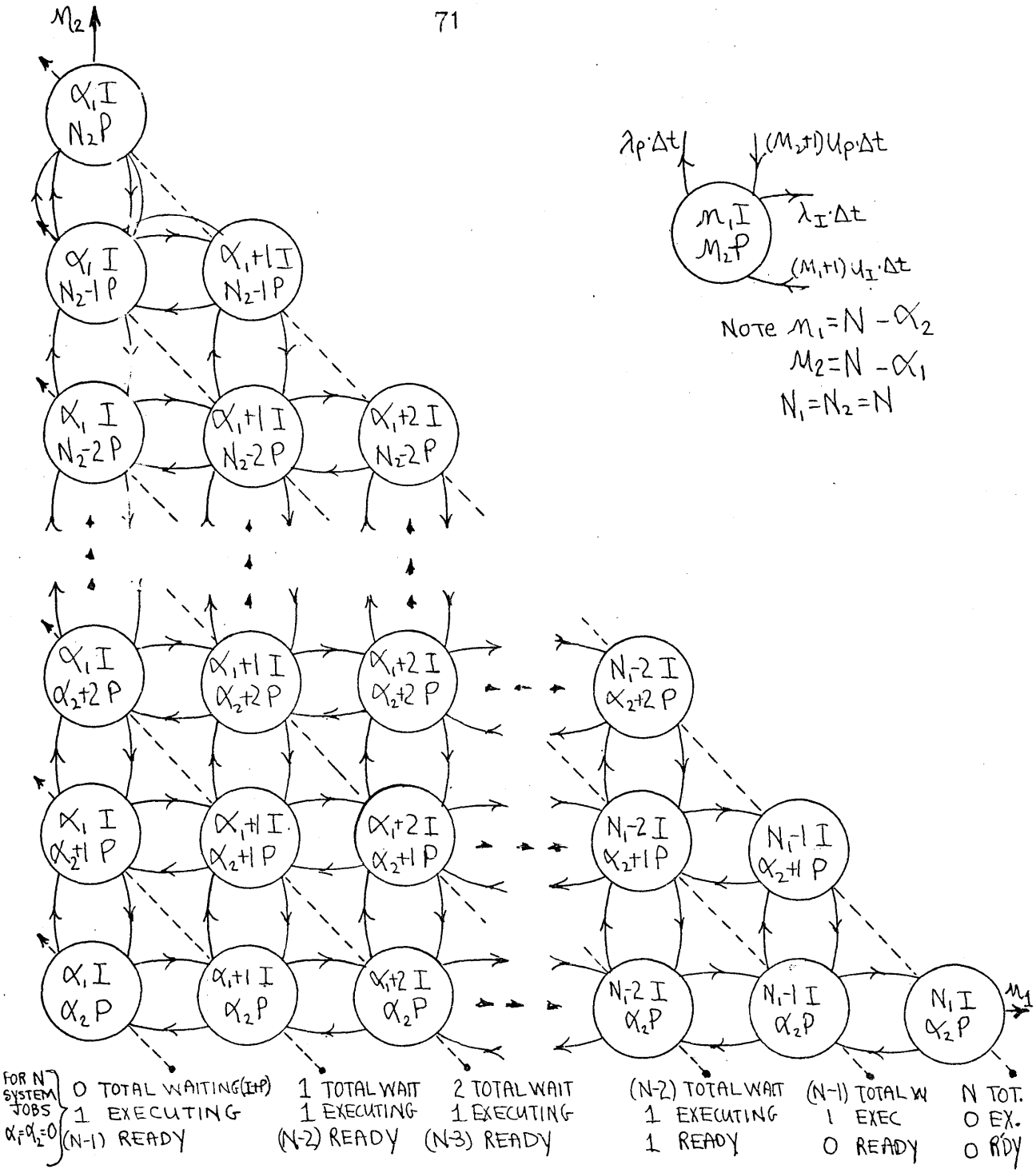


fig. 4-6 MARKOV-DIGRAPH FOR THE STATE TRANSITIONS IN A MULTIPROGRAMMING ENVIRONMENT WITH PAGE AND I/O TRAFFIC (THE GENERAL CASE)

$$\frac{dP}{dt}(0,0,t) = U_I P(1,0,t) + UpP(0,1,t) - (\lambda_I+\lambda_p)\, P(0,0,t) \quad \ldots\ n_1=n_2=0$$

$$\frac{dP}{dt}(N,0,t) = \lambda_I P(N-1,0,t) - NU_I P(N,0,t) \quad \ldots\ n_1=N, n_2=0$$

$$\frac{dP}{dt}(0,N,t) = \lambda_p P(0,N-1,t) - NUpP(0,N,t) \quad \ldots\ n_1=0, n_2=N$$

$$\frac{dP}{dt}(0,n_2,t) = \lambda_p P(o,n_2-1,t) + (n_2+1)\, U_p P(0,n_2+1,t) \quad \ldots\ n_1=0$$

$$+ U_I P(1,n_2,t) - (n_2 U_p + \lambda_I + \lambda_p)\, P(0,n_2,t) \quad 0<n_2<N$$

$$\frac{dP}{dt}(n_1,0,t) = \lambda_I P(n_1-1,0,t) + U_p P(n_1,1,t) + U_I P(n_1+1,0,t) \ldots\ 0<n_1<N$$

$$- (n\, U_I+\lambda p+\lambda_I)\, P(n_1,0,t) \quad n_2=0$$

$$\frac{dP}{dt}(n_1,n_2,t) = \lambda_I P(n_1-1,n_2,t) + \lambda_p P(n_1,n_2-1,t) \quad \ldots\ n_1=N-n_2$$

$$-(n_1 U_I + n_2 U_p)\, P(n_1,n_2,t) \quad 0<n_2<N$$

$$\frac{dP}{dt}(n_1,n_2,t) = \lambda_I P(n_1-1,n_2,t) + \lambda_p P(n_1,n_2-1,t) \quad \ldots\ n_1,n_2$$

$$+ (n_1+1)U_I P(n_1+1,n_2t) + (n_2+1)U_p P(n_1,n_2+1,t)$$

$$- (n_1 U_I + n_2 U_p + \lambda_I+\lambda_p)\, P(n_1,n_2,t)$$

Eqn. 4-5 Differential Difference Equations for Combined I/O and Page Model (immediate service / request)

$$0 = U_I \pi_{1,0} + U_p \pi_{0,1} - \{\lambda_I + \lambda_p\} \pi_{0,0} \quad \ldots n_1=0, n_2=0$$

$$0 = \lambda_I \pi_{N-1,0} - NU_I \pi_{N,0} \quad \ldots n_1=N, n_2=0$$

$$0 = \lambda_p \pi_{0,N-1} - NU_p \pi_{0,N} \quad \ldots n_1=0, n_2=N$$

$$0 = \lambda_p \pi_{0,n_2-1} + (n_2+1)U_p \pi_{0,n_2+1} + U_I \pi_{1,n_2} - \{(n_2) U_p + \lambda_I + \lambda_p\} \pi_{0,n_2} \quad \ldots n_1=0, 0<n_2<N$$

$$0 = \lambda_I \pi_{n_1-1,0} + U_p \pi_{n1,1} + (n_1+1) U_I \pi_{n1+1,0} - \{(n_1) U_I + \lambda_p + \lambda_I\} \pi_{n1,0} \quad \ldots 0<n_1<N, n_2=0$$

$$0 = \lambda_I \pi_{n_1-1,n_2} + \lambda_p \pi_{n_1,n_2-1} - \{(n_1)U_I + (n_2) U_p\} \pi_{n_1,n_2} \quad \ldots n_1=N-n_2, 0<n_2<N$$

$$0 = \lambda_I \pi_{n_1-1,n_2} + \lambda_p \pi_{n_1,n_2-1} + \{n_1+1)\} U_I \pi_{n_1+1,n_2} + \{(n_2+1)\} U_p \pi_{n_1,n_2+1} - \{(n_1) U_I + (n_2) U_p + \lambda_I + \lambda_p\} \pi_{n_1,n_2} \quad \ldots n_1, n_2$$

Eqn . 4-6 The Equilibrium Equations for the Combined Model

the states where the CPU is active,

$$\sum_{\text{s.t. } ni+np<N} \pi(ni,np)$$

yields the steady state fraction of time that the CPU is active: the CPU utilization factor proposed as a barometer for measuring system performance. It is also possible to determine the expected queue lengths that form for the shared resources by performing sums of the following nature,

Let P_{Ni} = probability that the queue length is=Ni

for the CPU: $E(\text{queue CPU}) = \sum_{i=0}^{N} Ni \cdot P_{Ni}(CPU)$

for the drum: $E(\text{queue drum}) = \sum_{i=0}^{N} Ni \cdot P_{Ni}(Drum)$

for the disk: $E(\text{queue disk}) = \sum_{i=0}^{N} Ni \cdot P_{Ni}(Disk)$

A further discussion of the relevence of these performance measures as well as results obtained from the application of this model to the analysis of the multiprogramming module are presented in Chapter 6. Let it be noted here that although determination of a cost function is possible for this combined module, it will not be necessary for the analysis. Pinpointing system degredation will be a simple matter by dint of CPU utilization.

4.4 The Queuing Network Model

In a further effort to qualify the analysis of the System Process Model of multiprogramming behavior, one final analytical model will be proposed that is of both practical and theoretical interest in the field

of computer performance analysis. This model represents SPM as a network of queues and asso iated servers. Presentation of this other viable technique for analyzing our multiprogrammed, virtual memory computer lends insight into the current approaches in systems analysis.

4.4.1 System Representation

The reader is referred back to chapter 4 for a review of the computer and programs to be modeled. Some additional comments about particulars of this system relevent to the representation of the system as a network of queues will now follow.

The view of process activity will be augumented by adding one additional interruption type. So, in addition to the I/O requests scattered throughout execution and page faults suffered in the"squeezed core" environment, jobs will be assumed to contain a symbolic stop as their last statement. Execution of this statement within a memory partition causes processing there to be discontinued. A new program is loaded and executed in its place at the instant when the processor returns to processing in that partition. This view is entirely consistent with our assumption of full load. Also, because of the assumptions that instantaneous switching of the processor occurs and that initial page loading is a part of the new programs general activity, this explicit characterization of job completions does not make this model different from the continuous time Markov representation. Remember, in that analysis, program completion was an implicit characteristic of the state space representation.

The conceptual view of this system is shown in figure 4-7. It is

noted that the new program path always maintains a constant level of multiprogramming, since this path returns jobs to the ready queue for loading and execution as a new job.

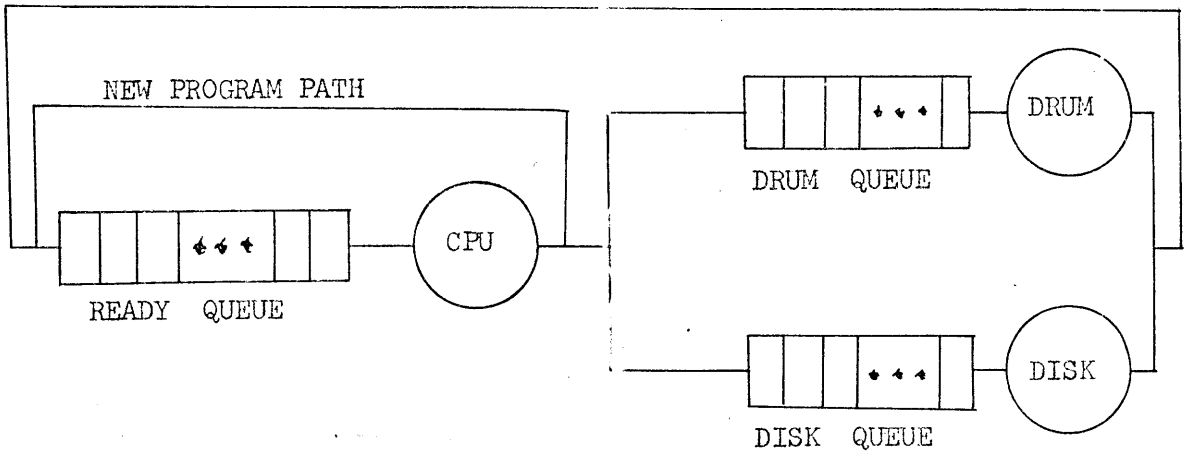


Fig.4-7 The Queuing Network Model of Multiprogramming

4.4. Parameterization of the Network

Adopting the notion that each memory partition represents a token that flows in the system competing for the shared resources, the need to specify a mechanism for path selection arises. Thus, upon completion of a CPU processing interval. determination of the next activity for the process is made in statistical selection mode. To motivate this choise of path determination, consider the relative frequency interpretation of probability. Observing with Buzen (B3), if the system being studied was observed for a long time under full load, one might find the following distribution of device requests upon completion of a CPU processing interval:

Path Destination	Relative Frequency
path to disk(I/O) - - - - - - - -	300/1000
path to drum(page) - - - - - - - -	600/1000
new program path - - - - - - - -	100/1000

It can then be said that, when a process leaves the CPU, it has a probability of 3/10 of selecting the disk path, a probability of 6/10 of selecting the drum path, and a probability of 1/10 of selecting the new program path. The selection mode is then specified by the probabilities p0, p1, and p2, where p0 represents the disk path, p1 the drum, and p2 the new program path. This is shown below in figure 4-8. One can think

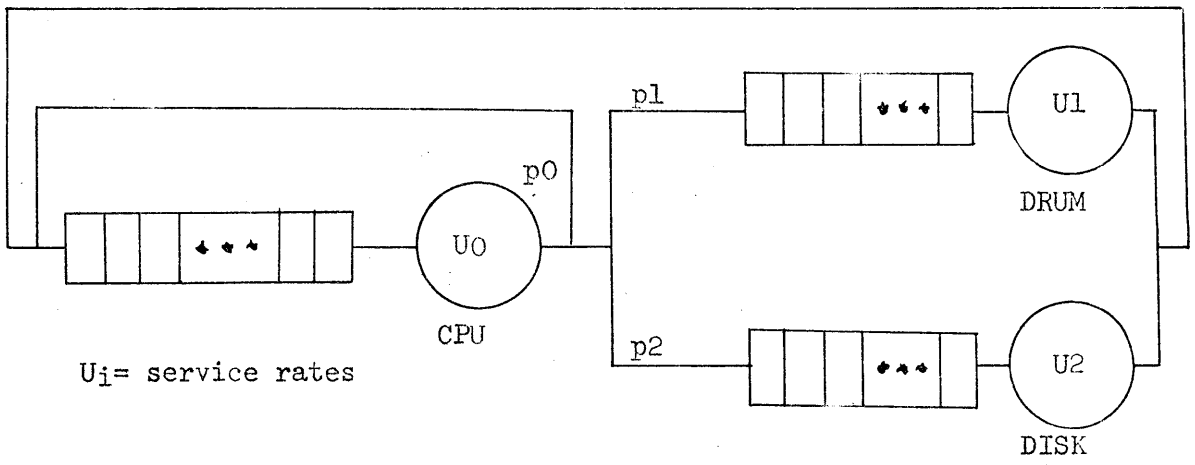


Fig. 4-8 Probabilistic Path Selection In the Queuing Network

of the device selection probabilities as representing the manner in which the system utilizes its resources. Although the actual program behavior is not individually depicted, the statistical behavior of the system is consistent with the previous model representation of a continuous time Markov process. Device service rates shown are again assumed to be expo.dist.

In order to quantify these probabilities for our system, reference is made to the discussion of elementary program properties in the Buzen thesis. There one finds specification of the distribution of the number of processing requests per program, the total processing time per program, and the number of service requests for the CPU as functions of the path selection probabilities and mean service rates of the devices. The following is a summary of these results (B3, B4):

(1) Expected number of service requests in a program for the jth peripheral processor=pj/p0

(2) Expected total service time per program on the CPU =1/U0*p0

(3) Expected number of service requests in a program for the CPU=1/p0

Now, referring to the program behavior parameters of the example program as specified in section (3.3.1), observe that the following is true:

let Ki= the # of requests to a peripheral processor

K1= the # of requests to the drum

K2= the number of requests to the disk

then,

K1= f(N)

K2= 60

and,

K1 + K2= the total number of I/O's of both types in the program

K1 + K2 +1= the total number of CPU requests / program

Utilizing these observations, specification of the key parameters U0, p0, p1, and p2 (U1 and U2 are already specified) is possible:

remember, the programs execute for 2000 msec.

$$2000 = (60 + K1 + 1)/U0$$

$$/\overline{U0 \neq} \ (60+ K1 + 1)/2000$$

$$1/p0 = 60 + K1 + 1$$

$$/\overline{p0 \neq} \ 1/(60 + K1 + 1)$$

$$p1 = K1*p0$$

$$/\overline{p1 \neq} \ K1/(60 + K1 + 1)$$

$$p2 = 60*p0$$

$$/\overline{p2 \neq} \ 60/(60 + K1 + 1)$$

The dependence of the CPU processing interval allotted to each program on the page faults predicted by the static parachor is evidenced in the expression for U0. It is clear from the equation that the mean service time per interval decreases as the number of page faults increases.

4.4.3 The Steady State Solution

Now that the basic model has been specified, the next step is to determine the steady state probability distribution for the system. This well known result was initially established by Gordon and Newell (G5) and later applied to computer modeling by Buzen, Moore and Rice (see section (2.4)).

In order to explain this solution method, the following mathematical description of the computer system is adopted:

L+1 = the number of servers in the network

Uj = the processing rate of the jth server

pij = the probability that a process leaving the ith server will proceed to the jth server. Note that

$$\sum_{j=0}^{L} pij = 1 \quad i=0,1,2,\ldots,L$$

N = the number of customers in the system, i.e. the degree of multiprogramming

For the queuing network, a customer leaving the ith server will proceed to the jth server according to the following transition matrix:

$$P=\begin{bmatrix} p0 & p1 & p2 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

For this system, let P(n0,n1,n2) be the steady state probability of finding nj customers present at the jth server, where $\sum_{j=0}^{L} nj=N \quad 0<nj<N$. Now define the state indicator random variable,

$$e(nj) = \begin{cases} 0 & \text{if } nj=0 \\ 1 & \text{if } nj\ 0 \end{cases}$$

Then, viewing the systems one step transition probabilities as in fig.4-9,

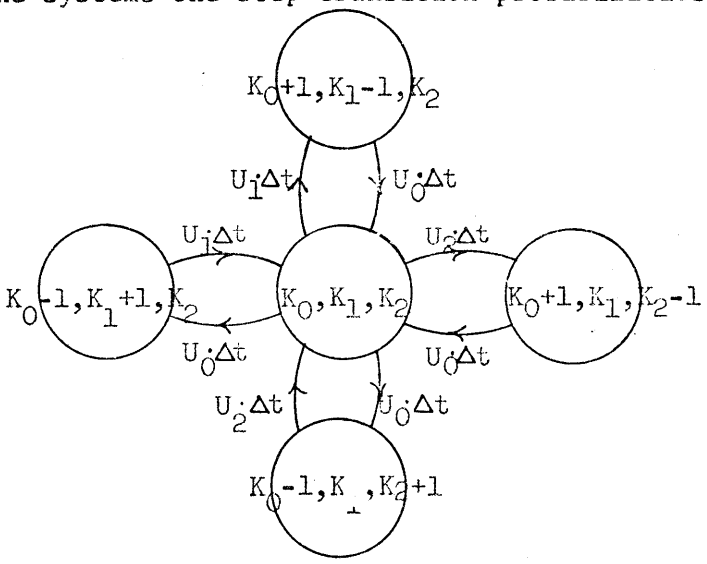


Fig.4-9 One Step Transitions for the Queuing Network

it can be ascertained that the rate of transition out of state (n0,n1,n2) at equilibrium is:

$$\sum_{j=0}^{2} e(nj)\cdot uj\cdot P(n0,n1,n2)$$

and that the rate of flow into state (n0,n1,n2) is

$$\sum_{i=0}^{2}\sum_{j=0}^{2} e(nj)\cdot ui\cdot pij\cdot P(\ldots ni+1,nj-1,\ldots)$$

Therefore, the balance equations for the continuous time Markov process are

$$\underbrace{\sum_{j=0}^{2} e(nj)\cdot uj\cdot P(n0,n1,n2)}_{\text{exit from state (n0,n1,n2) through the jth server}} = \underbrace{\sum_{i=0}^{2}\sum_{j=0}^{2} e(nj)\cdot ui\cdot pij\cdot P(\ldots ni+1,nj-1,\ldots)}_{\text{entrance into the jth server from state (...ni+1,nj-1,...)}}$$

Solution of this set of equations in the steady state is possible with a separation of variables technique. The details are available in the references (G5,B3). It follows that,

$$P(n0,n1,n2) = \frac{1}{G(N)} \prod_{j=1}^{2} (pj\cdot U0 / Uj)^{nj} \quad \text{s.t.} \sum_{n0+n1+n2=N} P(n0,n1,n2)=1$$

Thus, G(N) is the normalizing constant selected such that the sum of the steady state probabilities equals one:

$$G(N)= \sum_{\sum_{j=1}^{L} nj\le N}^{L} \prod_{j=1}^{L} (pj\cdot U0/Uj)^{nj}$$

A very efficient computational algorithm for calculating G(N) exists as a result of the doctoral work of Buzen. It is used to facilitate the evaluation of the above steady state probabilities.

4.4.4 Applying the Solution

The use of the computational algorithm yields G(N) and as by-

products, $G(N-k)$, $k=0,1,\ldots,N$. The value of these results lies in the fact that they are useful in calculating a variety of marginal distributions that are directly applicable to performance analysis. For example, Buzen has shown that the utilization of the CPU is $\frac{G(N-1)}{G(N)}$ and that the expected queue length for the CPU in the steady state is,

$$QO = \sum_{k=1}^{N} \frac{G(N-k)}{G(N)}$$

In addition, the expected queue length for the jth server is

$$Qj = \sum_{k=1}^{N} (pj \cdot UO/Uj)^{k} \cdot \frac{G(N-k)}{G(N)}$$

A PL/1 program has been written to evaluate the systems steady state probabilities and key marginal distributions. The results of this effort can be found in Appendix B where an extended program listing is shown along with output from sample runs of the model.

CHAPTER 5

SIMULATION: AN APPROACH TO SYSTEM PERFORMANCE ANALYSIS

5.1 Introduction

Simulation analysis has been established as a technique that lends itself to computer modeling, and analysists have produced valuable predictive information about system performance using this analysis technique (sections (1.2,2.3)).

Often, a computer model demands representation of a level of detail that precludes using analytical models. In such cases, a system simulation is developed as the sole model; then,if an actual system is being studied (as opposed to a feasibility study of a hypothetical system), the model can be validated on the basis of performance data provided by a real time environment. In the situation when the system isn't available, validation of the model is much more difficult. In these cases, it is possible to study the step by step operation of the simulation model in the

hopes of ascertaining its correctness. This, needless to say, may be prohibitively difficult. Another approach might be to macroscopically model system performance with an analytical model, and then perform a statistical analysis comparing results from the simulation with the mathematical results. Thus, trends that might transcend intuition could be verified and some confidence in the model as a performance analysis tool could be achieved.

In the previous chapter, analytical models were formulated to analyze a CP-67 like demand-paged computer system. Since three different techniques were available to provide results for comparison, it was possible to determine the predictive capacity and accuracy of the methods. In this chapter it will be demonstrated how a performance model of the system can be developed with simulation. Then confidence in the simulation will be gained by validating the data on the basis of the analytical models. It will then be shown how the simulation approach can be extended to model a greater level of detail and to overcome some of the shortcomings inherent in the System Process Model.

5.2 The Characteristics and Problems of Simulation Models

Developing a simulation model requires insight into a variety of issues. These will be discussed prior to the actual modeling effort. To begin, lets center on the technical issues found in model development. The first topic of importance is flexibility. It is desirable to produce a general program which permits analysis of a wide range of parameters with as little reprogramming as possible. Given this objective, the type

of simulation system to be used must be chosen. One then finds that the problems encountered in model development are very different if the system is being implemented in a general purpose, higher-level language than if it is being implemented in an available system simulator.

If starting from scratch, for example, the problem of equipment requirements is an open question. How much memory space is needed, how long will programs be, and how fast can they run? The synchronization of events that are being modeled in "real time" must be handled, as well as the maintainence of variable length lists in which entries are usually removed in different order than placed. Distribution sampling must be implemented and statistics gathering applied. Granted, organizing a simulation from inside out lends a much better understanding of system operation and can provide much more efficient operation than general systems that provide these facilities; however, since solutions to these problems are provided by simulation systems programmable in their own proprietary language, the convenience of their approach is widely taken advantage of. Let it be noted here that the models implemented in the following section are written in GPSS, a general purpose systems simulator.

Now, it is of interest to discuss those characteristics and problems common to both modeling approaches. To begin, observe that a salient feature of simulation modeling involves including components of the real system that make the modeling useful as a tool for analysis and prediction. Including componedts to make the model sufficiently real often involves highly subjective reasoning. Often, the trade off in this area is between producing simplistic, economical models and realistic, complex ones. A

conscientious effort must be made to pinpoint and modularize the essentials of the system, for all too often, the model with additional features can obscure the real value to be gained, let alone make it prohibitively expensive to run.

Another common problem revolves around the fundamental nature of systems simulation. That is to say, their discrete event reprensentation of a continuous physical process results in inherent approximations and results in the problem of simultaneity of events.

In order to appreciate the next problem inherent in simulation modeling, it is necessary to consider the nature of the behavior of individual components in the system. In a simulation model, it is the modular mathematical formulation of individual components, which when manipulated and monitored, produce the dynamic interactions of the real system. Now, component specification in computer modeling can be deterministic or stochastic. For the obvious reason that decisions and operations are executed in an inherently unpredictable environment, it is appropriate to select the probabilistic descriptions. Yet, such a choise points up the problem alluded to. Probabilistic specification leads to variability in the results; so how can one justify predictions about an environment that can only be approximately modeled?

Last but not least, it must be realized that the system simulator does not produce the answer to any question that is posed to it. In fact, it doesn't pretend to solve anything. It merely yields results descriptive of system operation in an environment specified by the programmer. The best one can hope for is to gain insight and display expected results by

"running" the simulator with a variety of parameters. This results in our final problem of simulation. How does one specify starting conditions and determine when the results have reached equilibrium?

These problems will be handled in the implementation to follow. The reader is referred to the references for more detailed discussions and solutions to these intrigueing problems (C8,C9,R1).

5.3 Simulation of the System Process Model

As indicated in the introduction, the System Process Model will be simulated in order to demonstrate the application of this technique to computer system analysis. Extensions of the model will be shown to overcome some of the weaknesses inherent in SPM.

5.3.1 Statement of the Problem

The problem to be tackled here is similar to the one studied with analytical models in Chapter 5. A GPSS model is proposed to provide information on multiprogramming operation in a page on demand virtual memory computer system. It is intended to demonstrate how simulation modeling can lend insight into system operation by obtaining a measure of system throughput, turn around time, device utilizations, and queue length statistics.

The simulation development phase can be viewed as an effort to produce another valid model of SPM using a different modeling tool. A point of departure in the modeling effort that permits demonstration of the more flexible nature of simulation is established when the results of the basic

simulation model are shown to agree with the analytical results obtained for SPM. Additional features that one might add to the model such as the inclusion of more detailed subsystems, system overhead, the use of non-exponential service disciplines, and scheduling of CPU requests are discussed. It is intended to show how the inclusion of such detail makes the SPM representation of multiprogramming more realistic.

Before discussing the actual model development, a brief review of the basic system that is initially modeled is of value. The nature of the system is depicted in figure 5-1. Note that the movement of tokens described in the analytical models is physically modeled in the simulation approach. These can be seen as the circles that represent user processes:

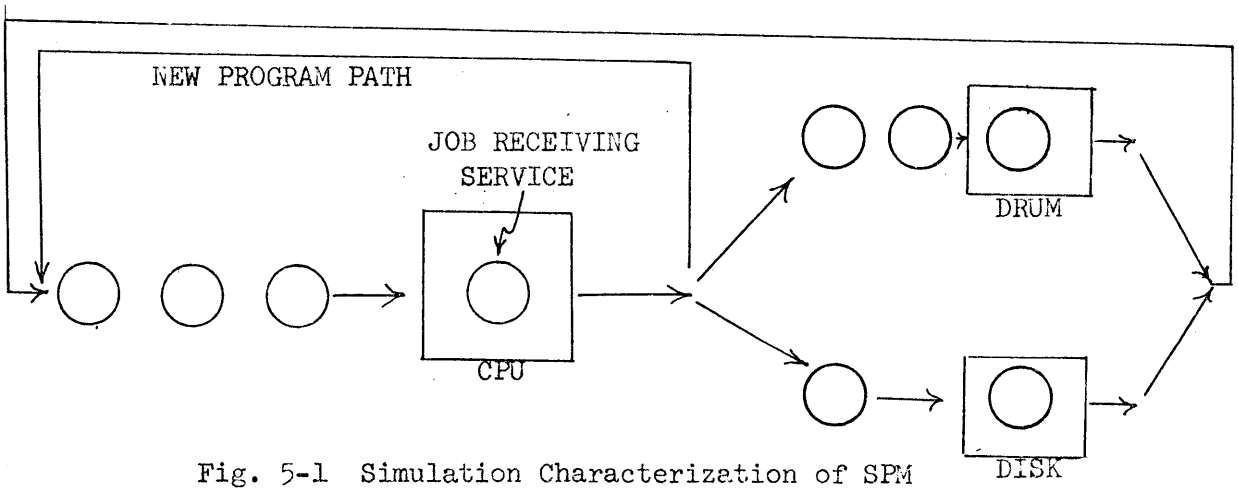


Fig. 5-1 Simulation Characterization of SPM

Service times are all exponentially distributed. Processes requesting use of the I/O disk queue in a FIFO manner, with mean service time of 50 msec. Processes receiving drum service are assumed to be requesting single pages on demand and also queue in a FIFO manner with mean service time depending on the drum (25,10.94 , or 5.00 msec). Processes

requesting use of the CPU also queue for service in a FIFO manner, with mean service time dependent on the degree of multiprogramming (remember, the mean time between requests for peripheral devices is dependent on the mean page fault time which is an explicit function of N).

5.3.2 Approach Taken In Building the Model

This discussion is intended to explain the block diagram that is presented in the next section. Those problems inherent in simulation modeling that were discussed in section (5.2) will be addressed in the development.

The selection of important system parameters was based on the desire to make the modeling effort both economical and comparable to the analytical work reported. The simulation parameters for path selection and service distributions are the same as those derived in the queuing network model (section (4.4)). This is acceptable since the simulation explicitly represents those tokens described in that analytical network.

In order to represent the activity of multiprogramming among the fixed memory partitions, the tokens can now be viewed as GPSS transactions. The model that they flow through is constructed as a tree of blocks (the blocks represent executable subroutines that perform the specific function described by the block name). These transactions enter the system through a generate block that is designed to establish the degree of multiprogramming, N. They then compete for CPU execution which is simulated by a QUEUE-SEIZE-DEPART-ADVANCE-RELEASE sequence.

This permits waiting line statistics to be gathered, representation of the capture of the central resource, and execution of CPU cycles by the capturing process.

A new path for the process is then determined on the basis of a statistical selection. The probabilities p0, p1, and p2 represent selection of the new program path, the drum path, and the disk path. The I/O facility and the paging facility are also represented by QUEUE-SEIZE-DEPART-ADVANCE-RELEASE blocks for the same purpose. Jobs that terminate are marked for total system residence time and are returned to the CPU ready queue as new jobs to compete for new central processing. As in the queuing network, the return path keeps the level of multiprogramming constant.

In building this model, it was not necessary to externally resolve any conflicts arising from the simultaneity of events. To understand how the GPSS processor was able to handle this situation, an explanation of simultaneity and how it is approached in GPSS is necessary.

Basically two chains are maintained by the processor for scheduling activities, the current events chain (CEC) and the future events chain (FEC). The CEC has placed on it those events that are to occur at the most recently updated value of the GPSS clock. The FEC has placed on it those events that are to occur at some future value of the clock.

The GPSS clock maintains virtual time for the simulation and is referenced and modified during the processors clock update phase, at which time events that expect to occur at that clock value are moved from the FEC to the CEC. When the clock update phase is completed, the processor's

scan phase begins. The first transaction that is found on the CEC is moved through as many blocks in it's current path as possible. Some sensitive blocks can restart the scan of the CEC.

Now, if two events are scheduled to occur at the same simulated time, they are both placed on the CEC for event execution. However, it is evident that the two events really occur sequentially in an order that depends on their position on the CEC. The problem that can arise is that events might occur in the wrong sequence, causing unexpected results.

Consider the arrival departure conflict in a teleprocessing system modeled in GPSS. Assume that a teleprocessor request needs a main memory buffer and that it renegs if the buffer isn't available. Assume also, that dispatched messages free a memory buffer. Now, picture the system at simulated time 88 with the entire memory full. If at this time a dispatch event happens to be moved onto the CEC following a request for buffer space, the request would be turned away and then the buffer would be freed. In such a situation, one might prefer to adopt a policy that caused all dispatches to occur before all arrivals in the event of simultaneity.

In the system to be studied here, such a problem does not occur with arrival-departure events (this is the only simultaneous event that might be expected to cause problems). To see this, consider this system at simulated time 88. Assume that a disk I/O processing interval is scheduled to complete (note, this will cause an arrival for CPU scheduling) as well as a CPU processing interval. Also assume that the CPU queue is empty. If the disk completion event is placed ahead of the CPU event on

the CEC, the processor effects termination of the event by moving it through the RELEASE-TRANSFER-ASSIGN-CPU QUEUE blocks. Finding the CPU occupied, it stops movement of that transaction. Since a release block was executed, the processor restarts its scan of the CEC to find the CPU termination event ready to move. Consequently, it moves that transaction through the RELEASE-TEST- and the next appropriate blocks until movement is no longer possible, i.e. a block is found that cannot be entered. Since a release block was executed again, the processor picks up on the disk transaction now in the CPU ready queue. It moves it through the SEIZE-DEPART-ADVANCE sequence at which point it is scheduled as a CPU processing completion for some future clock value. The scan phase is then completed and the next clock update phase is entered.

Thus, it can be seen that a zero CPU queue passage occurred (the disk I/O transaction experienced no waiting time for the CPU) as expected for this situation. Independent of the order in which the events occurred on the CEC, the statistics would not be biased in any way. The GPSS processor is designed to resolve such conflicts without any external intervention.

To return now to the discussion of model development, the problem of attaining equilibrium will be discussed. It has been implied that a model should be run for a long enough time to let statistics reach a steady state. One problem with this approach is that such runs become prohibitively expensive. It is also an open question as to how long a model must be run to swamp out biasing initial conditions.

Included in the repertoire of GPSS instructions is the reset card which facilitates achieving the goal of equilibrium. It has been used in

conjunction with the start card as can be seen in the extended program listing of Appendix C. The effect of this pair of statements is as follows. After the input phase and initial start up of the model, the simulation runs until a timer interrupt occurs (the implicit time unit is .01 msec, runs are for 20 min). Upon occurrence of this event, the processor reads the reset card. The effect of this card is to set all statistics back to zero, yet leaving all chains in tact, i.e. the state of the system is unaffected. Then the processor reads the next card which is a start card, and the model in run until another timer interrupt occurs.

The results of successive applications of these commands clearly demonstrates that equilibrium is attained after two twenty minute runs with resetting and one final twenty minute run. The statistic used as a barometer was CPU utilization as shown in figure 5-2.

RUN TIME	CPU UTILIZ.
20 min.	36.3
—reset—	
20 min.	36.8
—reset—	
40 min.	36.9

Fig. 5-2 CPU Utilization and Steady State Behavior

One cause of the extended length of time needed to achieve equilibrium is the bias of statistics incurred when the CPU queue is loaded with the N customers that initially establish the degree of multiprogramming.

In addition to the above considerations, the time each job is in the system is marked and tabulated to determine the distribution of turn

around time. The maximum contents of the respective queues can be used to specify required memory space for buffering the queues. At the end of each simulation run, the block count at the termination block can be used to determine system throughput, since this count gives the total number of jobs that have completed the system in the given simulated time. Also, execution of three independent program segments in included to determine the distribution of queue lengths. The basic idea being to use the tabulate blocks in weighted mode, entering line lengths weighted by the duration of clock time that they were that length. This results in a specification of the % relative frequency that the line was a specific length.

5.3.3 Table Definitions

GPSS ENTITY		INTERPRETATION
Transactions		
	Segment 1	Computer processes
	Segment 2	A timer
	Segment 3-5	A line line length tabulator
Facilities		
	CPU	execute computer processes
	IOCH	I/O service unit
	BKST	page fault service unit
Queues		
	CPUQ	CPU waiting line
	IOQ	I/O waiting line
	BKSTQ	page waiting line
Tables		
	CPULN	CPU queue line length distribution
	IOQLN	I/O queue line length distribution
	BKQLN	page queue line length distribution
Functions		
	XPDIS	service time distribution
	1-N	path selection probability distrib.

5.3.4 Block Diagram

Figures 5-3 and 5-4 show the flow diagrams for the program segments. Symbolic notation is based on IBM conventions. The extended program list-

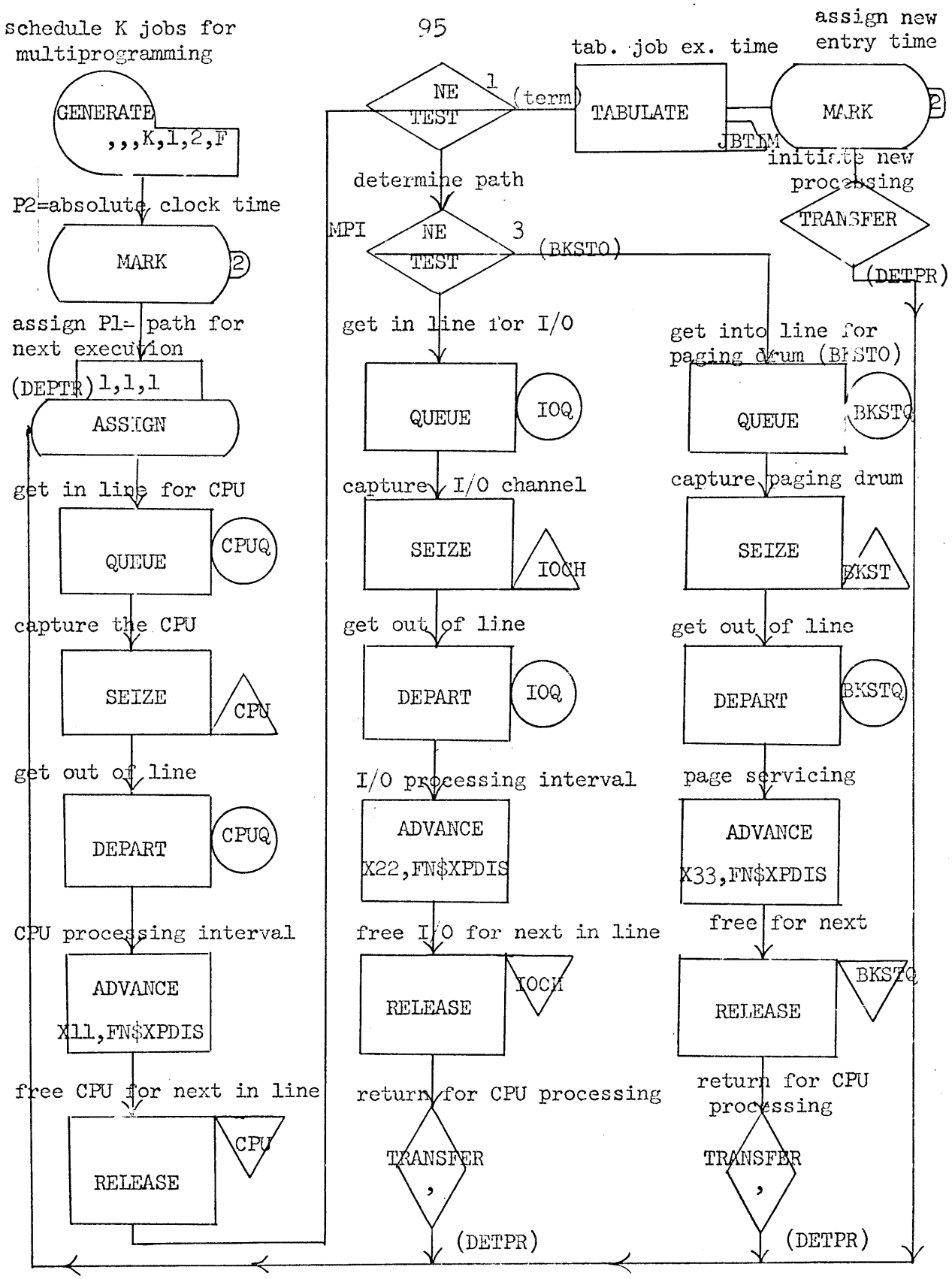


Fig.5-3 Block Diagram for GPSS Simulation

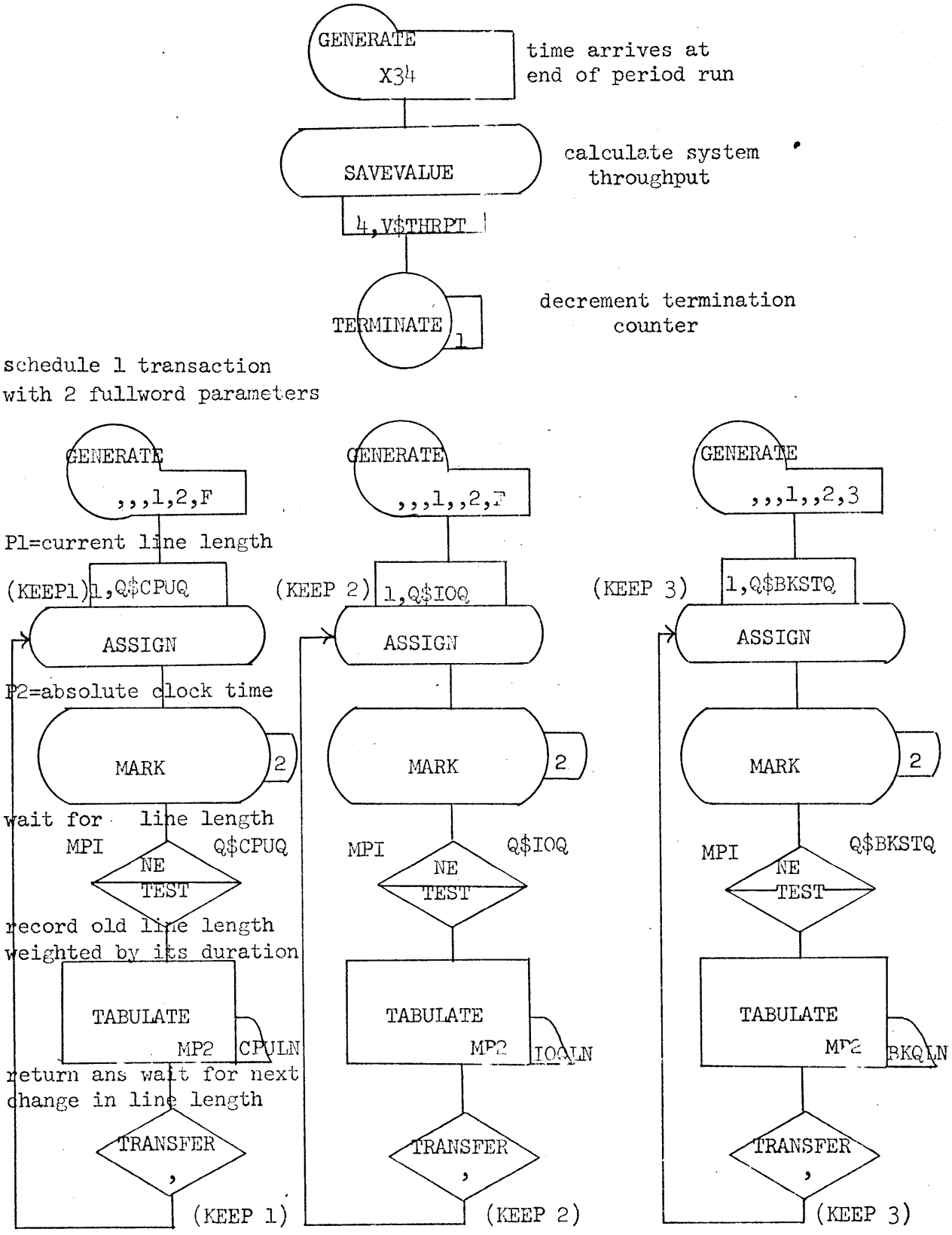


Fig. 5-4 Block Diagrams of Line-Length DistributionTabulators

ing for this program is included in Appendix C.

5.4 Extensions

The analysis performed thus far has mirrored the analytical work reported. In this section the flexibility inherent in the nature of simulation modeling will be discussed by example in order to demonstrate the reason why this technique is used so widely for computer system analysis.

The issue of the level of detail that the SPM represents will be addressed by looking at a more realistic I/O hierarchy and explaining an approach for modeling it with GPSS. Then, it will be shown how a variety of service disciplines can be implemented very easily. Also, introduction of priorities into the model will be demonstrated with a simple example. The concepts of preemption and synchronization of transactions are explained as further examples of techniques available for extending model depth.

5.4.1 Detailed I/O Subsystem

The view of the I/O subsystem taken in this thesis was simplified compared to the actual hierarchies one finds in real systems. In fact, I/O subsystems are so complex that the first attention to scheduling peripheral operating system operations was dedicated to this functional area (W3). A view of a typical module of this sort is shown in figure 5-5. It shows a typical I/O resource structure. The issue of satisfying a particular request becomes a more involved task that requires the aid

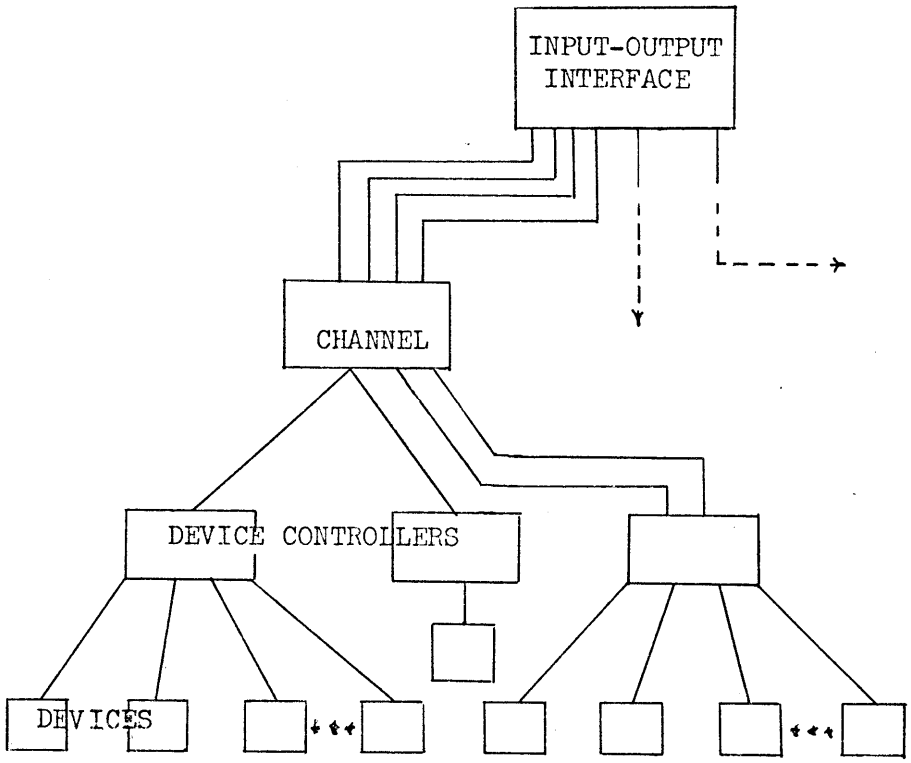


Fig. 5-5 Typical Input/Output Resource Structure

of the device management and information management functions of the operating system. The main system overhead is involved in setting up a path to the lowest level, the actual device. At each level of the hierarchy it is necessary to determine if the next level is available for acquistion. If it is, it can be reserved and the path extended. If it isn't, a decision must be made whether to reserve the path or free it and begin again requesting from the top. This process is demonstrated in figure 5-6.

Once a device has been assigned through execution of component management algorithms above, the information transfer process must be initiated. An example input process is depicted as the array of logical steps in figure 5-7 (W3).

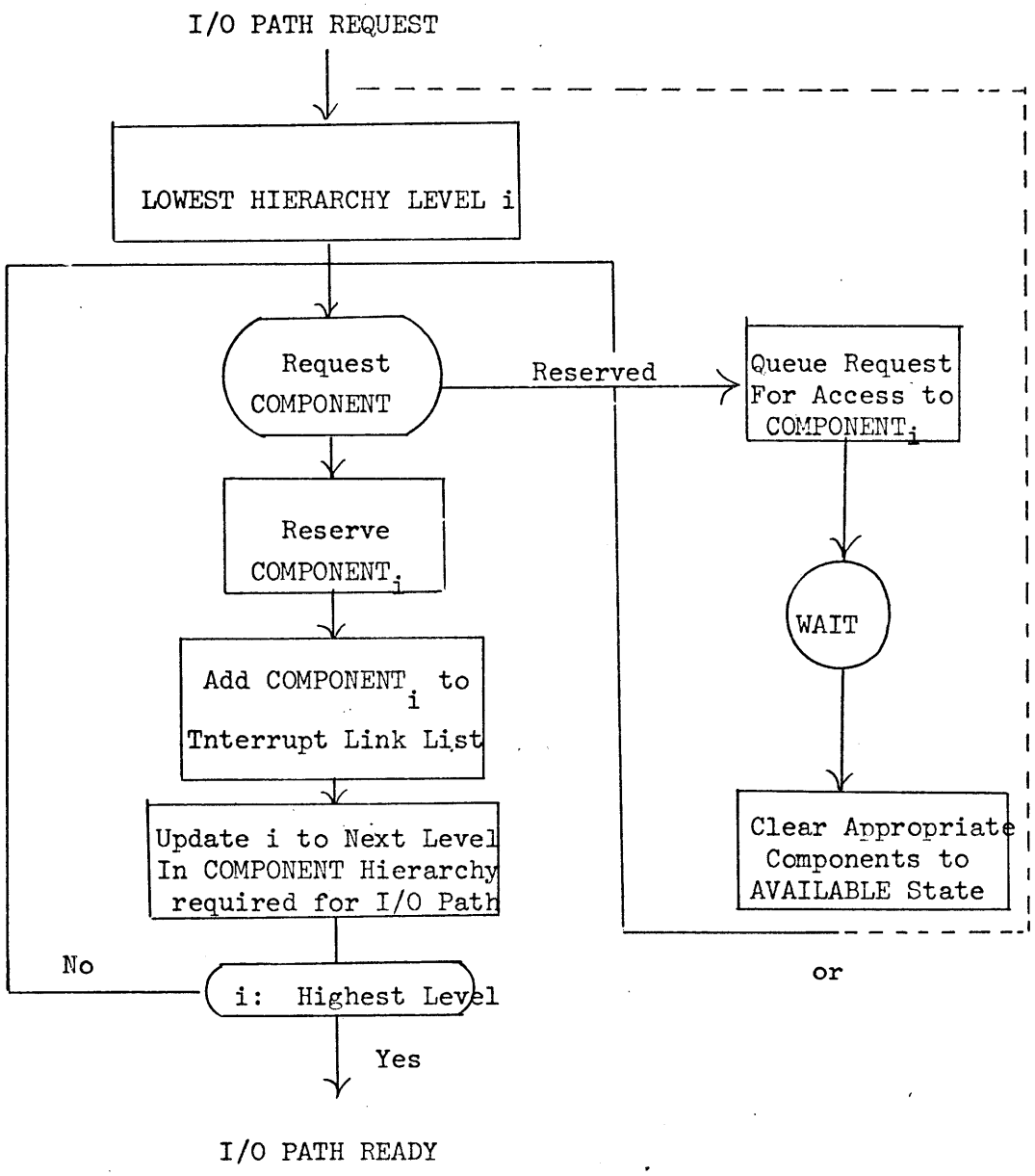


Fig. 5-6 Typical Component Scheduling

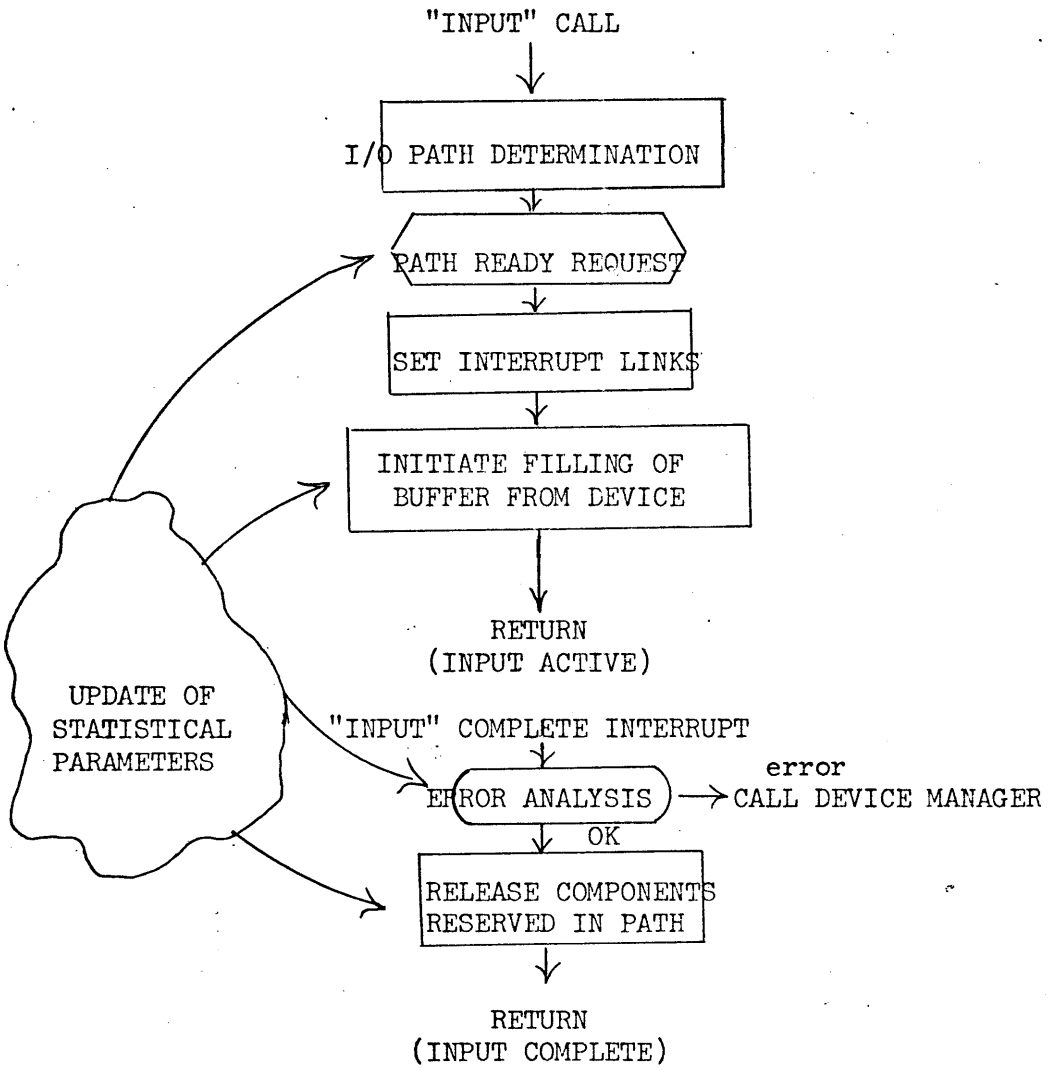


Fig. 5-7 An I/O Hierarchy, General Input Algorithm

If it is felt necessary to include such detail in a system model, simulation provides the facility for this. The computer system simulation language OSSL (W3) has these features included within the structure of the Fortran supported language. In GPSS it would be possible to describe such systems with Boolean Variables which are capable of representing complex logical conditions. In addition, the synchronizing property available from manipulations of the processors special purpose Matching chain and the interrupt facility provided by a similar chain, the Interrupt chain, could be gainfully employed in the modeling effort. The main point is that simulation is the only answer to the explicit representation of such a special purpose subsystem.

5.4.2 Arbitrary Service Disciplines

In the modeling performed, exponential distributions were assumed for all of the service facilities. Inclusion of other non-uniform distributions makes possible modeling of systems under considerably more realistic conditions. So, in GPSS the Function facility is provided to make this all possible. The basic steps involved are defining the non-uniform distribution with appropriate function cards, and providing operand references to these functions in the blocks in which they are needed. There are eight distinct sources of uniform random numbers provided by the GPSS processor that facilitate the sampling of the distributions. These stochastic generators introduce the variability into the modules as described in section (5.2). It is evident, however, that one who is using this modeling technique would certainly be willing to live with this margin of error if the model was sufficiently real to provide useful information

about system performance.

5.4.3 Priority Scheduling

The last area to be discussed is priority scheduling. One easy way to demonstrate this flexibility is with an example. The following block diagram shows a single queuing discipline that serves two process classes identifiable by priority distinction (figure 5-8). SPM is easily extended to this more interesting situation:

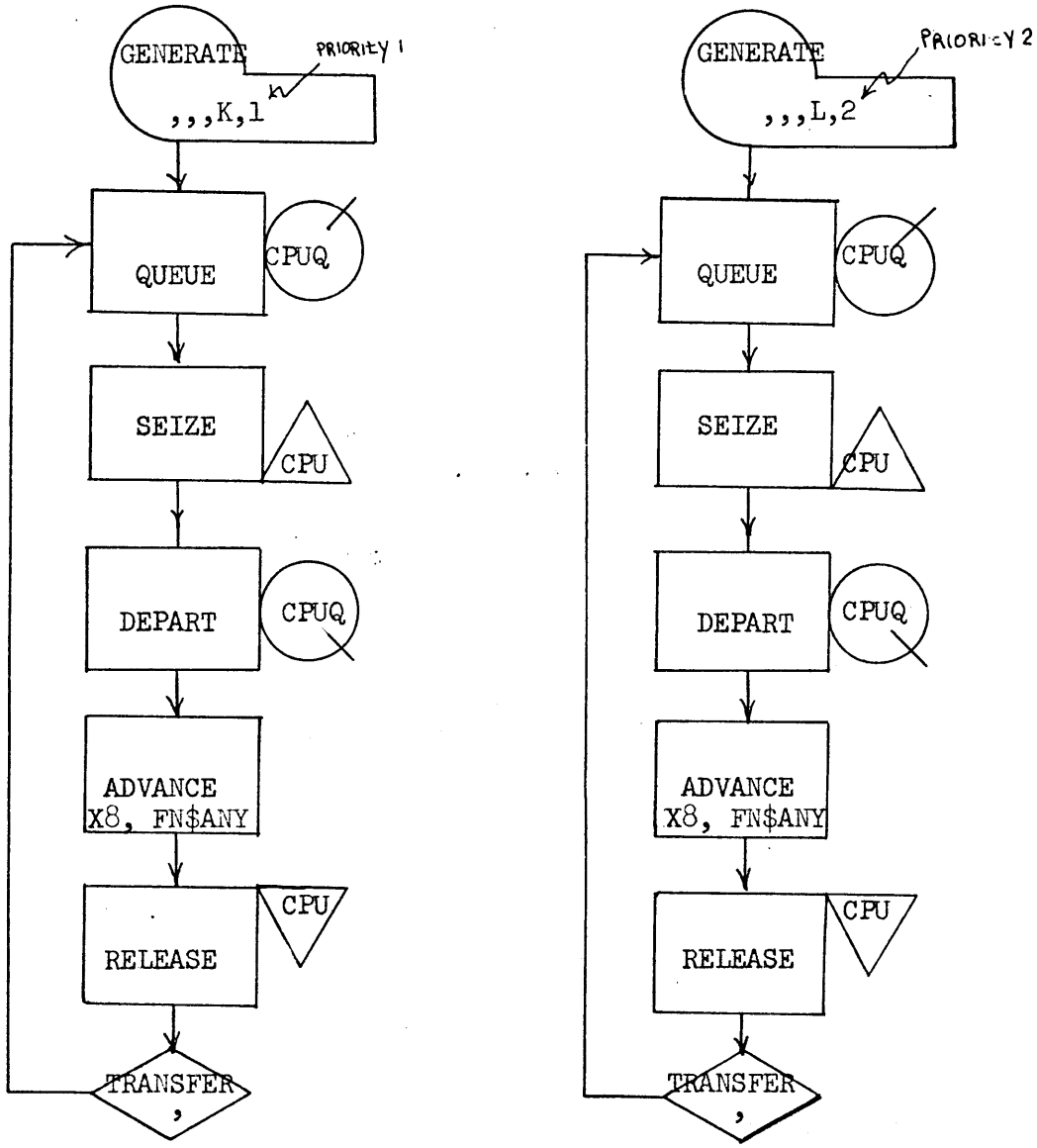


Fig. 5-8 Block Diagram For Priority Process Scheduling, N=L+K

CHAPTER 6

PERFORMANCE PROJECTION: THE RESULTS

6.1 Load and Performance Measures

A model for a CP-67 like multiprogramming module has been developed and analyzed with a variety of modeling techniques. The model is an absolute one in that its intended use is to predict actual performance of the system under study. It's ultimate purpose is to provide information about system operation that can be used in deciding the optimal performance region for a system, given it's characteristics.

In order to perform meaningful analysis with the system model, it is necessary to review the load and performance measures alluded to as valuable for reflecting system behavior.

The first aspect of this discussion is identification of a load measure. Of course, this is none other than the independent variable

of the previous modeling discussions: N, the degree of multiprogramming. This indicator adequately represents the collective demand of processes on the system resources. Often, such a load measure does not represent what the system is really doing, and hence does not reflect the demands placed on the system. Since our job set is a group of statistic-ly identical processes with similar demand needs, this often used indicator is valid for this application.

Performance indicators that merit review and discussion now follow. For example, in section (3.2.1) the composite multiprogramming cost function derived was based on the probability of non-overlapped processor time. Also, in section (3.3.4) it was shown that the average CPU utilization provided information on the effective processing capacity of the system.

One additional indicator, expected queue lengths for shared resources, was pointed out in the applications sections of the models solved. To explain the of this quantity, use is made of Buzen's derivation of server utilization and his definition of system bottleneck (B3,B5). He has shown that the most highly utilized server has the largest expected queue length. He has also defined a bottlenecked resource as one which is seriously degrading system performance. Intuitively, the expected queue length provides a means for recognizing a potentially critical resource (one that is a bottleneck). The longer the steady state queue length, the more utilized the resource, and the greater the possibility that it isn't adequate to permit optimal performance. Doing a sensitivity analysis on this service component by

improving its service rate and determining its effect on system processing capability is a sensible measurement to make. Thus, queue length will be measured in the hopes that it will be effective in monitoring the use of the shared resources.

This latter concept and related measurement points up one of several possible approaches for improving system performance. In fact, for the system under study, it will be shown that the most effective and appropriate technique for effecting improved resource utilization of the CPU is to increase secondary memory speed. This is done through both scheduling memory activities and upgrading the device to one with a faster service rate.

Sumarizing, the information available from the model analysis that will be of value in determining optimal multiprogramming performance and pinpointing system degredation is:

(1) The multiprogramming idle time

(2) The average CPU utilization

(3) Expected queue lengths that form for the shared resources

6.2 Performance Prediction For the Multiprogramming Module

Solving the multiprogramming model has resulted in mapping a class of input functions into a relevent set of output functions. The input functions have been specified in the discussion of the system model in section (3.3). They were found to be process request rates for the system resources (or equivalently, path selection probabilities) and expected device service times. The output functions useful for

analysis have been presented above in section (6.1).

This section will present the results of performance projection for the multiprogramming system. First, the multiprogramming cost accounting technique will be applied to specifing the range of desired operation of the system when it is configured with the medium speed drum. The purpose of the presentation is pedgagocial in nature, the intention being to explicitly identify the individual behavior characteristics as a function of the level of multiprogramming. It is found that the abstract view of the system taken in this analysis does in fact provide results that agree very closely with the dual model that follows.

In the dual model, performance is predicted under a variety of loads with all three drums. The reason for the sensitivity analysis using the drum is that it is this device that is found to be potentially critical with respect to system performance. It is shown how increasing the speed of this device's service mechanism through scheduling and improved device technology can greatly improve system performance. As a result of this projection, a very interesting characteristic of this system is discovered. The law of diminishing returns to scale is found to be in effect for the optimal degree of multiprogramming possible with the configuration under study. Both improvements in the system throughput as achieved by increased device speed, and the optimal degree of N approach a constant.

It is quite reasonable at this point to view the analysis performed by the continuous time Markov state approach, the queuing network approach and the GPSS approach, on the dual conceptual model of SPM, as equivalent.

The rational behind this is provided by a demonstration of the results obtained by these three techniques for the same model configuration and loads. The results agree within .4% with the simulation solutions causing the largest margin of error.

Reviewing the general approach taken in the following sections: A level of multiprogramming is selected by determining the number of pages that are to be permitted in real core/user. The paging behavior that reflects the effect of memory on program activity is determined by picking the number of page faults for the given level of N. The number of I/O's per program is constant. These values determine the appropriate transition probabilities for the models. Performance information is gathered for the level of N selected, and then the procedure is repeated again if desired. It has been found that a maximum level of eight jobs is sufficient to demonstrate system performance.

6.2.1 Multiprogramming Idle Time Analysis

In order to determine the optimal region of multiprogramming operation a lower and upper bound of expected performance were established by selecting two extreme views of secondary memory. For the given job mix and the same CPU processing speed, the input/output hierarchies were assumed to behave according to the two views presented in section (3.3.2). First, requests were assumed to queue in a FCFS manner for the disk and drum. The medium drum (mean service time 10.94 msec) was chosen as representative of channel behavior for these model runs. Then, each request for a peripheral device was assumed to receive instantaneous

service. In this situation, there were no queuing delays, the only delay per request was in access and transfer. The same device service rates were maintained. Tables 6-1,2 tabulate the results obtained. These data are then plotted in figures 6-1 and 6-2. Remember, each type of request is assumed to occur independently of the other in its own operating environment, hence the determination of individual costs for each activity. Sumarizing these results below in table 6-3, we have:

Service Discipline	N_{OPT}	$N_{THRASHING}$
FCFS Queuing	2	3
No Queuing	3	5

Table 6-3 Summary of Result From Cost Accounting Analysis

No attempt was made to obtain any other performance information with this model. Such analysis is left to the more realistic dual channel view of SPM which follows shortly. Let it suffice to observe that as the level of multiprogramming increases, we do in fact experience a decrease in the cost function as the benefits of multiprogramming are enjoyed. Then, there is a sharp increase based primarily on the cost of paging user jobs, as we get too much of a good thing! The excessive competition for primary memory resulted in system paging levels that could not be supported by the available hardware. It is also noted that scheduling the bottlenecked resource, the paging drum, would result in

N	T_{CPU}	$T_{I/O}$	T_{PAGE}	T_{TOTAL}
1	2.00	3.00	.44	5.44
2	2.00	2.37	.38	4.75
3	2.00	2.08	2.15	6.23
4	2.00	1.92	4.77	8.69
5	2.00	1.83	11.14	14.97
6	2.00	1.77	23.90	27.67
7	2.00	1.73	39.28	43.01
8	2.00	1.71	54.63	58.34

Table 6-1 Single Facility Markov Model Results For the Cost of Multiprogramming (FCFS Queuing-Medium Drum)

N	T_{CPU}	$T_{I/O}$	T_{PAGE}	T_{TOTAL}
1	2.00	3.00	.44	5.44
2	2.00	1.55	.22	3.75
3	2.00	.67	.71	3.48
4	2.00	.24	1.32	3.56
5	2.00	.07	4.61	6.68
6	2.00	.02	14.52	16.54
7	2.00	.00	27.63	29.63
8	2.00	.00	40.87	42.87

Table 6-2 Single Facility Markov Model Results For the Cost of Multiprogramming (No Queuing-Medium Drum)

Fig.6-1

COST OF MULTIPROGRAMMING IN NON-OVERLAPPED PROCESSOR TIME: FCFS QUEUING

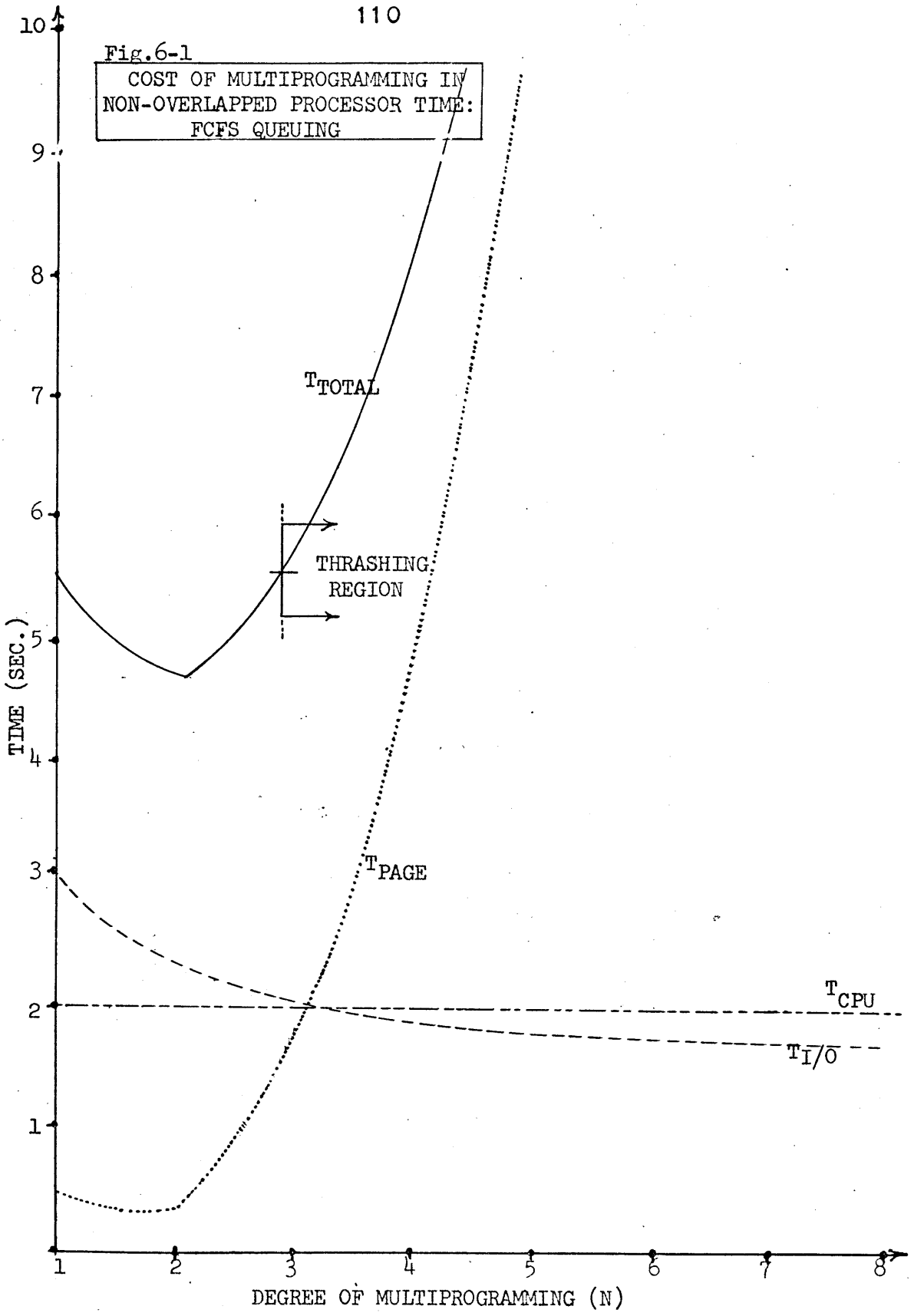
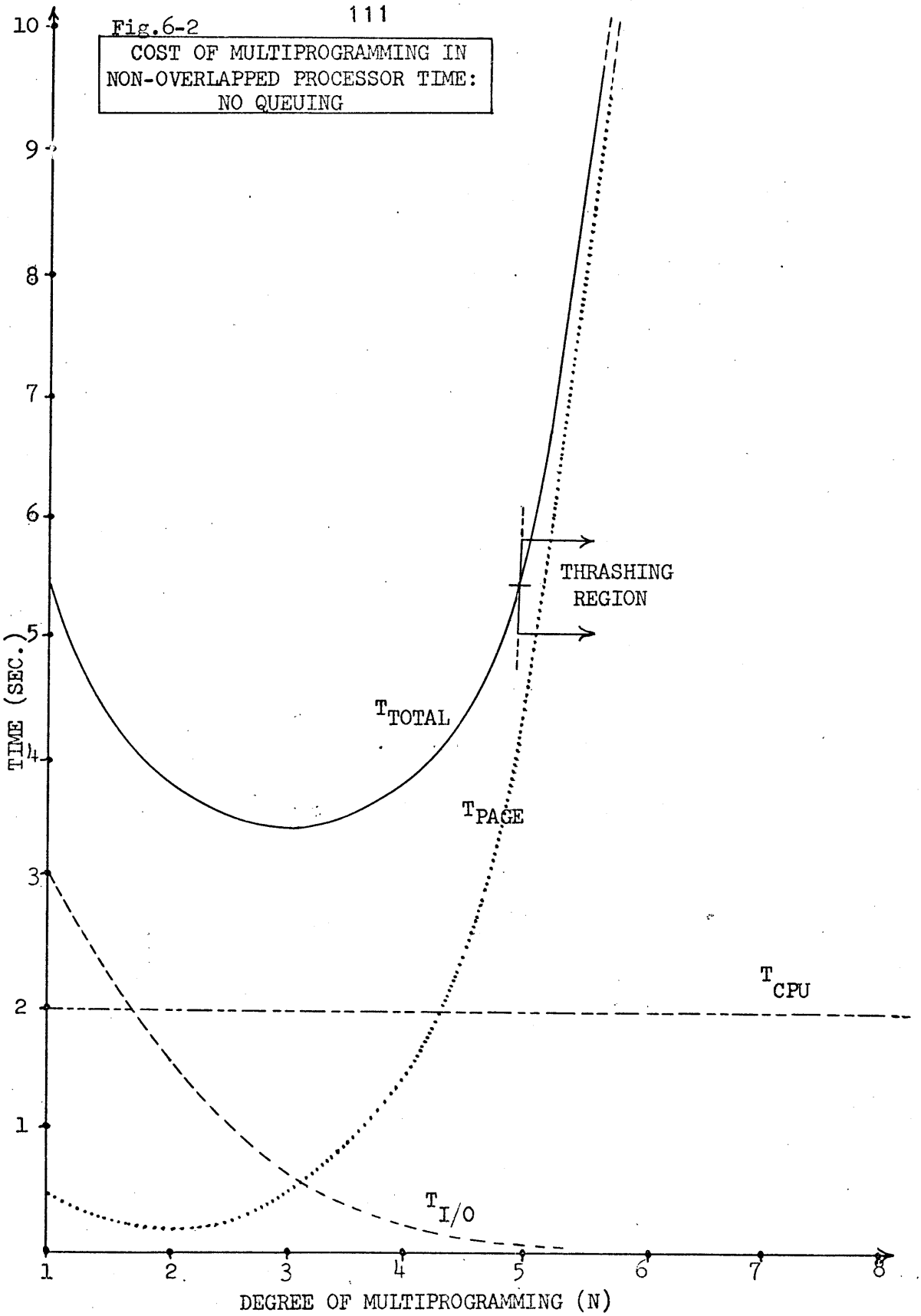


Fig.6-2

COST OF MULTIPROGRAMMING IN NON-OVERLAPPED PROCESSOR TIME: NO QUEUING



performance close to N_{OPT}. This claim will be justified in subsequent analyses. A guesstimate of CPU utilization for such operation is made as follows:

$$\text{CPU Utilization } (N_{OPT}) = \frac{T_{CPU}}{T_{CPU} + T_{I/O} + T_{PAGE}} = \frac{2.00}{3.48} * 100 = 57\%$$

It is at this point when throughput will be greatest by dint of having as many user jobs in the system as possible benefiting from CPU service. A side effect expected is a stretch out in execution time resulting in longer turn around time per job entry.

6.2.2 Average CPU Utilization Analysis

Our performance projection now proceeds with analysis of SPM in its entirety. That is to say, analysis is performed on a single module with both I/O and paging activities occuring naturally within. For the given system configuration, similar views of secondary memory are taken in order to establish a bounds on the degree of multiprogramming.

The tabulated results that follow demonstrate system performance for a variety of situations. The first (table 6-4, fig.6-3) are for the slow drum. The FCFS data shows extremely poor performance with maximum CPU utilization of 41.74 % obtained at N=2. Since the paging drum was clearly limiting system performance, scheduling this device was carried out in the hopes of improving performance. However, even for the optimal case, the benefits of multiprogramming were not being taken well advantage of. No increase in the optimal level of N was achieved through

FCFS

N	% UTILIZ	THRPT Jobs/min.	QUEUES		
			C	I	P
1	33.33	10.00	.33	.50	.17
2	41.74	12.52	.54	.89	.58
3	26.21	7.86	.33	.57	2.10
4	15.91	4.77	.19	.31	3.50
5	7.62	2.29	.08	.13	4.79
6	3.64	1.09	.04	.06	5.90
7	2.22	.67	.02	.04	6.94
8	1.60	.48	.02	.03	7.96

SCHEDULING PAGE CHANNEL

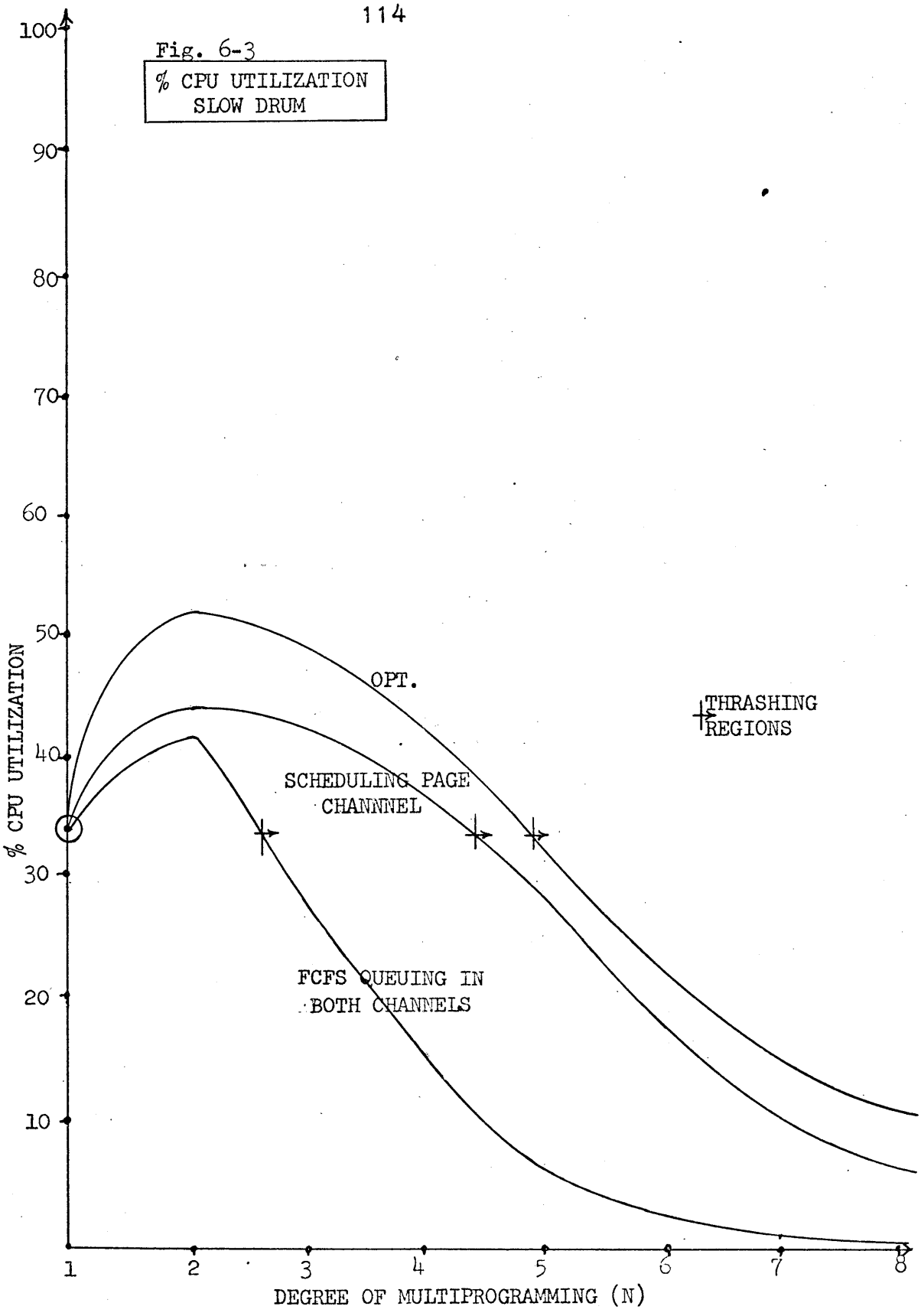
N	% UTILIZ	THRPT Jobs/min.	QUEUES		
			C	I	P
1	33.33	10.00	.33	.50	.17
2	44.70	13.40	.57	.95	.48
3	41.57	12.47	.56	.98	1.46
4	39.54	11.86	.55	.98	2.47
5	30.00	9.00	.39	.67	3.94
6	17.61	5.28	.21	.34	5.45
7	11.09	3.33	.12	.20	6.68
8	8.00	2.40	.09	.14	7.78

NO QUEUING-OPT

N	% UTILIZ	THRPT Jobs/min.	QUEUES		
			C	I	P
1	33.33	10.00	.33	.50	.17
2	52.04	15.61	.67	.78	.55
3	47.03	14.11	.65	.71	1.65
4	43.63	13.09	.62	.66	2.73
5	31.38	9.41	.41	.47	4.12
6	19.87	5.96	.24	.30	5.50
7	14.69	4.41	.17	.22	6.61
8	12.29	3.69	.14	.19	7.68

Table 6-4 Dual Facility SPM Results (Slow Drum)

Fig. 6-3


% CPU UTILIZATION
SLOW DRUM
100
90
80
70
60
50
40
30
20
10
% CPU UTILIZATION
OPT.
THRASHING
REGIONS
SCHEDULING PAGE
CHANNNEL
FCFS QUEUING IN
BOTH CHANNELS
1
2
3
4
5
6
7
8
DEGREE OF MULTIPROGRAMMING (N)

scheduling the page cahnnel. It was found that this technique yielded only a 2.96 increase in CPU utilization over the FCFS case. In addition, the performance curve for the slow drum is quite sharp, indicating that thrashing occurs at a level of 4 jobs. The peaked nature of the curve and the small range of reasonable system performance, (2-3) jobs, is clearly undesirable in an environment where the level of N might fluctuate around the optimal point.

Moving to a better drum was the next step taken in an effort to improve CPU processing capacity as well as achieve a higher level of multiprogramming. These results are tabulated in table 6-5 and are plotted in figure 6-4. More encouraging results were obtained as the upgraded device (medium speed drum, T=16.70 msec) not only reduced effective page queue lengths but also improved the level of multiprogramming to 4. Scheduling resulted in an increase of 4.52% in CPU utilization for the same drum, and it resulted in an increase of over 8% in CPU utilization compared to the slow drum in the same situation. This latter increase resulted in an increase of 20% in overall system throughput. The smoothing out of the performance curve is also a desirable feature gained by moving to the medium speed drum. In this case of scheduling the paging drum, it is evidenced by the thrashing point being moved out to N=6.

The next logical question is whether or not upgrading the hardware further can afford sufficient gains to make such an investment of interest. Results obtained for this situation are based on the use of the fast drum discussed in section (3.2.1). These data are tabulated in

FCFS

N	% UTILIZ.	THRPT Jobs/min.	QUEUES		
			C	I	P
1	36.78	11.03	.37	.55	.08
2	48.38	14.51	.65	1.09	.26
3	43.95	13.19	.64	1.16	1.20
4	33.86	10.16	.48	.88	2.64
5	17.38	5.21	.21	.35	4.44
6	8.31	2.49	.09	.14	5.77
7	5.08	1.52	.05	.08	6.86
8	3.66	1.10	.04	.06	7.90

SCHEDULING PAGE CHANNEL

N	% UTILIZ.	THRPT	QUEUES		
			C	I	P
1	36.78	11.03	.37	.55	.08
2	49.23	14.77	.66	1.11	.23
3	51.51	15.45	.78	1.43	.79
4	52.90	15.87	.87	1.69	1.45
5	48.08	14.42	.76	1.49	2.76
6	35.62	10.67	.50	.93	4.57
7	24.77	7.43	.33	.56	6.11
8	18.23	5.47	.22	.37	7.40

NO QUEUING-OPT

N	% UTILIZ.	THRPT	QUEUES		
			C	I	P
1	36.78	11.03	.37	.55	.08
2	60.57	18.17	.81	.91	.28
3	65.00	19.50	1.03	.98	1.00
4	66.71	20.01	1.78	1.00	1.83
5	56.12	16.84	.94	.84	3.22
6	40.05	12.01	.58	.60	4.12
7	31.07	9.32	.42	.47	6.12
8	26.55	7.96	.34	.40	7.26

Table 6-5 Dual Facility SPM Results (Medium Drum)

Fig. 6-4
% CPU UTILIZATION
MEDIUM DRUM

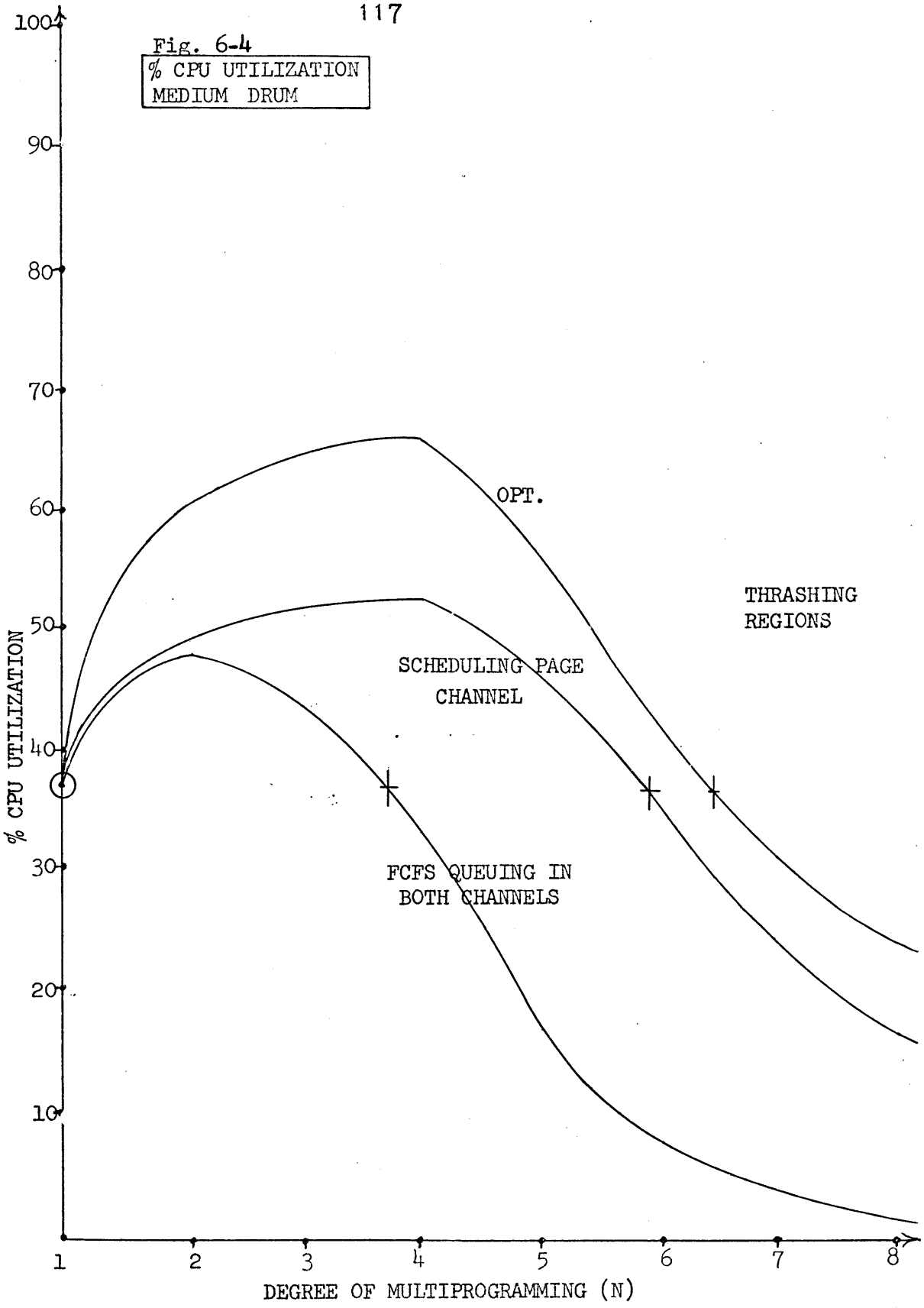


table 6-6 and plotted in figure 6-5. It is noticed that even for CPT, the best level of N is still 4. For the scheduling situation, the maximum gain in CPU utilization over the same situation for the medium speed drum was 5.52% and the increase in throughput from 15.87(jobs/min) to 17.53(jobs/min) yielded only a 10% increase in CPU processing power. A benefit gained was the further smoothing of the performance curve with thrashing occuring at N=7, in this scheduling situation.

The gains achieved for this latter upgrading of support hardware were not as great as those obtained in the former. The system is begining to operate in the region of diminishing returns to scale, as the benefits gained from improved page service are leveling off. Since adequate performance characteristics were achieved with the medium speed drum, the latter investment is not advisable.

To further demonstrate the law of diminishing returns that occur for the optimal level of N (as well as for system throughput), one further study was made with a drum of T=4 msec. The overall results, plotted below in figure 6-6, show that the paging drum has yielded it's share in improving system performance.

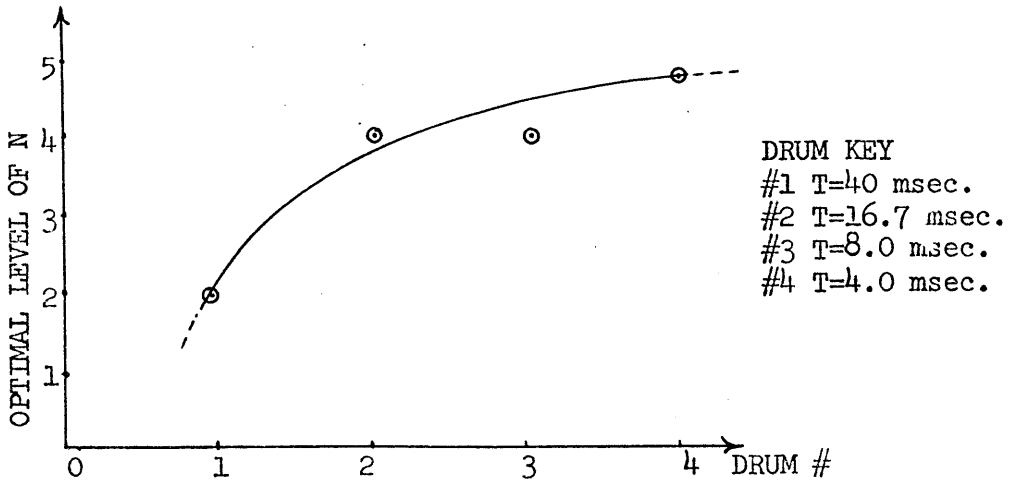


Fig. 6-6 System Performance Under the Law of Diminishing Returns

FCFS

N	% UTILIZ	THRPT Jobs/min.	QUEUES		
			C	I	P
1	38.46	11.54	.39	.58	.04
2	50.93	15.28	.70	1.19	.12
3	53.70	16.11	.86	1.62	.52
4	52.30	15.69	.90	1.80	1.30
5	36.20	1o.89	.55	1.05	3.40
6	18.17	5.45	.22	.37	5.40
7	11.11	3.33	.13	.20	6.67
8	8.00	2.40	.09	.14	7.78

SCHEDULING PAGE CHANNEL

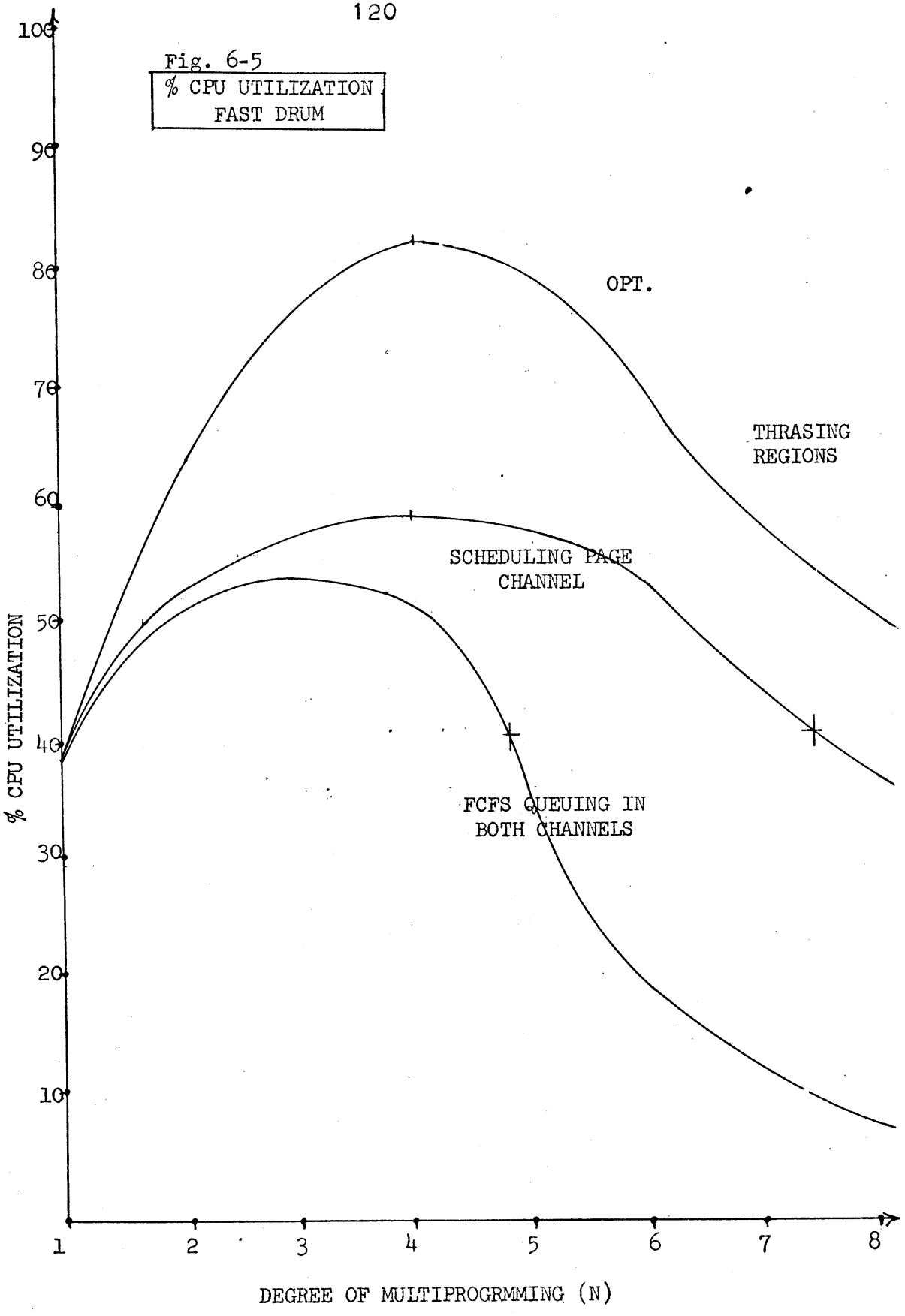
N	% UTILIZ	THRPT	QUEUES		
			C	I	P
1	38.46	11.54	.39	.58	.04
2	51.14	15.34	.70	1.19	.11
3	55.65	16.70	.90	.70	.39
4	58.42	17.53	1.06	2.20	.73
5	58.24	17.47	1.10	2.38	1.53
6	53.70	16.11	.96	2.04	3.00
7	46.23	13.87	.77	1.58	4.65
8	38.052	11.42	.59	1.17	6.24

NO QUEUING-OPT

N	% UTILIZ	THRPT	QUEUES		
			C	I	P
1	38.46	11.54	.39	.58	.04
2	64.91	19.47	.89	.97	.14
3	76.00	22.80	1.33	1.14	.53
4	82.13	24.66	1.74	1.23	1.03
5	78.98	23.69	1.74	1.19	2.07
6	66.87	20.06	1.32	1.00	3.68
7	56.93	17.08	1.02	.85	5.12
8	50.95	15.29	.87	.77	6.37

Table 6-6 Dual Facility SPM Results (Fast Drum)

Fig. 6-5
% CPU UTILIZATION
FAST DRUM



6.3 Discussion of Results

The following table summarizes the results obtained from the above analysis (table 6-7):

APPROACH TAKEN IN MODEL	DRUM DEVICE	QUEUING MECHANISM	N(OPT.)	N(THRASH)	%CPU UTIL. OPT. N
INDEPENDENT CTMPs	MEDIUM SPEED	FCFS	~2	3	32.1
INDEP.CTMP.	MEDIUM	OPT.	~3	5	57.5
DUAL SPM	SLOW	FCFS	2	3	41.74
DUAL SPM	SLOW	SCHEDULE	2	5	44.70
DUAL SPM	SLOW	OPTIMAL	2	5	47.03
DUAL SPM	MEDIUM	FCFS	2	4	48.38
DUAL SPM	MEDIUM	SCHEDULE	4	6	52.90
DUAL SPM	MEDIUM	OPTIMAL	4	6	66.71
DUAL SPM	FAST	FCFS	3	5	53.70
DUAL SPM	FAST	SCHEDULE	4	8	58.42
DUAL SPM	FAST	OPTIMAL	4	10	82.13

Table 6-7 Summary of Results of Performance Projection

One observation is that the cost function analysis was able to provide a useful indication of the expected bounds of multiprogramming performance. This justifies its use in a pedgagocial setting, where demonstration of the effect of the different components of the composite function would be instructive.

The difference in the predicted bounds and the lower % CPU utilization (57% vs 67.7% ,T=10.94,OPT.) can be explained most readily by considering the multiprogramming idle time in the systems. When the activities of paging and I/O were incorporated into the same operating system, it was

possible to overlap the operations of the processor for these activities, where as in the separate facilities, it was not. This is the mair reason for the observed discrepencies.

Concerning the degredation of system performance, it is important to remember the type of page removal algorithm that the data represents. The static parachor curve was obtained as an approximation to the paging behavior in a system with a global LRU page removal policy. For this data, it was found that multiprogramming was limited to 4 jobs even with the fast drum, because competition for primary memory caused system paging to increase at a rate beyond system capabilities. The most severe bottleneck was shown to occur in the paging channel where processes encountered excessive queuing delays for service. Hence, the loss of the benefits of multiprogramming and the quantification of the thrashing point.

System changes such as implementation of a working set policy for page removal have been employed to decrease excessive system paging as the level of multiprogramming increases, i.e. as programs operate with smaller real core allotments. Also, as has been demonstrated above, an increase in the relative speed of drum access time to main memory operations has proved successful in implementing efficient demand paging virtual memory systems.

Thus, it has been shown how a model for a multiprogramming operation can be used to study performance for given load conditions in both a teaching environment as well as a real decision making situation.

Determination of the optimal level of multiprogramming and specification of the thrashing region of system performance were based on sample program parameters characteristic of an assembler run under CP-67 in a "squeezed core" environment. The analysis was facilitated by such system indicators as multiprogramming idle time, average CPU utilization, and queue lengths forming for shared resources.

Other areas where application of such techniques can be used to provide performance information are based on the means for determining the parachor curve from which input function data was generated. For example:

(1) Using Bards model for average resident set size(B1), one can determine a dynamic parachor curve. Using this paging characteristic as an input function would yield more true to life performance projections for the actual system.

(2) System performance might be projected for programs run with various page sizes. The queuing model can then be used to isolate optimal performance for this new varying parameter.

(3) In addition, quantative statements can be advanced on the behavior of systems under different page removal algorithms. Parachor curve measurements must be performed to generate the necessary inputs here as well.

6.4 Comparison of the Results for the Alternative Modeling Techniques

In this section it will be shown that the analytical and simulation models chosen for analysis were all adequate for performing the above analysis. The reason for studying the system with three different techniques was to lend insight into the dppropriateness of each approach for computer modeling. This aspect of the effort will be discussed more fully in the next chapter when the techniques are compared on a number of

interesting, revealing points.

The three models compared in the following table are:

(1) the dual facility continuous time Markov model

(2) the queuing network model

(3) the GPSS simulation

These results are based on varying the level of N as done in the preceeding analysis. The data selected for display is from the FCFS queuing discipline and the medium speed drum (T=16.97). Table 6-8 is shown below:

N	Markov-state	Que. Ntwk	GPSS Sim.
1	36.781	36.792	36.9
2	48.379	48.380	48.5
3	43.950	43.956	43.8
4	38.855	33.860	33.9
5	17.376	17.376	17.4
6	8.310	8.310	8.2
7	5.079	5.078	5.1
8	3.657	3.656	3.5

Table 6-8 Summary of Results of the Alternative Modeling Techniques

Thus application of any one of the methods to the solution of SPM would have provided us with similar predictive information. It is safe to say that we have validated the predictive accuracy of the models.

As indicated above, the next chapter provides a comparison of the modeling approaches. It is hoped that this will aid the modeler in selecting the most appropriate modeling tool for his given application.

CHAPTER 7

CONCLUSIONS

7.1 A Comparison of the Modeling Techniques

The analysis of SPM with both analytical and simulation models has resulted in similar performance projections for the CP-67 like mulitprogramming module (sect. (6-3)). Yet, the means to achieve these ends were quite different. It would be of value, now, to assess these means in order to determine the technique that was best suited for the problem posed in this thesis. In addition, an effort will be made to shed light on the application of these models to problems further reaching in nature than those studied here.

The comparison will be based on the following criteria considered important in the development and implementation of system models:

(1) the level of detail describable

(2) the number of parameters needed to describe the model

(3) the ease of development of the model for implementation

(4) the ease of solution and the exactness of the solution

(5) the overall cost of the approach

Lets begin then, with the level of detail describable. In terms of the models studied here, all of the modeling approaches were capable of representing the input functions and mapping them into the output functions essential for the analysis performed. However, on the basis of the extensions discussed for GPSS, simulation is clearly the most flexible of the three approaches. Selection between the continuous time Markov model state approach and the queuing network continuous time Markov model is difficult. The level of detail that they can be used to describe is tied into the view of the system that they choose to take. For example, Sekino(S7) modeled a general purpose time sharing system with the intention of including all factors that contribute to overall system performance. He was able to represent multiple processors, data base lockout, and various system overheads by building on a hierarchical set of models much like the single facility continuous time Markov state models used here. Moore(M7), on the other hand, modeled a general purpose time sharing system paying more attention to intricate job routing in such systems. His approach was to model the system as a network of queues similar to the queuing network model used here. Clearly, both analysists set out to study similar systems, both men arrived at valid models of their systems operation, yet both described different details of the systems behavior. One can see that both methods have met with notable success.

Concerning the number of parameters needed to specify the model, the

queuing network model was most reasonable. Since the GPSS simulation was based on the network interpretation of multiprogramming, deriving its parameters from that analysis, it certainly fits into the same category. Not only were fewer parameters required in these implementations, but they were more readily understandable as well. The obvious reason for this was the manner in which the closed form solution used explicit probabilities to describe movement among queues that represented physical characteristics of the real system. The single and dual continuous time Markov processes, instead, chose to explicitly represent all system states; and transitions in the system consequently, require multiple use of the program and device behavior parameters. Note, the closed form solution of the queuing network model obviated this view of the system.

In terms of ease of model development for implementation, the network approach was again superior. This is not only based on its parameterization but also on the closed form solution developed by Buzen and the associated computational algorithm he derived. The simulation model was facilitated by the General Purpose System Simulator and implementation was supported by the debugging facilities in the language. The most involved technique was the Markov state approach. Representation of the differential difference equations as a general NxN matrix for solution by MLSQ was tedious.

The next to last category involves a discussion of the exactness of the solutions. Both of the analytical models were solved with available algorithms. The closed form solution proposed by Buzen resulted in an exact solution; however, a numerical analysis technique used to solve the

equations economically resulted in introducing marginal errors into the results. The approximate solution for the differential difference equations agreed within .005% . Both of these approaches were clearly superior to the simulation analysis with its stochastic generators and running approach to achieving equilibrium. These results agreed within .4% .

In the last category, cost, simulation exceeded the other two methods by a factor of 10. The network of queues solution was less costly than the solution of the N simultaneous equations because of the neat computational algorithm. In general, simulation is the most expensive of the modeling approaches. When more complex systems are under study, such analysis can become prohibitively expensive very quickly if extreme care is not taken in the model development phase. The trade off is that mentioned in section (5.1) of economy versus detail.

So, in conclusion, the queuing network continuous time Markov model was superior and was the most appropriate for the study undertaken here. This is not surprising, for the conceptual framework of the Central Server Model on which its development is based is much like SPM (B3). It is suggested, as a final comment on these models, that reasoning out the issues above is an important first step before any model selection is made. An important factor in this regard, as pointed out by the look at the work of Sekino and Moore, is the ultimate view to be taken of the system.

7.2 The Scope of SPM: Future Work

The system process model has been gainfully employed in analyzing

the performance of a virtual memory system that multiprograms under a demand paging strategy. The advantage gained in applying this model was to transcend the use of human intuition in a problem characterized by complex interdependencies. Yet, in studying the multiprogramming module, certain limitations in the use of the analytical models to solve SPM became more apparent. These are now presented in the hope that future work in this area is addressed to some of these issues:

(1) The composite activity of N real processes in the system were represented by N virtual, statistically identical jobs. It is clear that more work must be done in the area of including job classes in the model of the system.

(2) Priorities are not associated with any of the jobs that were represented in SPM. It certainly would be desirable to model this relevent system characteristic.

(3) System overload was assumed. This aspect of the model has been overcome in the literature (R2,B3) but was not included in this model. Practical application of this model feature to realistic system behavior should be studied further.

(4) Model parameters cannot vary with time. This limitation is one that is basic to the representation.

(5) Operating system overhead is not included in the processing activities of the CPU. The behavior parameters of the model should be studied further to gain more insight into this important factor.

(6) Peripheral queues represent devices that cannot be dedicated, i.e. a process cannot occupy two marker positions at one time. This limitation is cited as basic to the nature of the representation.

(7) Effort should be made to include more realistic distributions for the server mechanisms as done in Moore's work (M7).

Further effort in resolving these limitations will result in more resourceful applications of such models in computer analysis. To date, the work done in the area of SPM has yielded valuable predictive results

in a number of situations and has contributed to insight into the operations not only of multiprogramming modules, but of the more complex time-sharing systems in use today.

7.3 Final Comments

It has been the intention of this thesis to present a model that is useful for performance projection of computers managing multiprogramming under a page on demand virtual memory strategy. The limitations of the model that have been cited are receiving attention. The exclusion of system overhead from this analysis represents a restriction that has yielded performance results that are slightly better than might really be expected.

Furthermore, it was intended to show how parameterization and analysis of a complex computer system can provide insight into the potential problems and interdependencies of such systems.

In addition, the result of the multi-pronged modeling effort performed was a comparative analysis of the techniques currently in use in the area of performance evaluation. This review is of value to anyone who is thinking about selecting one of these tools for system studies. Another result of this approach was to demonstrate the predictive capacity of these models.

The final comment will be addressed to the general nature of computer system models. All to often, models have been developed that are too specific to be of value in answering questions about new or different systems. Rather than having to develop a new model for each

system studied, stress should be placed on developing classes of general purpose models capable of representing a variety of important behavior characteristics and system configurations. In the light of the rapidly changing technology attention should also be paid to future systems being developed. The emphasis on developing new systems should go hand in hand with research on modeling techniques capable of accurate representation of these systems.

APPENDIX A: The Continuous Time Markov-State Analysis Program

The following extended program listing performs the solution of the differential difference equations, in the steady state, for both the single and combined Markov interpretations of SPM.

The input parameters and output parameters are defined on the sample outputs that follow the listing.

For reasons based on the lettering available for symbol definition, the following choises were necessary:

P for π
L for λ

```
ANALYZE:    PROCEDURE OPTIONS(MAIN);
            DECLARE (NO_RHS,M) FIXED BINARY INITIAL(1),
                    (DISP,CNT,N1,N2,K)FIXED BINARY INITIAL(0),
                      STOP,NP1 ,NOC FIXED BINARY;
            DECLARE (NP,N,CONFIG_CODE,TYPE) FIXED BINARY(15,0);
            DECLARE (NO_PAGE,NO_IO, NO_PARAL,N1DUM, N2DUM,
                    NO_IOS,NO_PF) FIXED BINARY(15,0);
            DECLARE(LP,LI,UP,UI,L,U) FLOAT BINARY;
            DECLARE (TIME,THROUGH_PUT,MEAN_EXEC_TIME) FLOAT BINARY;
            DECLARE (EQIO,EQCPU,EQPAGE) FLOAT BINARY;
            DECLARE (MLP,MLI,MUP,MUI,ML,MU) FLOAT BINARY;
            DECLARE (TIO,TPAGE) FLOAT BINARY;
            DECLARE (EPF,EPFT) FLOAT BINARY INITIAL(20);
            DECLARE (CPU_IDLE_TIME,SUM_PI) FLOAT BINARY INITIAL(0);
            DECLARE CPU_BUSY FLOAT BINARY;
            DECLARE (PI(K,K),RHS(K,1)) BINARY FLOAT (53) CTL;
            DECLARE (QSUM_IO(N), QSUM_PAGE(N), QSUM_CPU(N))
                    FLOAT BINARY CTL;


            ON CONDITION (EOF) GO TO DONE;
GETDAT:     GET LIST(N,NP,MUP,MUI,MU,NO_PF,CONFIG_CODE,TYPE,
                 NO_PAGE,NO_IO,NO_PARAL,NO_IOS,MEAN_EXEC_TIME,NQ);
            K=0;
            IF CONFIG_CODE=2  THEN DO;
                                     DO I=N+1 TO 1 BY -1;
                                     K=I+K;
                                     END;
                                   MLI=MEAN_EXEC_TIME/NO_IOS;
                                   MLP=MEAN_EXEC_TIME/NO_PF;
                                   LP=1/MLP; UP=1/MUP;
                                   LI=1/MLI; UI=1/MUI;
                                   GO TO ALLOC;
                                   END;
            K=N+1;
            IF TYPE=1 THEN ML=MEAN_EXEC_TIME/NO_IOS;
```

```
                        ELSE ML=MEAN_EXEC_TIME/NO_PF;
              L=1/ML; U=1/ML;
ALLOC:        ALLOCATE PI,RHS;
              IF CONFIG_CODE=2  THEN DO;
                                     CALL INITIAL;
                                     CALL DUAL;
                                     END;
                                ELSE DO;
                                     CALL INITIAL;
                                     CALL SINGLE;
                                     END;

/*    ENTER CONSTRAINT RELATIONSHIP INTO PI                                  */
              DO J=1 TO K;
              PI(K,J)=1;
              END;

/*    SET UP RIGHT HAND SIDE MATRIX(RHS) FOR MLSQ SUBROUTINE                 */
              DO I=1 TO K-1;
              RHS(I,1)=0;
              END;
              RHS(K,1)=1;

/*  PRINT OUT SYSTEM VARIABLES                                               */
PUT PAGE;
PUT SKIP(6);
PUT SKIP EDIT('SYSTEM CONFIGURATION IS ',CONFIG_CODE)
            (COLUMN(48), A(23), F(2,0));
              IF TYPE=1 THEN DO;
                             PUT SKIP EDIT('I/O MODEL')(COLUMN(56),
                                      A(9));
                             GO TO OUTPT;
                             END;
```

```
                IF TYPE=2 THEN DO;
                              PUT SKIP EDIT('PAGE MODEL')(COLUMN(56),
                                    A(10));
                              GO TO OUTPT;
                              END;
                PUT SKIP EDIT('COMBINED PAGE AND I/O MODEL')(COLUMN(47)
                             ,A(27));
OUTPT:
                              IF NQ=0 THEN DO;
                              PUT SKIP EDIT('NO QUEUING')(COLUMN(56),
                                    A(10));
                                    GO TO CONT8;
                                    END;
                              IF NQ=1 THEN DO;
                              PUT SKIP EDIT('FCFS QUEUING')(COLUMN(55)
                                    ,A(12));
                                    GO TO CONT8;
                                    END;
                              PUT SKIP EDIT('SCHEDULING PAGE CHANNEL')
                                    (COLUMN(51),A(23));
CONT8:
/*  USE MATRIX SUBROUTINE MLSQ TO SOLVE PI*X=RHS                        */
                CALL MLSQ(PI,RHS,K,K,NO_RHS);

/*  CALCULATE THE CONSTRAINT RELATION AS A CHECK=>SUM_PI                */
                SUM_PI=0;
                DO I=1 TO K;
                SUM_PI=SUM_PI + RHS(I,1);
                END;

/*  CALCULATE FRACTION OF CPU IDLE TIME IN THE STEADY STATE             */
                CPU_IDLE_TIME=0;
                N1=0; N2=0;
                DO I=1 TO K;
```

```
                IF N1+N2=N THEN DO;
                                CPU_IDLE_TIME=CPU_IDLE_TIME+RHS(I,1);
                                N1=0;
                                N2=N2+1;
                                GO TO CONT5;
                                END;
                N1=N1+1;
CONT5:          END;

                CPU_BUSY=1-CPU_IDLE_TIME;


/*  CALCULATE THE SYSTEM THROUGHPUT IN JOBS/MIN                          */
                THROUGH_PUT=CPU_BUSY*60*1000/MEAN_EXEC_TIME;



PUT SKIP(2) EDIT('NUMBER OF PROCESSORS=', NP)
                 (COLUMN(16),A(21),F(4,0));
PUT SKIP EDIT('DEGREE OF MULTIPROGRAMMING(N)=',N)
                (COLUMN(16),A(31),F(4,0));
PUT SKIP EDIT('NUMBER OF SYSTEM EQUATIONS(K)=',K)
                (COLUMN(16),A(30),F(4,0));
IF CONFIG_CODE=2 THEN DO;
PUT SKIP(2) EDIT('EXPECTED TIME BETWEEN PAGE FAULTS(MSEC)=',MLP)
                 (COLUMN(16),A(40),F(8,5));
PUT SKIP EDIT('PAGE FAULT RATE(LP,MSEC)=',LP)
                (COLUMN(16),A(26),F(7,5));
PUT SKIP(2) EDIT('EXPECTED TIME BETWEEN I/O REQUESTS(MSEC)=',MLI)
                 (COLUMN(16),A(41),F(8,5));
PUT SKIP EDIT('I/O REQUEST RATE(LI,MSEC)=',LI)
                (COLUMN(16),A(26),F(7,5));
PUT SKIP(2) EDIT('EXPECTED TIME TO SERVICE A PF(MSEC)=',MUP)
                 (COLUMN(16),A(37),F(8,5));
PUT SKIP EDIT('PAGE SERVICE RATE(UP,MSEC)=',UP)
                (COLUMN(16),A(27),F(7,5));
```

```
PUT SKIP(2) EDIT('EXPECTED TIME TO SERVICE AN I/O(MSEC)=',MUI)
                (COLUMN(16),A(38),F(8,5));
PUT SKIP EDIT('I/O SERVICE RATE(UI,MSEC)=',UI)
              (COLUMN(16),A(27),F(7,5));
                        END;

                    ELSE DO;
PUT SKIP(2) EDIT('EXPECTED TIME BETWEEN I/O''S(ML,MSEC)=',ML)
                (COLUMN(16),A(38),F(8,5));
PUT SKIP EDIT('REQUEST RATE(L,MSEC)=',L)
              (COLUMN(16),A(21),F(7,5));
PUT SKIP(2) EDIT('EXPECTED TIME BETWEEN SERVICES(MU,MSEC)=',MU)
                (COLUMN(16),A(40),F(8,5));
PUT SKIP EDIT('SERVICE RATE(U,MSEC)=',U)
              (COLUMN(16),A(21),F(7,5));
                        END;
```



```
PUT SKIP(2) EDIT('CONSTRAINT RELATION SUM_PI YIELDS SUM_PI=',SUM_PI)
                (COLUMN(16),A(43),F(7,5));
PUT SKIP EDIT('STEADY STATE FRACTION OF CPU IDLE TIME=',CPU_IDLE_TIME)
              (COLUMN(16),A(39),F(7,5));
PUT SKIP EDIT('STEADY STATE FRACTION OF CPU BUSY TIME=',CPU_BUSY)
              (COLUMN(16),A(39),F(7,5));

                IF TYPE=1 THEN DO;
                                TIO=5*RHS(K,1);
                     PUT SKIP(2) EDIT('TIO(MSEC)=',TIO)
                                     (COLUMN(16),A(10),F(8,5));
                                END;
                IF TYPE=2 THEN DO;
                                TIME=NO_PF*MU+2000;
                                TPAGE=TIME*RHS(K,1);
                 PUT SKIP(2) EDIT('TPAGE(MSEC)=',TPAGE)
```

```
                                    (COLUMN(16),A(12),F(12,5));
                                END;

                IF CONFIG_CODE=1 THEN GO TO FOLLOW;
/* INITIALIZE SUMS FOR COMPUTING EXPECTED QUEUE LENGTHS                    */
                ALLOCATE QSUM_IO;
                ALLOCATE QSUM_PAGE;
                ALLOCATE QSUM_CPU;
                DO I=0 TO N;
                QSUM_CPU(I)=0;
                QSUM_PAGE(I)=0;
                QSUM_IO(I)=0;
                END;

/*   CALCULATE SUMS FOR ESTABLISHING EXPECTED QUEUE LENGTHS             */

                N1=0; N2=0;
                DO I=1 TO K;
                NOC=N-(N1+N2);
                QSUM_IO(N1)=QSUM_IO(N1)+RHS(I,1);
                QSUM_PAGE(N2)=QSUM_PAGE(N2)+RHS(I,1);
                QSUM_CPU(NOC)=QSUM_CPU(NOC)+RHS(I,1);
                IF N1+N2=N THEN DO;
                                N1=0;
                                N2=N2+1;
                                GO TO CONT6;
                                END;
                N1=N1+1;
CONT6:          END;

/*   CALCULATE AND PRINT EXPECTED QUEUE LENGTHS                         */

                EQCPU=0;  EQPAGE=0;  EQIO=0;

                DO N1=0 TO N;
                EQIO=EQIO+N1*QSUM_IO(N1);
```

```
                END;

                DO N2=0 TO N;
                EQPAGE=EQPAGE+N2*QSUM_PAGE(N2);
                END;

                DO NOC=0 TO N;
                EQCPU=EQCPU+NOC*QSUM_CPU(NOC);
                END;

PUT SKIP(2) EDIT('EXPECTED CPU QUEUE LENGTH=',EQCPU)
                (COLUMN(16),A(26),F(5,3));
PUT SKIP EDIT('EXPECTED IO QUEUE LENGTH=',EQIO)
              (COLUMN(16),A(25),F(5,3));
PUT SKIP EDIT('EXPECTED PAGE QUEUE LENGTH=',EQPAGE)
              (COLUMN(16),A(27),F(5,3));
FOLLOW:

PUT SKIP(2) EDIT('STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=',
                THROUGH_PUT)(COLUMN(16),A(41),F(8,5));

/*  PRINT OUT RESULTS OF SOLUTION OF SYSTEM OF EQUATIONS                */
                PUT SKIP(2) EDIT('STEADY STATE PROBABILITIES')
                                (COLUMN(16),A(27));
                N1=0; N2=0;
                DO I=1 TO K;
                PUT SKIP EDIT('STEADY STATE (',N1,N2,')=',RHS(I,1))
                      (COLUMN(16),A(14),F(2,0),F(2,0),A(2),F(7,5));
                IF N1+N2=N THEN DO;
                                N1=0;
                                N2=N2+1;
                                GO TO CONT7;
                                END;
                N1=N1+1;
CONT7:          END;
                FREE PI,RHS;
```

```
                FREE QSUM_IO;
                FREE QSUM_PAGE;
                FREE QSUM_CPU;
                GO TO GETDAT;
/*  INITIALIZE PI MATRIX FOR MLSQ SUBROUTINE                              */
  INITIAL:      PROCEDURE;
                DO I=1 TO K;
                   DO J=1 TO K;
                   PI(I,J)=0;
                   END;
                END;
                END INITIAL;


/*   NOW MAKE THE APPROPRIATE ENTRIES IN THE STRUCTURED PI MATRIX    */


/*  CALCULATE ALL OF THE DIAGONAL ENTRIES FOR THE DUAL SYSTEM         */
  DUAL:         PROCEDURE;
                N1=0;  N2=0;
                NP1=NP;
                DO I=1 TO K-1;
                N1DUM=N1; N2DUM=N2;
                IF N1>=NO_IO THEN N1DUM=NO_IO;
                IF N2>=NO_PAGE THEN N2DUM=NO_PAGE;
                IF N1+N2=N THEN DO;
                                PI(1,I)=-((N1DUM)*UI+(N2DUM)*UP);
                                N1=0;
                                N2=N2+1;
                                NP1=NP;
                                GO TO CONT1;
                                END;
                           ELSE DO KNP=1 TO NP;
                                IF N1+N2=N-NP+KNP THEN DO;
                                                       NP1=NP-KNP;
                                                       GO TO ENT1;
```

```
                                                        END;
                                 END;
ENT1:            PI(I,1)=-((N1DUM)*UI+(N2DUM)*UP+(NP1*(LI+LP)));
                 N1=N1+1;
CONT1:           END;
/*   CALCULATE AND ENTER UPPER AND LOWER DIAGONAL ENTRIES                */
                 N1=0; N2=0; NP1=NP;
                 STOP=N+1;
                 DO I=2 TO K-1;
                 N1=N1+1;
                 N1DUM=N1;
                 IF N1>=NO_IO THEN N1DUM=NO_IO;
                 IF N1=STOP THEN DO;
                                 NP1=NP;
                                 N1=0;
                                 N2=N2+1;
                                 STOP=STOP-1;
                                 GO TO CONT2;
                                 END;
                            ELSE IF NP=1 THEN GO TO ENT2;
                                         ELSE DO KNP=1 TO NP;
                              IF N1+N2=1+N-NP+KNP THEN DO;
                                                       NP1=NP-KNP;
                                                       GO TO ENT2;
                                                       END;
                                 END;
ENT2:            PI(I,I-1)=NP1*LI;
                 PI(I-1,I)=N1DUM*UI;
CONT2:           END;
/*  CALCULATE AND ENTER UPPER -UPPER AND LOWER-LOWER DIAGONAL ENTR       */
                 N1=0; N2=1; NP1=NP; DISP=0;
                 STOP=N+1;
                 DO I=N+2 TO K;
                 N1=N1+1;
                 N2DUM=N2;
                 IF N2>=NO_PAGE THEN N2DUM=NO_PAGE;
```

```
                IF N1=STOP THEN DO;
                                STOP=STOP-1;
                                DISP=DISP+1;
                                NP1=NP;
                                IF NP+N2=N+1 THEN NP1=NP1-1;
                                N1=1;
                                N2=N2+1;
                                N2DUM=N2;
                                IF N2>=NO_PAGE THEN N2DUM=NO_PAGE;
                                GO TO ENT3;
                                END;
                           ELSE DO KNP=1 TO NP;
                                IF N1+N2=2+N-NP+KNP THEN DO;
                                                         NP1=NP-KNP;
                                                         GO TO ENT3;
                                                         END;
                                END;
ENT3:           PI(I,I-(N+1)+DISP)=NP1*LP;
                PI(I-(N+1)+DISP,I)=N2DUM*UP;
CONT3:          END;
                END DUAL;


  SINGLE:       PROCEDURE;
/*  ENTER THE DIAGONAL ENTRIES FOR THE SINGLE SYSTEM                    */
                N1=0; NP1=NP;
                DO I=1 TO K-1;
                   N1DUM=N1;
                   IF N1>=NO_PARAL THEN N1DUM=NO_PARAL;
                   IF NP=1 THEN GO TO ENT4;
                   DO KNP=1 TO NP;
                   IF N1=N-NP+KNP THEN DO;
                                       NP1=NP-KNP;
                                       GO TO ENT4;
                                       END;
                   END;
```

```
ENT4:           PI(I,I)=-((N1DUM)*U+(NP1)*L);
                N1=N1+1;
                END;
/*      ENTER UPPER AND LOWER DIAGONAL ENTRIES FOR THE SINGLE SYSTEM */
                N1=0; NP1=NP;
                DO I=2 TO K;
                   IF N1>=NO_PARAL THEN N1DUM=NO_PARAL;
                                   ELSE N1DUM=I-1;
                   DO KNP=1 TO NP;
                   IF N1=N-NP+KNP THEN DO;
                                       NP1=NP-KNP;
                                       GO TO ENT5;
                                       END;
                   END;
ENT5:           PI(I,I-1)=NP1*L;
                PI(I-1,I)=N1DUM*U;
                N1=N1+1;
                END;
                END SINGLE;

DONE:           END ANALYZE;
```

SYSTEM CONFIGURATION IS 1
I/O MODEL
NO QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 5
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN I/O'S(ML,MSEC)= 33.33333
REQUEST RATE(L,MSEC)=0.03000

EXPECTED TIME BETWEEN SERVICES(MU,MSEC)=50.00000
SERVICE RATE(U,MSEC)=0.02000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.01418
STEADY STATE FRACTION OF CPU BUSY TIME=0.98582

TIO(MSEC)= 0.07092

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=29.57449

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.22413
STEADY STATE (1 0)=0.33619
STEADY STATE (2 0)=0.25214
STEADY STATE (3 0)=0.12607
STEADY STATE (4 0)=0.04728
STEADY STATE (5 0)=0.01418

SYSTEM CONFIGURATION IS 1
I/O MODEL
FCFS QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 5
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN I/O'S(ML,MSEC)= 33.33333
REQUEST RATE(L,MSEC)=0.03000

EXPECTED TIME BETWEEN SERVICES(MU,MSEC)=50.00000
SERVICE RATE(U,MSEC)=0.02000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.36541
STEADY STATE FRACTION OF CPU BUSY TIME=0.63459

TIO(MSEC)= 1.82703

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=19.03778

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.04812
STEADY STATE (1 0)=0.07218
STEADY STATE (2 0)=0.10827
STEADY STATE (3 0)=0.16241
STEADY STATE (4 0)=0.24361
STEADY STATE (5 0)=0.36541

SYSTEM CONFIGURATION IS 1
PAGE MODEL
NO QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 5
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN I/O'S(ML,MSEC)= 1.90476
REQUEST RATE(L,MSEC)=0.52500

EXPECTED TIME BETWEEN SERVICES(MU,MSEC)=10.94000
SERVICE RATE(U,MSEC)=0.09141

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.34211
STEADY STATE FRACTION OF CPU BUSY TIME=0.65789

TPAGE(MSEC)= 4614.09766

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=19.73654

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.00657
STEADY STATE (1 0)=0.03773
STEADY STATE (2 0)=0.10834
STEADY STATE (3 0)=0.20742
STEADY STATE (4 0)=0.29783
STEADY STATE (5 0)=0.34211

SYSTEM CONFIGURATION IS 1
PAGE MODEL
FCFS QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 5
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN I/O'S(ML,MSEC)= 1.90476
REQUEST RATE(L,MSEC)=0.52500

EXPECTED TIME BETWEEN SERVICES(MU,MSEC)=10.94000
SERVICE RATE(U,MSEC)=0.09141

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.82591
STEADY STATE FRACTION OF CPU BUSY TIME=0.17409

TPAGE(MSEC)= 11139.07813

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)= 5.22262

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.00013
STEADY STATE (1 0)=0.00076
STEADY STATE (2 0)=0.00436
STEADY STATE (3 0)=0.02504
STEADY STATE (4 0)=0.14380
STEADY STATE (5 0)=0.82591

SYSTEM CONFIGURATION IS 2
COMBINED PAGE AND I/O MODEL
FCFS QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 2
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN PAGE FAULTS(MSEC)=23.52940
PAGE FAULT RATE(LP,MSEC)= 0.04250

EXPECTED TIME BETWEEN I/O REQUESTS(MSEC)=33.33333
I/O REQUEST RATE(LI,MSEC)=0.03000

EXPECTED TIME TO SERVICE A PF(MSEC)= 10.94000
PAGE SERVICE RATE(UP,MSEC)=0.09141

EXPECTED TIME TO SERVICE AN I/O(MSEC)=50.00000
I/O SERVICE RATE(UI,MSEC)= 0.02000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.51621
STEADY STATE FRACTION OF CPU BUSY TIME=0.48379

EXPECTED CPU QUEUE LENGTH=0.647
EXPECTED IO QUEUE LENGTH=1.093
EXPECTED PAGE QUEUE LENGTH=0.260

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=14.51377

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.16317
STEADY STATE (1 0)=0.24476
STEADY STATE (2 0)=0.36714
STEADY STATE (0 1)=0.07586
STEADY STATE (1 1)=0.11380
STEADY STATE (0 2)=0.03527

SYSTEM CONFIGURATION IS 2
COMBINED PAGE AND I/O MODEL
SCHEDULING PAGE CHANNEL

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 2
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN PAGE FAULTS(MSEC)=23.52940
PAGE FAULT RATE(LP,MSEC)= 0.04250

EXPECTED TIME BETWEEN I/O REQUESTS(MSEC)=33.33333
I/O REQUEST RATE(LI,MSEC)=0.03000

EXPECTED TIME TO SERVICE A PF(MSEC)=10.94000
PAGE SERVICE RATE(UP,MSEC)=0.09141

EXPECTED TIME TO SERVICE AN I/O(MSEC)=50.00000
I/O SERVICE RATE(UI,MSEC)= 0.02000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.50752
STEADY STATE FRACTION OF CPU BUSY TIME=0.49248

EXPECTED CPU QUEUE LENGTH=0.659
EXPECTED IO QUEUE LENGTH=1.112
EXPECTED PAGE QUEUE LENGTH=0.229

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=14.77433

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.16610
STEADY STATE (1 0)=0.24915
STEADY STATE (2 0)=0.37373
STEADY STATE (0 1)=0.07723
STEADY STATE (1 1)=0.11584
STEADY STATE (0 2)=0.01795

SYSTEM CONFIGURATION IS 2
COMBINED PAGE AND I/O MODEL
NO QUEUING

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 2
NUMBER OF SYSTEM EQUATIONS(K)= 6

EXPECTED TIME BETWEEN PAGE FAULTS(MSEC)=23.52940
PAGE FAULT RATE(LP,MSEC)= 0.04250

EXPECTED TIME BETWEEN I/O REQUESTS(MSEC)=33.33333
I/O REQUEST RATE(LI,MSEC)=0.03000

EXPECTED TIME TO SERVICE A PF(MSEC)=10.94000
PAGE SERVICE RATE(UP,MSEC)=0.09141

EXPECTED TIME TO SERVICE AN I/O(MSEC)=50.00000
I/O SERVICE RATE(UI,MSEC)= 0.02000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
STEADY STATE FRACTION OF CPU IDLE TIME=0.39435
STEADY STATE FRACTION OF CPU BUSY TIME=0.60565

EXPECTED CPU QUEUE LENGTH=0.810
EXPECTED IO QUEUE LENGTH=0.908
EXPECTED PAGE QUEUE LENGTH=0.282

STEADY STATE SYSTEM THROUGHPUT(JOBS/MIN)=18.16956

STEADY STATE PROBABILITIES
STEADY STATE (0 0)=0.20427
STEADY STATE (1 0)=0.30641
STEADY STATE (2 0)=0.22981
STEADY STATE (0 1)=0.09497
STEADY STATE (1 1)=0.14246
STEADY STATE (0 2)=0.02208

APPENDIX B: The Queuing Network Analysis Program

The extended program listing that follows solves the queuing network model of SPM. The input and output functions are defined on the sample outputs which follow. The GN normalizing constants which are by-products of the computational algorithm are output along with the performance information. They have been used within the program to generate the key marginal distributions outlined in section (4.4.4).

```
SPM:      PROCEDURE OPTIONS(MAIN);
          DECLARE(N,N_COUNT,CHAN_COUNT,NO_CHANNELS,NO_IO_CHAN,
                NO_PAGE_CHAN,NO_IOS,NO_PAGE_FAULTS) FIXED BINARY(15,0);
          DECLARE (UTILIZ,THROUGH_PUT,MUI,MUP,MCPU,UI,UP,UCPU,
                   DENOM,P0,PI,PP,QSUM,Q1,Q2,Q3) FLOAT BINARY;
          DECLARE (GN(I,J), X(CHAN_COUNT)) BINARY FLOAT CTL;
          DECLARE (Q_DEV2,Q_DEV3) BINARY FLOAT (53);

          ON ENDFILE(SYSIN) GO TO DONE;

GETDAT:   GET LIST(N,NO_IO_CHAN,NO_PAGE_CHAN,MUI,MUP,NO_IOS,
                   NO_PAGE_FAULTS,MEAN_EXEC_TIME);
          UP=1/MUP;   UI=1/MUI;
/*DETERMINE PATH SELECTION PROBABILITIES P0, PI, PP AND UCPU FOR CSM */
          DENOM=NO_IOS+NO_PAGE_FAULTS+1;
          UCPU=(NO_IOS+NO_PAGE_FAULTS+1)/MEAN_EXEC_TIME;
          MCPU=1/UCPU;
          P0=1/DENOM;
          PI=NO_IOS/DENOM;
          PP=NO_PAGE_FAULTS/DENOM;
          SUM_PI=P0+PI+PP;

/*INITIALIZE GN CALCULATING MATRIX                                    */
          CHAN_COUNT=NO_IO_CHAN+NO_PAGE_CHAN+1;
          I=N+1;   J=CHAN_COUNT;
          ALLOCATE GN;
          DO J=1 TO CHAN_COUNT;
          GN(1,J)=1;
          END;
          DO I=1 TO N+1;
          GN(I,1)=1;
          END;

/*CALCULATE THE COLUMN MULTIPLIERS                                    */
          ALLOCATE X;
          X(2)=UCPU*PP/UP;
```

```
          X(3)=UCPU*PI/UI;

/* INITIALIZE FOR SUMS                                                    */

          QSUM=0;
          Q_DEV2=0;
          Q_DEV3=0;

/*CALCULATE GN ENTRIES                                                   */
          DO J=2 TO CHAN_COUNT;
            DO I=2 TO N+1;
            GN(I,J)=GN(I,J-1)+X(J)*GN(I-1,J);
            IF J=CHAN_COUNT & I¬=N+1 THEN QSUM=QSUM+GN(I,J);
            END;
          END;

/*CALCULATE Q SUMS FOR THE OTHER DEVICES                                 */
          DO K=1 TO N-1;
            Q_DEV2=(X(2)**K)*GN(N+1-K,CHAN_COUNT)+Q_DEV2;
          END;

          DO K=1 TO N-1;
            Q_DEV3=(X(3)**K)*GN(N+1-K,CHAN_COUNT)+Q_DEV3;
          END;

/*CALCULATE SYSTEM UTILIZATION FOR CSM                                   */
          UTILIZ=GN(N,CHAN_COUNT)/GN(N+1,CHAN_COUNT);

/*CALCULATE SYSTEM THROUGHPUT                                            */
          THROUGH_PUT=UTILIZ*60*1000/MEAN_EXEC_TIME;

/*CALCULATE EXPECTED QUEUE LENGTHS                                       */
          Q1=QSUM/GN(N+1,CHAN_COUNT);
          Q2=Q_DEV2/GN(N+1,CHAN_COUNT);
          Q3=Q_DEV3/GN(N+1,CHAN_COUNT);
```

```
/*OUTPUT RESULTS                                                    */
 PUT PAGE;
 PUT SKIP(6) EDIT('CENTRAL SERVER MODEL')(COLUMN(53),A(20));
 PUT SKIP EDIT('FCFS QUEUING')(COLUMN(57),A(12));
/*PRINT OUT GN MATRIX                                               */
 PUT SKIP(3);
         DO J=1 TO CHAN_COUNT;
            DO I=1 TO N+1;
            PUT SKIP EDIT('GN(',I,',',J,')=',GN(I,J))
              (COLUMN(16),A(3),F(3,0),A(1),F(3,0),A(2),F(8,5));
            END;
         END;
 PUT SKIP(2) EDIT('NUMBER OF PROCESSORS=',1)
                  (COLUMN(16),A(21),F(4,0));
 PUT SKIP EDIT('DEGREE OF MULTIPROGRAMMING(N)=',N)
               (COLUMN(16),A(31),F(4,0));
 PUT SKIP EDIT('EXPECTED TIME BETWEEN CHANNEL REQUESTS=',MCPU)
               (COLUMN(16),A(39),F(8,5));
 PUT SKIP EDIT('EXPECTED SERVICE RATE OF CPU(MSEC-1)=',UCPU)
              (COLUMN(16),A(37),F(8,5));
 PUT SKIP EDIT('EXPECTED TIME TO SERVICE A PF(MSEC)=',MUP)
               (COLUMN(16),A(36),F(8,5));
 PUT SKIP EDIT('EXPECTED TIME TO SERVICE AN I/O(MSEC)=',MUI)
               (COLUMN(16),A(36),F(8,5));
 PUT SKIP(2) EDIT('CONSTRAINT RELATION SUM_PI YIELDS SUM_PI=',SUM_PI)
                  (COLUMN(16),A(43),F(7,5));
 PUT SKIP EDIT('P0=',P0,'  PI=',PI,'  PP=',PP)(COLUMN(16),
               A(3),F(7,5),A(5),F(7,5),A(5),F(7,5));
 PUT SKIP(2) EDIT('STEADY STATE FRACTION OF CPU BUSY TIME=',UTILIZ)
                  (COLUMN(16),A(39),F(7,5));
 PUT SKIP EDIT('EXPECTED LENGTH OF CPU QUEUE=',Q1)
               (COLUMN(16),A(29),F(8,5));
 PUT SKIP EDIT('EXPECTED LENGTH OF I/O QUEUE=',Q3)
               (COLUMN(16),A(29),F(8,5));
 PUT SKIP EDIT('EXPECTED LENGTH OF PAGE QUEUE=',Q2)
               (COLUMN(16),A(30),F(8,5));
```

```
 PUT SKIP(2) EDIT('SYSTEM THROUGHPUT(JOBS/MIN)=',THROUGH_PUT)
                 (COLUMN(16),A(28),F(7,2));
          FREE X;
          FREE GN;
          GO TO GETDAT;
DONE:     END SPM;
```

CENTRAL SERVER MODEL
FCFS QUEUING

GN(1, 1)= 1.00000
GN(2, 1)= 1.00000
GN(3, 1)= 1.00000
GN(1, 2)= 1.00000
GN(2, 2)= 1.46494
GN(3, 2)= 1.68110
GN(1, 3)= 1.00000
GN(2, 3)= 2.96490
GN(3, 3)= 3.12833

NUMBER OF PROCESSORS= 1
DEGREE OF MULTIPROGRAMMING(N)= 2
EXPECTED TIME BETWEEN CHANNEL REQUESTS=13.69900
EXPECTED SERVICE RATE OF CPU(MSEC-1)= 0.07300
EXPECTED TIME TO SERVICE A PF(MSEC)=10.94000
EXPECTED TIME TO SERVICE AN I/O(MSEC)=50.00000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI= 1.00000
P0=0.00685 PI=0.41096 PP=0.58219

STEADY STATE FRACTION OF CPU BUSY TIME=0.48380
EXPECTED LENGTH OF CPU QUEUE= 0.48380
EXPECTED LENGTH OF I/O QUEUE= 0.72568
EXPECTED LENGTH OF PAGE QUEUE= 0.22494

SYSTEM THROUGHPUT(JOBS/MIN)= 14.51

CENTRAL SERVER MODEL
FCFS QUEUING

```
GN(  1,  1)= 1.00000
GN(  2,  1)= 1.00000
GN(  3,  1)= 1.00000
GN(  4,  1)= 1.00000
GN(  1,  2)= 1.00000
GN(  2,  2)= 2.53134
GN(  3,  2)= 4.87634
GN(  4,  2)= 8.46734
GN(  1,  3)= 1.00000
GN(  2,  3)= 4.03109
GN(  3,  3)=10.52195
GN(  4,  3)=24.84746

NUMBER OF PROCESSORS=   1
DEGREE OF MULTIPROGRAMMING(N)=    3
EXPECTED TIME BETWEEN CHANNEL REQUESTS= 5.86609
EXPECTED SERVICE RATE OF CPU(MSEC-1)= 0.17047
EXPECTED TIME TO SERVICE A PF(MSEC)=10.94000
EXPECTED TIME TO SERVICE AN I/O(MSEC)=50.00000

CONSTRAINT RELATION SUM_PI YIELDS SUM_PI=  1.00000
PO=0.00293  PI=0.17595  PP=0.82111

STEADY STATE FRACTION OF CPU BUSY TIME=0.43956
EXPECTED LENGTH OF CPU QUEUE= 0.60179
EXPECTED LENGTH OF I/O QUEUE= 1.02413
EXPECTED LENGTH OF PAGE QUEUE= 1.05355

SYSTEM THROUGHPUT(JOBS/MIN)=  13.19
```

APPENDIX C: The General Purpose Systems Simulation Program

The following extended program listing is for the GPSS simulation which was used to model SPM. The program is explained by block diagrams in section (5.3.4). The technique that permits use of the same model for varying levels of N as well as varying path selection probabilities is shown of pages 162-164. There, the block redefinition statements which follow the initial start up of the model facilitate the solution of the model for the various loads and parameters.

```
        REALLOCATE STO,0,COM,7000    GIVE MORE MEMORY TO COMMON POOL
*LAB    OPERATION  A,B,C,D,E,F                COMMENTS
*
        SIMULATE
*
*  FUNCTION DEFINITIONS
*
 1      FUNCTION   RN3,D3
.00990,1/.40594,2/1,3
 2      FUNCTION   RN3,D3
.00685,1/.58904,2/1,3
 3      FUNCTION   RN3,D3
.00293,1/.82404,2/1,3
 4      FUNCTION   RN3,D3
.00178,1/.89305,2/1,3
 5      FUNCTION   RN3,D3
.00090,1/.94509,2/1,3
 6      FUNCTION   RN3,D3
.00044,1/.97346,2/1,3
 7      FUNCTION   RN3,D3
.00027,1/.98361,2/1,3
 8      FUNCTION   RN3,D3
.00020,1/.98815,2/1,3
*
 XPDIS FUNCTION   RN2,C24
0.0,0.0/0.1,0.104/0.2,0.222/0.3,0.355/0.4,0.509/0.5,0.69
0.6,0.915/0.7,1.2/0.75,1.38/0.8,1.6/0.84,1.83/0.88,2.12
0.9,2.3/0.92,2.52/0.94,2.81/0.95,2.99/0.96,3.2/0.97,3.5
0.98,3.9/0.99,4.6/0.995,5.3/0.998,6.2/0.999,7.0/0.9997,8.0
*
*
* ELEMENT SAVE VALUES DECLARATIONS/INITIALIZATIONS
        INITIAL    X5,60000000/X11,1981/X22,5000/X33,1094/X34,120000000
*
* TABLE DEFINITIONS
*
```

```
 CPULN TABLE       P1,0,1,W5          TABLE FOR CPU LENGTHS
 IOQLN TABLE       P1,0,1,W5          TABLE FOR IO LENGTHS
 BKQLN TABLE       P1,0,1,W5          TABLE FOR BKST LENGTHS
 JBTIM TABLE       MP2,0,100000,100 TABULATE JOB TIMES TIME UNIT MSEC/100
*
*VARIABLE DEFINITIONS
 THRPT VARIABLE    X5*N$TERM/X34          STEADY STATE SYS THPT JOBS/HOUR
*
*MAIN PROGRAM SEGMENT
 KEY   GENERATE    ,,,1,1,2,F         ESTABLISH THE DEGREE OF MULTIPROGRAM
       MARK        2                  ESTABLISH JOB ENTRY TIME FOR NEW JOBS
 DETPR ASSIGN      1,1,1              PARM1 IS PTH SELECT VEH BASED ON RUN
 CPUIN QUEUE       CPUQ               GATHER Q STAT FOR CPU
       SEIZE       CPU                CPU IS A FACILITY
       DEPART      CPUQ               LEAVE QUEUE
       ADVANCE     X11,FN$XPDIS       ADVANCE FOR CPU PROCESSING
       RELEASE     CPU                FREE CPU FOR OTHER PROGRAM EXECUTION
       TEST NE     P1,1,TERM          IF P1 HAS VALUE 1 GO TO TERM
       TEST NE     P1,2,BKSTO         P1=2 GO TO BACKING STORE
       QUEUE       IOQ                OTHERWISE QUEUE UP IN I/O CHANNEL
       SEIZE       IOCH               GET CHANNEL
       DEPART      IOQ                EXIT STATISTICS ROUTINE
       ADVANCE     X22,FN$XPDIS       I/O SERVICE
       RELEASE     IOCH               DONE WITH CHANNEL AND DEVICE
       TRANSFER    ,DETPR             RETURN FOR CPU PROCESSING INTERVAL
 BKSTO QUEUE       BKSTQ              PAGE SERVICE INITIATED
       SEIZE       BKST               GET PAGING DRUM
       DEPART      BKSTQ              COMPLETE STAT.
       ADVANCE     X33,FN$XPDIS       FIXED RATE OF PAGE SERVICE
       RELEASE     BKST               FREE FOR NEXT FIFO ENTRY
       TRANSFER    ,DETPR             RETURN FOR CPU PROCESSING INTERVAL
 TERM  TABULATE    JBTIM              DETERMINE JOB EXECUTION TIMES
       MARK        2                  ESTABLISH NEW ENTRY JOB TIME
       TRANSFER    ,DETPR             BLOCK COUNT HERE IS NUMBER  OF TERM
*
*
```

```
        GENERATE    X34
        SAVEVALUE   4,V$THRPT          EVALUATE THROUGHPUT
        TERMINATE   1                  SHUT OFF THE RUN
*
*
*
*
* LOGIC TO ESTIMATE LINE LENGTH DISTRIBUTION
*
* QUEUE LENGTH DISTRIBUTION - CPU
        GENERATE    ,,,1,,2,F          SEED CPU LNGTH-DISTRIB SEGMENT
 KEEP1  ASSIGN      1,Q$CPUQ           P1=CURRENT LINE LENGTH
        MARK        2                  P2=ABSOLUTE CLOCK TIME
        TEST NE     P1,Q$CPUQ          WAIT FOR A NET CHANGE IN LINE LENGHT
        TABULATE    CPULN,MP2          RECORD OLD LING LENGTH, WT BY DURATI
        TRANSFER    ,KEEP1             BACK TO WAIT FOR NEXT LINE CHANGE
*
* QUEUE LINE LENGTH DISTRIBUTION - I/O
        GENERATE    ,,,1,,2,F          SEED IO  LNGTH-DISTRIB SEGMENT
 KEEP2  ASSIGN      1,Q$IOQ            P1=CURRENT LINE LENGTH
        MARK        2                  P2=ABSOLUTE CLOCK TIME
        TEST NE     P1,Q$IOQ           WAIT FOR A NET CHANGE IN LINE LENGHT
        TABULATE    IOQLN,MP2          RECORD OLD LING LENGTH, WT BY DURATI
        TRANSFER    ,KEEP2             BACK TO WAIT FOR NEXT LINE CHANGE
*
* QUEUE LINE LENGTH DISTRIBUTION - BKST
        GENERATE    ,,,1,,2,F          SEED BKS LNGTH-DISTRIB SEGMENT
 KEEP3  ASSIGN      1,Q$BKSTQ          P1=CURRENT LINE LENGTH
        MARK        2                  P2=ABSOLUTE CLOCK TIME
        TEST NE     P1,Q$BKSTQ         WAIT FOR A NET CHANGE IN LINE LENGHT
        TABULATE    BKQLN,MP2          RECORD OLD LING LENGTH, WT BY DURATI
        TRANSFER    ,KEEP3             BACK TO WAIT FOR NEXT LINE CHANGE
*
*
        START       1                  START THE FIRST RUN FOR 20 MIN
        RESET
```

```
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,2,1,2,F          N(DEGREE OF MULTIPROGRAMMING)=2
 DETPR  ASSIGN      1,1,2               2ND PROB DISTRIBUTION
        INITIAL     X11,1370            ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE SECOND
        RESET
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
 KEY    GENERATE    ,,,3,1,2,F          N=3
 DETPR  ASSIGN      1,1,3               3RD PROB DISTRIB
        INITIAL     X11,587             ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE THIRD
        RESET
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,4,1,2,F          N=4
 DETPR  ASSIGN      1,1,4               4TH PROB DISTRIB
        INITIAL     X11,357             ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE FOURTH
        RESET
        START       1
```

```
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,5,1,2,F          N=5
 DETPR  ASSIGN      1,1,5               5TH PROB DISTRIB
        INITIAL     X11,180             ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE FIFTH
        RESET
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,6,1,2,F          N=6
 DETPR  ASSIGN      1,1,6               6TH PROB DISTRIB
        INITIAL     X11,88              ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE SIXTH
        RESET
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,7,1,2,F          N=7
 DETPR  ASSIGN      1,1,7               7TH PROB DISTRIB
        INITIAL     X11,55              ADVANC5 FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE SEVENTH
        RESET
        START       1
```

```
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
 KEY    GENERATE    ,,,8,1,2,F          N=8
 DETPR  ASSIGN      1,1,8               8TH PROB DISTRIB
        INITIAL     X11,39              ADVANCE FOR NEW CPU PROCESSING INTER
        CLEAR       X5,X11,X22,X33,X34
        START       1                   START THE EIGHTH
        RESET
        START       1
        RESET
        START       1
        RESET
        START       2                   ASSUME STEADY STATE REACHED RUN 2 IN
*
        END
```

BIBLIOGRAPHY

A1 Abate, J. and Dubner, H., "Optimizing the Performance of Drum-like Storage," IEEE T.C. C-18, 11, (Nov. 1969), pp. 992-997.

B1 Bard, Y., "A Characterization of Program Paging in a Time-Sharing Environment," IBM Data Processing Division, Cambridge Science Center, (Feb. 1973), Ref. No. G320-2083.

B2 Belady, L.,"A Study of Replacement Algorithms for a Virtual Storage Computer," IBM System J., Vol. 5, No. 2, (1966), pp. 78-101.

B3 Buzen,J.P., Queuing Network Models of Multiprogramming, Ph.D. Thesis, Harvard University, (April 1971).

B4 Buzen, J. P., Optimizing the Degree of Multiprogramming in Demand Paging Systems, IEEE Compt. Soc. Conf., Boston, Mass., (Sept. 22-24,1971), pp. 141-142

B5 Buzen, J. P., "Analysis of System Bottlenecks Using a Queuing Network Model," Proc. ACM SIGOPS Workshop on System Performance Evaluation,(April 1971), pp. 82-103.

C1 Calingaert, P., "System Performance Evaluation: Survey and Appraisal," CACM, Vol. 10, No. 1, (Jan. 1967), pp. 12-20.

C2 Chang, W., "Preemptive Priority Queues," Operation Research, Vol. 13, No. 5, (Sept. 1965), pp. 620-623.

C3 Chang, W. and Wong, D. L., "Computer Channel Interference Analysis," IBM System J., Vol.16, No. 3, (May-June 1965), pp. 162-170.

C4 Chang, W., "Queues Feedback for Time-Sharing Computer System Analysis," Operation Research, Vol. 16, No. 3, May-June 1968, pp. 613-627.

C5 Chen, P., Optimal File Allocation, PH.D. Thesis, Harvard University, Camb., Mass., (May 1973).

C6 Coffman, E. G., Stochastic Models of Multiple and Time-Shared Computer Operations, Ph. D. Thesis, Dept. of Eng. Univ. of California, Los Angles, (June 1966).

C7 Coffman, E. G., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System," JACM, Vol. 16 No. 1, (Jan. 1969), pp. 73-90.

C8 Conway, R. W., Johnson, B. M., and Maxell, N. L., "Some Problems of Digital Systems Simulation," 5th International Convention of the Institute for Mangement Sciences, Philadelphia, Pa., (Oct. 1958).

C9 Conway, R. W., "Some Tactical Problems in Digital Simulation,' Management Science, Vol. 10, No. 1, (Oct. 1963), pp. 47-61.

C10 Cooper, R. B., Introduction to Queuing Theory, Macmillan Co. (1972), pp. 65-71.

C11 Corbato, F. J., "A Paging Experiment with the Multics System," Peshbach and Ingard, MIT Press, Camb., Mass., (1969).

C12 Crooke, S. and Minker, J., "KWIC Index and Bibliography on Computer Systems Simulation and Evaluation," Computer Sci. Center, Univ. Md., College Park, Md., (May 1969).

D1 Denning, P. J., "Effects of Scheduling on File Memory Operations," Proc. AFIPS SJCC (1967), pp. 9-18.

D2 Denning, P. J., "The Working Set of Program Behavior," Comm. ACM, Vol. 11, No. 5, (May 1968), pp. 323-333.

D3 Denning, P. J., "Thashing: Its Causes and Prevention," AFIPS Conf. Proc., Vol. 33, (1968), FJCC, pp. 915-922.

D4 Denning, P. J., "Virtual Memory," Computing Surveys of ACM, Vol. 2, No. 3, (Sept. 1970), pp. 153-189.

D5 Dewan, P.B., Dowaghey, C. E., and Wyatt, J. B., "OSSL- A Specialized Language for Simulating Computer Systems," Proc. AFIPS SJCC, (1972), pp. 799-814.

D6 Dimsdale, B. and Markowitz, H. M., "A Description of the Simscript Language," IBM Systems J., Vol. 3, No.1, (1964), pp. 57-67.

D7 Drake, A. W., Fundamentals of Applied Probability Theory, McGraw Hill Inc., (1967).

E1 Estrin, G., Muntz, R. R., and Uzgalis, R. C., "Modeling Measurement and Computer Power," Proc. AFIPS SJCC, (1972), pp. 725-738.

F1 Feller, W., An Introduction to Probability Theory and its Applications, Vol. 1, 3rd Edition, Wiley, New York,(1968).

F2 Frank, W., "Analysis and Optimization of Disk Storage Devices for Time-Sharing Systems," JACM, Vol. 16, No. 4, (1969), pp. 602-620

G1 Gaver, D. P. Jr., "A Waiting Line with Interrupted Service Including Properties," Journal of the Royal Stat. Soc., Vol. 13, No. 24, (1962), pp. 73-90.

G2 Gaver, D. P. Jr., "Probability Models for Multiprogramming Computer Systems," JACM, Vol.14, No. 3, (July), pp. 423-436.

G3 Gay, T. W. Jr., "A Queuing Model for Scan Conversion," Proc. AFIPS FJCC (1969), pp. 553-560.

G4 Gordon, G., "A General Purpose Systems Simulator," IBM Systems J., Vol. 1, (Sept. 1962), pp. 18-32.

G5 Gordon, W. J. and Newell, G. F., "Closed Queuing Systems with Exponential Servers," Operations Research, Vol. 15, No. 2, (April 1967), pp. 254-265.

G6 Gould, R. L., "An Improved General Purpose Simulator," IBM Systems J., 8, No. 1, (1969), pp. 1-37.

G7 Granger, R. L., and Robinson, G. S., "COMSL- A Communication System Simulation Language, Proc. AFIPS FJCC, (1970), pp. 407

H1 Hatfield, D. J., "Experiments on Page Size, Program Access Patterns, and Virtual Memory Performance," IBM J. of R. and D., Vol. 16 No. 1, (Jan. 1972), pp. 58-66.

H2 Holland, F. C. and Merikallio, R. A., "Simulation of a Multiprocessing System Using GPSS," IEEE Transactions on Syst. Science and Cybernetics, (Nov. 1968), pp. 395-400.

H3 Huesman, L. R. and Goldberg, R. P., "Evaluating Computer Systems Through Simulation," Computer J., Vol. 10, No. 2, (Aug. 1967) pp. 150-155.

H4 Humphrey, T. A., "The Role of Simulation in Computer," IEEE Transactions in Systems and Cybernetics, (Nov. 1966), pp. 393-395.

H5 Hutchinson, G. K. and Maguire, J. N., "Computer Systems Design and Analysis Through Simulation," Proc. AFIPS FJCC, (1965), pp. 161- 167.

J1 Jackson, J. R., "Networks of Waiting Lines," Operations Research, Vol.5, No. 4, (Aug. 1957), pp. 518-521.

J2 Jackson, J. R., "Jobshop-Like Queuing Systems," Manag. Sci., Vol. 10, No. 1, (Oct. 1963), pp. 131-142.

K1 Katz, J. H.,"Simulation of a Multiprocessor Computer System," Proc. AFIPS JCC, (1966), pp. 127-139.

K2 Kay, I. M., "An Over-the-Shoulder Look at Discrete Simulation Language," Proc. AFIPS SJCC, (1972), pp. 791-794.

K3 Kendall, D. G., "Some Problems in the Theory of Queues," J. of Roy. Stat. Soc., Series B, Vol. 13, No. 2, (1951), pp. 151-171.

K4 Kendall, D. G., "Stochastic Processes Occuring in the Theory of Queues and Their Analysis by the Method of the Imbedded Markov Chain," Ann. Math. Stat., Vol. 24, (1953), pp. 338-354.

K5 Kimbleton, S. R. and Moore, C. G., "A Probabilistic Framework for System Performance Evaluation," ACM SIGOPS Workshop on System Performance Evaluation, (April 1971), pp. 337-357.

K6 Kimbleton,S.,"Performance Evaluation- A Structural Approach," Proc. AFIPS SJCC, (1972), pp. 411-416.

K7 Kleinrock, L., "Time-Shared Systems: A Theoretical Treatment," JACM, Vol. 14, no. 2, (April 1967), pp.242-261.

K8 Kleinrock, L., "A Continuum of Time-Sharing Scheduling Algarithms," Proc. AFIPS SJCC (1970), pp. 435-458.

L1 Lucas, H. C. Jr., "Performance Evaluation and Monitoring," Computing Surveys, Vol. 3, No. 3, (Sept. 1971), pp. 79-91.

L2 Lum, V. Y., Ling, H. and Senko, N. E., "Analysis of a Complex Data Management Access Method by Simulation Modeling," Proc. AFIPS FJCC, (1970), Vol. 37, pp. 211-222.

M1 MacDougal, M. H., "Computer System Simulation: An Introduction," Computing Surveys, Vol. 2, No. 3, (Sept. 1970), pp. 191-209.

M2 Madnick, S. E. and Donovan, J. J., Operating Systems, Unpublished Notes for Course 6.802 at MIT, (1973).

M3 Madnick, S. E., "Sample Program Measurement Under CP-67," MIT, (Nov. 1972).

M4 Madnick, S. E., Storage Heirarchy Systems, Ph.D. Thesis, MIT, (June 1972).

M5 Meguire, J. N., "Discrete Computer Simulation - Technology and Applications - The Next Ten Years," Proc. AFIPS SJCC, (1972), pp. 815-826.

M6 Martin, F.F., Computer Modeling and Simulation, John Wiley and Sons, Inc. (1968), Introduction.

M7 Moore, C. G. III, Network Models for Large-Scale Time-Sharing Systems, Ph. D. Thesis Tr. No. 71-1 Dept. of Indust. Eng.,

Univ. of Michig., Ann Arbor, (April 1971).

N1 Newell, G. F., Applications of Queuing Theory, Chapman and Hall ltd., London, (1971) Introduction.

N2 Nielsen, N. R., "The Simulation of Time-Sharing Systems," CACM, Vol. 10, No. 7, (July 1967), pp. 397-417.

P1 Phister, M. Jr., "The Data Processing Industry," Colloquium, Harvard Univ. Center for Research in Computing Technology, (1972).

R1 Rechtschaffen, R. N., "Queuing Simulation Using a Random Number Generator," IBM Systems J., No. 3, (1972), pp. 255-271.

R2 Rice D. R., An Analytical Model for Computer System Performance Analysis, Ph.D. Thesis, Univ. of Florida, (1971).

S1 Saltzer, J. H., "A Simple Linear Model Of Demand-Paging Performance," Project MAC MQ 131, MIT, (Nov. 1972).

S2 Scherr, A. L., An Analysis of Time-Shared Computer Systems, Ph. D. Thesis, MAC-TR-18, MIT, (June 1965).

S3 Schrage, L. E., "The Queue M/G/1 with Feedback to Lower Priority Queues," Management Sci., Vol. 13, No. 7, (March 1967), pp. 466-474.

S4 Schreiber, T. J., A GPSS Primer, Graduate School of Business Adm., Ann Arbor, Michigan, (1972).

S5 Seaman, P. H., "On Teleprocessing System Design, Part VI, The Role of Digital Simulation," IBM System J., Vol. 5, No. 3, (1966).

S6 Seaman, P. H. and Soucy, R. C., "Simulating Operating Systems," IBM Systems J., Vol.8, No. 4, (1969), pp. 264-279.

S7 Sekino, A., Performance Evaluation of Multiprogrammed Time-Shared Computer Systems, Ph. D. Thesis of E.E. MAC-TR-103, (Sept. 1972).

S8 Shirley, T., Computers, Holt, Rinehart, and Winston Inc., N.Y., (1965), pp. 64-65.

S9 Smith, J. L., "An Analysis of Time-Sharing Computer Systems Using Markov Models," Proc. AFIPS SJCC, Vol. 28, (1966), pp. 87-95.

S10 Smith, J.L., "Multiprogramming Under a Page on Demand Strategy," CACM, Vol. 10, No. 10, (Oct. 1967), pp. 636-646.

T1 Takacs, L., Introduction to the Theory of Queues, Oxford Univ. Press. N. Y., (1962).

T2 Takacs, L., "Priority Queues," Operations Research, Vol.12, No. 1, (Jan.-Feb. 1964).

T3 Tien, J. M., Control of a Two Customer Class Interactive-Multi-Facility Queuing System, Ph. D. Thesis, MIT, (June 1972).

T4 TRW Systems, "Simulation of the CDC 7700 Tactical Operating System," Internal Company Memo, (1972).

W1 Wallace, V. L. and Mason, D. L., "Degree of Multiprogramming in Page-On-Demand Systems," Comm. ACM, Vol. 12, No. 6, (June 1969), pp. 305-308.

W2 Weizer, N. and Oppenheimer, G., "Virtual Memory Management in a Paging Environment," Proc. AFIPS SJCC, Vol. 34, (1969), pp. 249-256.

W3 Wyatt, J., Course Notes of a Book to be Published on OSSL, Harvard Class Am. 209, (1973).