NETWORK MODELS

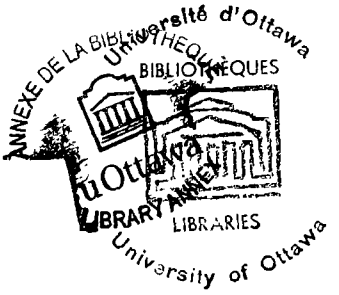
FOR

MULTIPROGRAMMING COMPUTER SYSTEMS

by

Mehmet Akın SENCER

Submitted to the School of
Graduate Studies in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy



Department of Electrical Engineering
Faculty of Science and Engineering
The University of Ottawa
Ottawa, Canada.

May 1974

NETWORK MODELS

FOR

MULTIPROGRAMMING COMPUTER SYSTEMS

BY

MEHMET AKIN SENCER

ABSTRACT

In this work, comprehensive models for the multiprogramming computer systems, which interrelate the arrival process, the heterogeneous processors, the finite main memory size and several classes of programs that are characterized by processing lengths and memory space requirements, are provided. In the models, the dynamic variation of the processing lengths and the memory space requirements of the programs in a paged main memory environment are allowed under various processing disciplines.

The models are analyzed in steady-state condition and the steady-state probability distributions of the queues forming in the system are obtained. Practical measures involving the main system resources are derived and applied to examples. The network of queues is the main tool of analysis where the related theory is generalized to a broad class of problems which may also be encountered in other fields such as computer

communication networks, transportation, mining and storage systems. Finally, areas for further research are suggested.

ACKNOWLEDGEMENT

I wish to express my sincere appreciation to my thesis advisor, Professor C. L. Sheng of University of Ottawa, for the support and confidence he has conveyed to me throughout this work. I wish to thank my co-advisor, Professor N. U. Ahmed of University of Ottawa, for his helpful suggestions and encouragement during the preparation of the thesis. I also acknowledge here the value of the friendly discussions I had with Professor I. Sahin (formerly with University of Ottawa) in the early days of my work.

This research was, in part, supported by The National Research Council of Canada grant; to them go my thanks. I wish to thank Mr. C. Lemieux of Bell Canada for the support he has given the thesis and also making Bell Canada computer available.

Finally, the excellent secretarial work of Miss Susan Shaver is sincerely acknowledged.

CONTENTS

CHAPTER I

MULTIPROGRAMMING COMPUTER SYSTEMS AND RELATED RESEARCH

1.0 MOTIVATION

In this work we present a study on the problems of congestion and the resource allocation in computer systems. The hardware resources in a computer system can be grouped into the following subsystems:

a) central processing unit(s), CPU('s),

b) memory space,

c) input - output processor(s), IOP('s).

There are also the software resources such as compilers, loaders, control programs and operating system (programs).

The resources of a computer system are made available to the users of the system through the programs which they write to make the computer system perform several tasks. Sometimes the users of the system do not have to submit individual programs to the computer system. In this case the user of the system just initiates a program sequence already stored in the computer, such is the situation in an inquiry-response or a process control system.

In computers, we observe a system where the user community initiates demand as a set of tasks to be

performed. The results of the processing of these tasks are returned to the users in various forms. This relation of the user community and the computer system is shown in Figure 1.

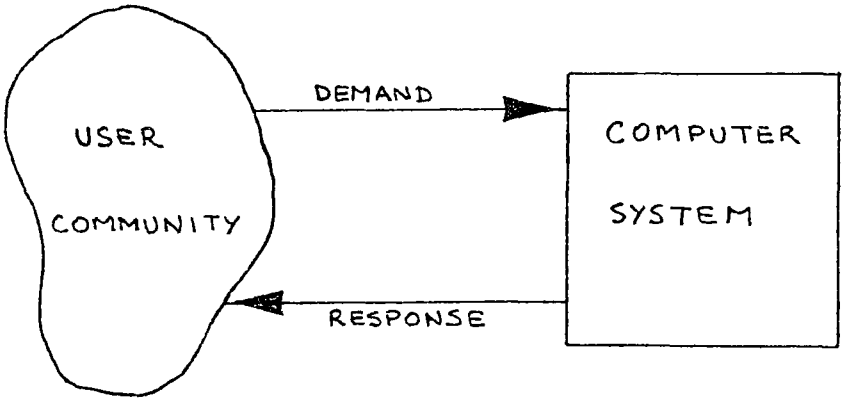


Figure 1 The relation of computer system and the user community.

We want to analyze here the behaviour of a computer system under user demand and also to observe how the system parameters, in terms of finite system resources, affect the response of the computer system. Because of the random nature of the demand on a computer system, we shall be employing mathematical models where our tool of analysis will be the theory of stochastic processes or more specifically the theory on the network of queues.

We believe that in this study we have advanced the state of the understanding of the computer systems from the congestion and resource allocation points of view by including the effects of the main system resources in our

models and results. The results obtained also further extend the existing theory on the network of queues which has applications in the fields other than computer systems.

1.1 DEFINITIONS

It would be convenient to define a number of terms as a basis of our subsequent discussions. Although the following definitions are not new in the language of computers there is usually some confusion about what every author means by them.

Time-sharing: To have time-sharing there has to be more than one user (or program) present in a processor at the same time, demanding service. The time-sharing facility serves each user a small amount at a time toward their service completion at each visit. If the time between each serving (intervisit time) for the same user is sufficiently short (and similarly for the other users in the system) then it appears to the users that the processor is serving them continuously and simultaneously. In fact, they have been served intermittently and one-at-a-time. This type of operation in a computer system is referred to as time-sharing. Usually it is the capability of the CPU which is time-shared, because of its faster operating speed compared to the demand made on it.

In present multiprogramming systems there may be more than one CPU and they, together with the IOP's, can overlap their operations in providing time-sharing.

Multiprogramming: If in a computer system there can be several programs being processed at the same time, then this system is said to have the multiprogramming capability.

Multiprocessing: The term multiprocessing implies that there is more than one independent processor (CPU or IOP) available for the programs in a computer system. Each processor is independent from the others in the sense that it may overlap its action with the others. Thus multiprogramming implies multiprocessing.

Interactive (conversational) programming: When a user of a computer system can interact with his program or the system during or between the processing intervals of the program, then this type of programming is referred to as interactive or conversational.

2.0 MEMORY CENTERED MODEL OF COMPUTER SYSTEMS

In this section, a general multiprogramming computer is to be discussed and a descriptive model for such a system is to be provided. The mathematical substance to this model is to be given in chapters two and three.

The reason for choosing a multiprogramming system for the basis of our analysis is that this kind of system is representative of the most of the existing computers. It seems, at least for the near future, the use of the multiprogramming systems with larger numbers of processors is the only way to increase the computing available power. In this respect the networks of computers can be considered a special type of a multiprogramming system [1].

A general hardware organization of multiprogramming computer system is shown in Figure 2. Briefly a CPU is the sub-system where the programmed arithmetical and logical operations are performed. The memory is the place where the program instructions and other necessary information such as data, compilers, operating system programs etc. are stored. All the devices used in handling of the information flow in and out of the memory are referred to here as the IOP's.

The sequencing and control of all the operations in a computer system is performed by the hardware and software devices which are collectively called the operating system.

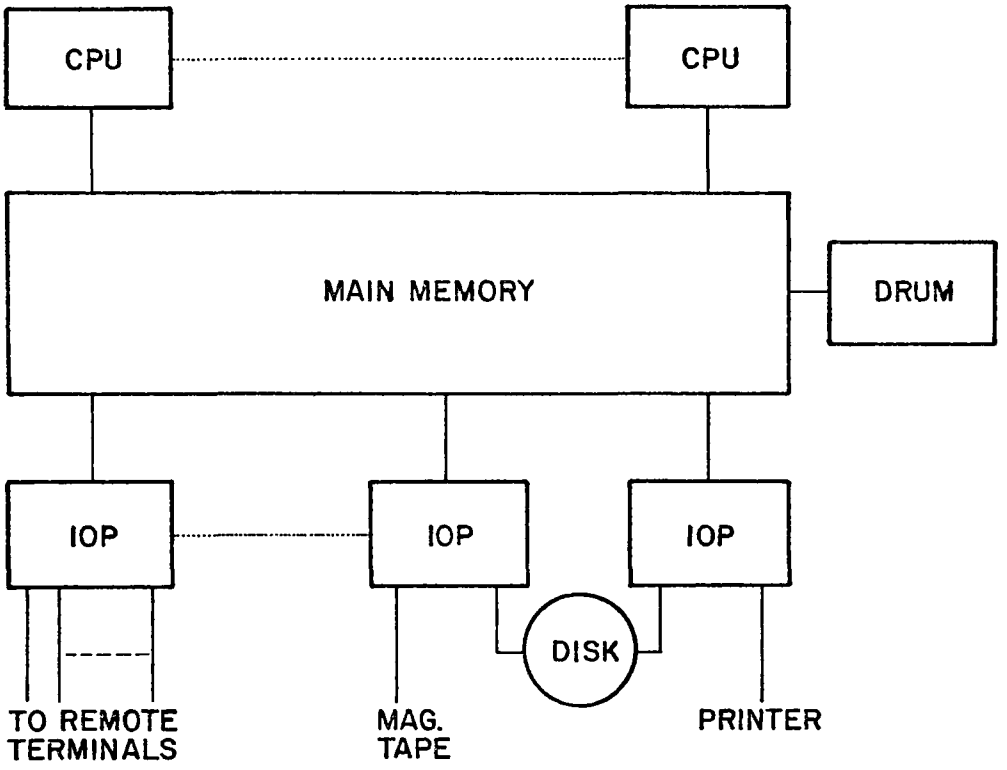


Figure 2 Basic hardware configuration in a multiprogrammed computer system.

To obtain a perspective of the operation let us examine how a program progresses through a simplified multiprogramming computer system. First, a program is originated by a user (a human or another machine) at a terminal (say a card reader) and introduced through an IOP into the main memory. When the input process of the

program is completed, assuming that it needs to be processed by a CPU, it is placed in a queue with other requests, if there are any. The CPU queues are served by a CPU (or CPU's) without delay as long as they are not empty, according to some predetermined priority rule. Here, we assume that the CPU's are operating independently of each other and the IOP's. During the processing by a CPU a program may need some IOP action. At that time the program would be placed in the particular IOP queue. This program, later, when IO action is completed, may be returned for further CPU processing and so on, until the task is completed and the program departs from the system through an IOP.

It is clear from the above simplified explanation that programs demand service from the processors they require during their life in the computer system. Sometimes the programs may have to wait for the availability of a particular resource thus forming the above mentioned queues in the system. These queues are conceptual and formed in the main memory and they only represent a particular arrangement of the processing sequence of the programs, although the space occupied by each program in the main memory may change during the processing steps.

Therefore we are going to employ the "memory centered" model (the terminology first mentioned by Watson [2]) of

the multiprogramming computer system to describe the congestion of programs before the processing units in the main memory.

The memory of the computer systems is not monolithic in structure. According to the speed of operation and function, we divide the system memory resource into, (a) the main memory, which is faster and more expensive per unit of storage space, (b) the auxiliary memory, which is of slower speed and cheaper.

As it was explained in the description of the multiprogramming system (without explicitly identifying) the main memory operates in conjunction with the faster processors - the CPU's. The IOP's handle the transfer of data between the main and the auxiliary memories or the user terminals. The size of the main memory is limited with the cost and the technological considerations of the information storage. Thus, the limited main memory space has to be utilized efficiently and without undue inconvenience to the system user. To realize this, there are techniques to increase utilization of the main memory such as "paging, virtual memory, segmentation" (cf. [2]) in addition to the multiprogramming.

From the resource management point of view the technique of paging interests us most here. The other

two techniques (virtual memory and segmentation) are devised to accommodate larger program sizes in limited physical main memory (or sub-units of main memory) space without unnecessary programmer interventions. To reduce the unusable idle space in the main memory, due to departure of programs of varied sizes at different times, the programs and the main memory are divided into fixed, equal and contiguous pieces which are called pages. In this way, the programs are loaded in multiples of pages into the main memory. Thus the wasted main memory space is limited to the empty part of the last page per program. The pages of a program may be stored at different places in the memory, not necessarily next to each other, thus further increasing the employment of the idle space in the main memory.

One other advantage of paging is realized in increasing the possible number of active programs (thus increasing the degree of multiprogramming) by limiting each program to occupy only a number of pages in the main memory. This restriction of number of pages belonging to a given program in the main memory may result in having only a part of a program to be resident in the main memory. In this case when an executing program references to a page which is absent from the main memory (page exception) it has to be brought in.

When the main memory is full or the program has the maximum allowed number of pages in the main memory, to bring in a new page, one of the pages already in the main memory has to leave (or overwritten if it has not been changed since copied in from the auxiliary memory or no longer needed). This decision, to choose the page to be replaced (without the intervention of the programmer), is taken by the "page replacement algorithm" of the operating system. If the page replacement algorithm has not been designed properly the page traffic between the main and the auxiliary memories increases the system overhead substantially.

The following factors have direct influence on the congestion properties of a multiprogrammed computer system:

(a) the size of the main memory,

(b) the size of a page,

(c) the number of different programs that can be actively resident in the main memory - degree of multiprogramming,

(d) the time needed to exchange (swap) a page between the main and the auxiliary memories,

(e) the frequency of a page being swapped,

(f) the processing (length) requirements per page,

(g) the number of available processors and their speeds.

Clearly some of the factors listed above (not necessarily independent of each other) have non-deterministic characters and cannot be determined a priori for each program.

The models we propose and study in this work incorporate the relationship between the main system resources and their utilization as briefly explained above. The models to be developed follow the organization of the hardware resources and the sequence of processing a program receives through these sub-systems. The software resources required in the operation of such a system is assumed to be available for each program together with the associated hardware resource. This is possible if all the software resources involved are "re-entrant" and there are enough copies of them, which is practically the case in modern multiprogramming computer systems.

As a tool of anlysis of the congestion properties in a computer system the theory of the stochostic processes, in particular the queueing theory, suggests itself naturally because of the random nature of the origination times of the programs, their processing lengths and their memory requirements.

3.0 PREVIOUS RELATED RESEARCH

In the past fifteen years there has been an intense effort to study the congestion problems related to the

computer systems. However, most of this effort was directed to the analysis of the problems confined to the individual resources, such as the CPU, IOP and the main memory. Except for a few attempts these resources have been studied in isolation from each other. The models and the results obtained in these works are of interest to us here only in their description and the insight into the nature of the problems with each of the sub-systems involved. The models we later propose for the multiprogramming computer systems, encompassing the main system resources, justify their generality by comparison with these previous partial models.

The previous work on the congestion problems of the multiprogramming computer systems is to be reviewed here under the headings of processor models, memory partitioning models and multiprogramming models. The network models which are applicable to a wider class of problems is to be treated last.

3.1 PROCESSOR MODELS

In a computer many parts of the system can be considered as processors. Broadly, one can classify them under CPU('s) and IOP('s), though in detail one can cite adders, registers, channel controller(s),

buses, storage modules (disks, tapes, pages, etc.), buffers and terminals. The demand on these processors is generated in the form of programs, program pieces, bytes or bits to be processed, usually fed by other processors or coming from outside of the system. As we have seen in the previous section the nature of the generation of demand on processors and its process requirements is stochastic in nature. Therefore the theory of queues, which up to now has provided an extensive treatment of stochastic service systems, has been also utilized in the analysis of the sub-systems of computers.

A good review of the application of the one stage (single and multiple server) queueing models for computer systems has been given by Chang in [3]. He has also provided the main references in queueing theory as well as the applications of the theory in computer systems. Further references to the theory of queues can be found through Kingman [4] and Bhat [5]. The contributions to the theory of queues after 1969 can be found in the journals of ORSA, Applied Probability and Management Sciences.

A particular survey of the application of queueing theory to the time-sharing computer systems has been provided by McKinney [6], where the various models of

the time-sharing CPU is presented along with the classification according to the characteristics of the arrival and service processes. Recently Kleinrock [7], Hellerman [8], Rasch [9], Coffman, et al. [10], Sakata, et al. [11], Bhat and Nance [12], Adiri [13], Adiri and Avi-Itzhak [14], Kleinrock [15], Heacox and Purdom [16], Nance, et al. [17], and Kleinrock and Muntx [18] contributed to the analysis of the various isolated processors in a computer system.

We are not going to elaborate on these owrks here as some of them have been reviewed elsewhere and the analysis of the isolated processors is not our central interest. We shall refer to the particular works when necessity arises.

3.2 MEMORY PARTITIONING MODELS

The effects of page replacement algorithms in the performance of the multiprogramming computer systems have been studied by various authors. Among the main investigators, Belady [19] has classified the page replacement algorithms into "random", "first-in-first-out" and "least recently used (or changed)" types and compared them using simulation with given test programs.

Fine, et al. [20] and Varian and Coffman [21] investigated the demand paging and concluded that the increased paging traffic between the main and the

auxiliary memories may seriously decrease the efficiency of the system.

Dening [22] has proposed the "working set" (set of pages which are most recently used) model. Based on this model he defined the concepts of memory and processor demand which he has utilized to find the balance condition of a system.

Oden and Shedler [23] have described a stack algorithm for demand paging. In this model they have provided the distributions of the number of pages allocated to a running program. They have also calculated the expected values of the running time of a program for a given number of pages and the running interval of multiprogrammed jobs. They have indicated the superiority of the variable partitioning of the main memory.

Coffman and Ryan [24] have studied the main memory partitioning and have shown the superiority of the dynamic variable partitioning over the fixed one using paging rate as a performance criterion.

3.3 MULTIPROGRAMMING MODELS

The multiprogramming computer systems can best be examined by including the multiprogramming property of such systems in the models. That is, by allowing the

processors in the model to work independently, in overlapped fashion. The modelling of multiprogramming systems has taken two separate directions, one being the deterministic scheduling and the other being the queueing theory approach.

The deterministic scheduling of the programs on various processors in a computer system was attempted by representing each program by a directed, acyclic graph. These graphs were arranged according to "Gnatt charts" (cf. [25]) over the processors involved. Heller [25], was the first to employ the technique. Ochner [26] has described a programming technique to accomplish the sequencing of the order dependent tasks on a number of processors. Manacher [27] also has employed Gnatt charts to eliminate the unnecessary elongations of task completion times with respect to a prepared schedule, due to variations in the task lengths. Graham [28] was first to show the "anomalies" (elongation of the total processing time) due to the increased number of processors. Muntz and Coffman [29] have presented an optimum assignment of partially ordered task sets to two identical processors by preemptive scheduling. Recently, Ramamoorthy et al [30] have been able to minimize the total execution time and determine the minimum number of processors required for a set of tasks represented by a directed graph model.

All these attempts have required a precise, a priori definition of the programming steps and the processing times for each program. Except in [27], in a limited way, all the analysis of deterministic type has assumed all the programs to be present at the beginning of the execution. The above points restrict the applicability of this type of analysis of multiprogramming systems to very special cases.

The queueing theory approach offers more realistic models to the multiprogramming computer systems although it has its own shortcomings.

In the analysis of multiprogramming the "cyclic queueing" model, which was first introduced by Koenigsberg [31], to study the coal mining operations, plays an important role. In this model a number of processors are arranged in cyclic stages (one processor at each stage), where the output of one stage forms the input of the next one. The system model is "closed", that is the number of customers (or programs in our terminology) in the system is fixed. In the study by Koenigsberg each processor has a negative exponential service distribution.

Kleinrock [32] was the first to employ 2-stage cyclic queueing model to study the multistage sequential processors. The model of the time-sharing computer

system proposed by Sherr [33] can also be classified as a 2-stage cyclic queueing model. In the closed system model, one stage was comprised of the sources (terminals each associated with a human operator) which can be considered as peripheral processors and a CPU formed the other stage. Each peripheral processor is dedicated to a single program and while this program is in the CPU, the peripheral processor cannot initiate or process any other program. When a program returns from the CPU to its source, it is reinitiated after a "think time". The think times and the length of processing each program receives from the CPU at each visit are assumed to be negative exponentially distributed. In this work the queues in the system have been analyzed.

The 2-stage cyclic queueing model has been further developed by Gaver [34] by incorporating an arbitrary number of identical processors in one of the stages (available to any program), the IO stage. The "busy period" analysis for the CPU stage is provided under the assumption of various processing distributions for the CPU such as exponential, hyperexponential etc.

Mitrani [35] has utilized the 2-stage cyclic queueing model, with single servers in both stages having negative exponential service time distributions, to study the effects of the degree of multiprogramming

on the response time.

The 2-stage cyclic queueing models with multiple servers in one of the stages have also been used to study the behaviour of the queues forming in the system by Buzen [36], Baskett and Gomez [37] and Baskett and Muntz [38, 39].

It should be noted that in the cyclic queueing models the number of programs in the system is kept at an arbitrary but fixed number. Thus it has been assumed that there is a constant backlog of programs in the system, awaiting processing, in the auxilliary memory possibly. In these models there is an approximation relating the input and output of the system. When a program departs from the system through one of the stages, say IO stage, then a new program is assumed to join the next stage, namely the CPU stage, thus keeping the number of programs circulating in the system fixed. It should be remarked that in such a case the entry and the exit points of the model do not reflect the practical operation of a computer system. If a program cycles large numbers of times in the system then the effect of this discrepancy is expected to diminish [34].

Although the above researchers have seen the necessity of it, none of them included the effects of the storage requirements of the programs in their models

except in some cases observing the effects of the degree of multiprogramming [24 - 36].

3.4 NETWORK OF QUEUES MODELS

The network of queues in the general context describe the behaviour of customers who, after entering a system composed of collection of service centers, gets served in a service center and then is routed to another service center until departure from the system. The queues may form before each service center where customers are served according to their class in some predetermined service order. In our terminology, the customers represent the programs to be processed at each group of processors.

The main work on the network of queues has been done by Jackson [40, 41]. Before him, J.R.R. Jackson [42] and Finch [43] treated special forms of network of queues and similarly the cyclic queueing model of Koenigsberg [31], which we saw in Section 3.3, all of which can be easily deduced from the more general results of Jackson.

Jackson has considered, [41], arbitrary numbers of service centers with finite numbers of identical servers each, where there is single class of customers. The system is open with no limit on the number of customers in the system. It is assumed that the new customers arrive in the form of Poisson processes to each service center

and the service lengths are negative exponentially distributed. The customers are routed probabilistically in the system until departure. The service discipline at each service center is arbitrary with no forced idle times or wasted work. He has obtained steady-state distribution of the number of customers for open (with unlimited storage room) and closed systems. He also has shown that in open systems with unlimited storage room, the queue at each service center can be analyzed independently.

Posner and Bernholtz [44] have considered closed network of queues with exponential servers where customers experience random delays between service centers and have derived steady-state distribution of the queue sizes in the system. In a companion paper [45] they have analyzed the same system with several classes of customers.

Recently Baskett and Muntz [38] using the idea of "local balance" of Chandy [46] have extended the theory to cover service time distributions which can be represented by rational Laplace transforms utilizing the method of stages given by Erlang [47] and Cox [48], under particular service disciplines.

One should also mention the work done by Gordon and Newell [49], on the closed network of queues, although the case they have studied, as they later acknowledged [50], is a special case treated by Jackson [41].

4.0 NEW MODELS

In this work we present models for the multiprogramming computer systems based on the extended theory of network of queues. In these models, apart from the processor stages of any configuration, we include the effects of the finite main memory space as well as permitting new arrivals and departures of programs from the system (Chapter II). In the next step (Chapter III) we generalize and include different classes of programs, each having different memory space and processing requirements that can dynamically change during the processing steps. The behaviour of such a system is analyzed in steady-state and practical results are given In the above ways, the models presented are more comprehensive than the ones presented in references [36 - 39].

Further, we extend the existing theory of network of queues by including the space requirements of different classes of customers which can change dynamically in the course of service steps in the system. Thus, the theory presented in this work is of a more general nature than the existing one [38, 41, 44 - 46] and it can treat a larger class of problems that are encountered in transportation, inventory systems, in the network of computers and job-shop like systems.

CHAPTER II

NETWORK OF QUEUES MODEL FOR COMPUTER SYSTEMS

1.0 GENERAL

In this chapter we provide a mathematical model for the multiprogramming computer systems which have been described in the previous chapter.

The structure of the model to be presented is a network of queues. As it was pointed out before, in Section 3.4 of Chapter I, these types of models for the multiprogramming computer systems are not new. In this and the next chapters we generalize the network of queues model for the computer systems and obtain practically useful results.

The multiprogramming computer system consists of groups of processors (service centers) where an arriving program gets processed before being routed to another group of processors or departs from the system. First we discuss the basic assumptions necessary in the derivation of the mathematical model. These assumptions are about the nature of the arrival and service (processing) length processes of the programs and how the programs are selected for service (service discipline) and how they behave after service completions.

1.1 THE ARRIVAL PROCESS

We suppose always in this work that the time ranges over the interval $[0,\infty)$. The new programs are assumed to arrive to the IOP stages in the form of a time-homogeneous Poisson process. That is during any time interval $(t, t+h]$, $h>0$,

i) The probability of exactly one arrival in the interval is proportional to the length of the interval,

$$P\{\text{exactly one arrival during } (t, t+h]\} = \lambda h + o(h).$$

ii) The probability of more than one arrival in the interval is $o(h)$.

$$P\{\text{more than one arrival during } (t, t+h]\} = o(h).$$

iii) The probability of no arrival in the interval is given by

$$P\{\text{no arrival during } (t, t+h]\} = 1-\lambda h + o(h).$$

iv) The occurrence or non-occurrence of an arrival in the interval is independent of any other arrivals or the time since the last arrival.

Where $o(h)/h \to 0$ as $h \to 0$ and the proportionality constant $\lambda>0$ can be shown to be the average arrival rate.

Alternately, let us denote by $\tau_1, \tau_2, \ldots, \tau_n, \ldots$ the arrival instants of the programs. We suppose that the inter-arrival times $\Theta_n = \tau_{n+1} - \tau_n$ $(n=0,1,\ldots;\ \tau_0 = 0)$ are mutually independent, positive random variables with the distribution function

$$P\{\theta_n \leqq x\} \triangleq F(x) \qquad (n = 0,1,2,\ldots),$$

where (1)

$$F(x) = \begin{cases} 1-e^{-\lambda x} & \text{for } x \geqq 0, \\ 0 & \text{for } x < 0, \end{cases}$$

then $\{\tau_n\}$ is said to be a Poisson process. The average inter-arrival time is equal to $1/\lambda$.

Since the probability of an arrival in any interval is independent of the time since the last arrival, the Poisson process has the unique Markov or "memoryless" property (cf. Feller [51], Takács [52]).

We also note that a combination of a number of Poisson input streams into one stream results in a Poisson process, having an arrival rate, which equals the sum of the individual arrival rates of the input streams. Conversely, if we assign each arrival on a Poisson stream to one of the number of other streams independently, the resulting individual streams will be still Poisson with the arrival rates proportional to the probability of assignment to a particular stream (cf. Cox and Miller [53]).

The assumption of the Poisson process for the arrival of programs or program pieces to a multiprogramming computer system is justified where there is practically a large number of program-originating sources and these attempt to access the system, each time, independent of

each other and the system status. Such is the case in the general purpose time-sharing computer systems, telephone switching machines (a special purpose computer in modern installations) and a computer system connected to a network of computers. In the cases where the above conditions are not justified or no other data is available for the system under study, the Poisson assumption may form a useful first approximation to the actual process.

Additionally, Poisson process has been utilized in the analysis of computer systems by Chang [3], Kleinrock [7], Rasch [9], and others [10, 11, 13, 14, 18].

However, Coffman and Wood [54] have collected data which indicate that the exponential distribution is only a rough approximation to the inter-arrival times. They have shown that a biphase or triphase hyper-exponential distribution would fit the data better.

1.2 THE PROCESSING MECHANISM

In this section we refer to the general properties of processing mechanism. The specification of the service process per stage is to be given latter.

We consider the programs (or program pieces) are processed at each processor stage (CPU or IO stage) before having departed for another processor stage or

leave the system.

The processing times of the programs are supposed to be identically distributed, independent, positive random variables, also independent of the arrival process at each processor stage. However, the departure (service completion) rate, μ', of the programs may depend on both the number of programs and processors in that stage.

There may be a number of identical processors at each processor stage attending the programs in an arbitrary service order with the condition that there is no idle processor if there is a program in the queue in that stage and there is no wasted work due to interruptions. That is to say that we allow service discipline of the types "first come first served", "random order of service", "processor sharing" [7], "pre-emptive-resume priority", "time-slicing", etc. This is due to the identical behaviour of the queue lengths in the system under these service disciplines. We do not allow the service orders where there is set-up times for the processors, or "preemptive-head of the line priority" arrangements.

The processing time of the nth program be denoted by χ_n. Then we define

$$P\{\chi_n \leq x\} \triangleq H(x)$$

where

$$H(x) = \begin{cases} 1-e^{-\mu' x} & \text{for } x \geq 0 \\ 0 & \text{for } x < 0. \end{cases} \qquad (2)$$

The exponential distribution has the memoryless property [51].

We point out here that in a multiprogramming computer system the life of a program is composed of alternating sequence of IO and compute times (CPU processing). There is no evidence that there is dependence between the successive CPU and IO processing times. Also the successive processing lengths of a given program in a particular processor stage can be assumed to be mutually independent. Therefore, we suppose, for our model, each processor is kept busy for a random length of time, per program, per visit, which is sampled from an exponential distribution particular to that stage of processors. This assumption is also a mathematically convenient one.

The exponential processing lengths were utilized to represent IO and CPU processing by Chang [3], Rasch [9], Coffman et al [10], Adiri [13], Adiri and Avi-Itzhak [14], Sherr [33], Gaver [34], and others [36 - 39].

The discrete equivalent of exponential distribution, that is the geometric distribution, for processing lengths was utilized by Kleinrock [7], Chang [55], Coffman [56] and Coffman and Kleinrock [57].

However, Fife [58] has collected a set of measurement data to indicate that the exponential distribution for processing lengths is only an approximation, where triphase hyper-exponential distribution was a better fit.

The assumption of exponential distribution for processing lengths can be relaxed by approximating complex distributions with the method of exponential stages (which, in fact, can be easily treated by the network of queues model) due to Erlang (cf. [47]) and Cox [48].

2.0 MATHEMATICAL MODEL

The configuration of the model for a general multiprogramming computer system is shown in Figure 1. Accordingly,

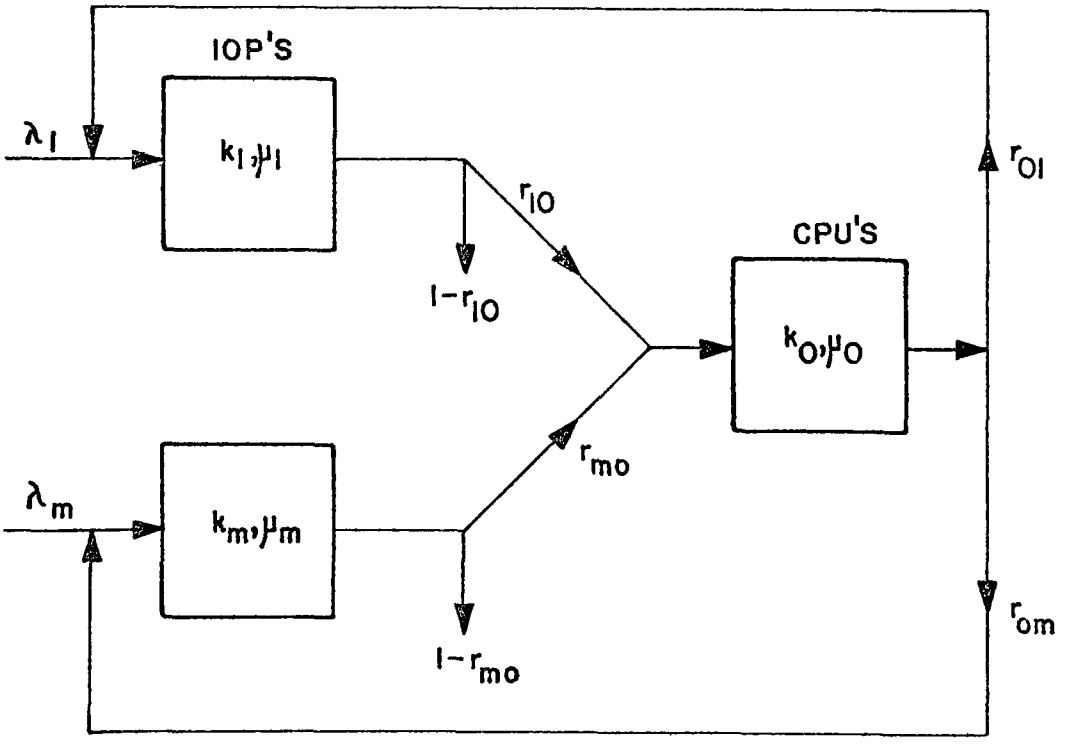


Figure 1 Configuration of the System Model

there are m+1 stages of processors. The stage i has $k_i \geq 1$ identical processors where i = 0,...,m. The stage 0 is the CPU stage and stages 1 to m correspond to m group of IOP's where there are k_i (i=1,...,m) identical processors at each stage.

The new programs are assumed, in the open system case, to arrive to the ith IOP stage in the form of time-homogeneous Poisson process with an average arrival rate $\lambda_i \geq 0$, (i=1,...,m). The maximum number of programs in the system is limited to N. If there is less than N programs in the system, a new arrival is accepted into the system and is immediately placed in the common queue before the particular IOP stage. If the number of programs in the system is N, then any new arrival is assumed to leave the system never to return.

Here we consider that each program currently in the system has one unit of storage space (a page) reserved for it in the main memory.

A program (or a piece of it) after being processed, in the ith IOP stage according to some service discipline say first come first served, either proceeds to queue for CPU processing or departs from the system with the probabilities r_{i0} and $1-r_{i0}$ respectively, for $1 \leq i \leq m$.

The programs demanding CPU processing are served by

k_o CPU's in their order of service. Thus, as in an IO stage, one of the idle CPU's selected randomly takes in for processing the program with the highest priority among the waiting ones in the common CPU - queue. It may be that the processing in the CPU may proceed in time slices for each program as in the time-sharing computer systems. Here we place no restriction in the manner in which programs are processed other than the rules stated in Section 1.2.

The programs, after completing the CPU processing or requiring the IO action (e.g. data transfer), return to the ith IOP stage with the probability r_{0i}, where $\Sigma_{i=1}^{m} r_{0i} = 1$.

In this model, the processing times of the CPU's and the IOP's are assumed to be mutually independent and distributed exponentially. The processors in the stage i (i=0,...,m) have the instantaneous service rate μ'_i which is the function of the number of processors and the programs at stage i at that time.

The routing probabilities, r_{ij}, of programs between the processor stages, $0 \leq i, j \leq m$, are assumed to be stationary and to be determined by measurements or predictions.

2.1 DEFINITION OF STATES AND MARKOV PROPERTY

We are interested in the behaviour of the queues in

the system forming before each processor stage. This information will provide us where the bottlenecks in the system are and how the common memory space, the operating speed and the number of processors at each stage affect the performance of the system. Thus, we study the number of programs at each stage in our model.

Let $P(n_0,\ldots,n_m;t)$ denote the probability of the state of the system with n_i programs in stage i (being processed or waiting) at time t, (i=0,...,m).

The vector $(n_0,\ldots,n_m)$ belongs to m+1 dimensional state-space of NxNx...xN. By definition, $0 \leq n_i \leq N$ for i = 0,...,m, and probability of a non-existent state is assumed to be zero.

Then using the relations defining the Poisson process (Section 1.1) and also equation (2) and employing elementary rules of probability theory we have $P(n_0,\ldots, n_m;t+h)$ equal to the sum of the following probabilities.

i) $P(n_0,\ldots,n_i,\ldots,n_m;t)\ [1-\lambda_i h+o(h)]\ [1-\mu'_i h+o(h)]$
for $0 \leq i \leq m$,

ii) $P(n_0,\ldots,n_i-1,n_{i+1},\ldots,n_m;t)\ [\lambda_i h+o(h)]$
for $0 \leq i \leq m$,

iii) $P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_m;t)\ (1-r_{i0})\ [\mu'_i h+o(h)]$

for $0 \leq i \leq m$,

iv) $P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_j-1,n_{j+1},\ldots,n_m;t)$

$.r_{ij}[\mu'_i h+o(h)]$ for $0 \leq i \leq m$,

v) P {two or more transitions in (t,t+h]) = o(h).

We observe that the transition probabilities in the interval (t,t+h] only depend on the state of the system at time t. This establishes the Markov property of the stochastic process defined over the states $(n_0,\ldots,n_m)$ of the system in the time interval $[0,\infty)$. The state probabilities where any of the entries, n_i, is negative, is assumed to be zero.

2.2 SYSTEM EQUATION

Now we write the system equation of the model presented by summing the probabilities (i) to (v) given in the previous section and equating it to $P(n_0,\ldots,n_m;t+h)$. In writing the system equation we observe the following rules and notations besides the ones given before.

Let a binary variable C be defined as

$$C=1 \quad \text{if} \quad \sum_{i=0}^{m} n_i < N$$

$$C=0 \quad \text{if} \quad \sum_{i=0}^{m} n_i \geq N \ . \tag{3}$$

Denote by

$$\alpha_i(n) = \min\{n, k_i\} \qquad \text{for } i=0,\ldots,m \tag{4}$$

and

$$\varepsilon(n) = \min\{n, 1\} \qquad \text{for } n=0,\ldots,N \tag{5}$$

where n equals the number of programs and k_i equals the number of processors in stage i.

We assume instantaneous service completion rate μ'_i at stage i is given by

$$\mu'_i = \alpha_i(n_i)\,\mu_i \qquad \text{for } 0 \leq n_i \leq N \text{ and } i=0,\ldots,m, \tag{6}$$

where $\mu_i > 0$ is a constant.

Then

$$\begin{aligned} P(n_0,\ldots,n_m;t+h) = {} & \{1-[C\sum_{i=1}^{m}\lambda_i + \sum_{i=0}^{m}\alpha_i(n_i)\mu_i]h\}. \\ & .P(n_0,\ldots,n_m;t) \\ & + \sum_{i=1}^{m}\lambda_i\varepsilon(n_i)h\,P(n_0,\ldots,n_i-1,n_{i+1},\ldots,n_m;t) \\ & + \sum_{i=1}^{m}\alpha_0(n_0+1)\,\mu_0 r_{0i}\varepsilon(n_i)h\,P(n_0+1,n_1,\ldots,n_i-1,n_i,\ldots,n_m;t) \\ & + \sum_{i=1}^{m}\alpha_i(n_i+1)\,\mu_i r_{i0}\varepsilon(n_0)h\,P(n_0-1,n_1,\ldots,n_i+1,n_{i+1},\ldots,n_m;t) \end{aligned} \tag{7}$$

$$+ C \sum_{i=1}^{m} \alpha_i(n_i+1)\ \mu_i\ (1-r_{i0})h\ P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_m;t)$$

$$+ o(h). \qquad (7)$$

2.3 STEADY-STATE EQUATION

Here we obtain the steady-state equation of (7).

From the both sides of equation (7) we subtract $P(n_0,\ldots,n_m;t)$ then divide by h and let $h\to 0$, left-hand side becomes the derivative of the state probability with respect to time while $o(h)/h\to 0$. In steady-state, observing that the derivative should equal to zero and also denoting the steady-state probabilities by $P(n_0,\ldots,n_m)$, as they are stationary in time, we have for the steady-state equation of the system.

$$\begin{aligned}
0=-&[C\sum_{i=1}^{m} \lambda_i + \sum_{i=0}^{m} \alpha_i(n_i)\mu_i]\ P(n_0,\ldots,n_m) \\
&+ \sum_{i=1}^{m} \lambda_i \varepsilon(n_i)\ P(n_0,\ldots,n_i-1,n_{i+1},\ldots,n_m) \qquad (8) \\
&+ \sum_{i=1}^{m} \alpha_0(n_0+1)\mu_0 r_{0i} \varepsilon(n_i)\ P(n_0+1,n_1,\ldots,n_i-1,n_i,\ldots,n_m) \\
&+ \sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i r_{i0} \varepsilon(n_0)\ P(n_0-1,n_i,\ldots,n_i+1,n_{i+1},\ldots,n_m) \\
&+ C\sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i(1-r_{i0})\ P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_m)\ .
\end{aligned}$$

2.4 SOLUTION OF THE STEADY-STATE EQUATION

Let us denote by Λ_i, i=0,...,m, the average arrival rate of programs to the processor stage i, which is the sum of both internal and external average arrival rates of the programs. Then, by observation, we write

$$\Lambda_i = \lambda_i + \sum_{j=0}^{m} r_{ji}\Lambda_j \quad \text{for } i=0,\ldots,m. \tag{9}$$

For the explanation of equation (9) a similar argument to the one provided in [41] can also be made.

The solution of the steady-state equation (8) is given by the following theorem which is different from the previously proved theorems for the general network of queues by Jackson [40, 41], in that, it permits up to an arbitrary but a fixed number, N, of programs (customers) in the system. We leave the treatment of the general network of queues to Chapter IV.

THEOREM 1: The joint, steady-state probability $P(n_0,\ldots,n_m)$ of an open system with finite storage space and single class of programs, defined by equation (8), that there are n_i programs in the processor stage i, for i=0,...,m, is given by

$$P(n_0,\ldots,n_m) = B(N) f_0(n_0)\ldots f_m(n_m) \tag{10}$$

$$\text{for } 0 \leq \sum_{i=0}^{m} n_i \leq N,$$

where

$$f_i(n_i) = (\Lambda_i)^{n_i} \prod_{m=1}^{n_i} \frac{1}{\mu_i \alpha_i(m)} \tag{11}$$

or

$$f_i(n_i) = \begin{cases} (\Lambda_i/\mu_i)^{n_i}/n_i! & \text{for } n_i=0,\ldots,k_i \\ (\Lambda_i/\mu_i)^{n_i}/(k_i! k_i^{n_i-k_i}) & \text{for } n_i=k_i,\ldots,N. \end{cases}$$

Λ_i's are given by equation (9). The normalization constant B(N) is determined from the normalization condition to be

$$B^{-1}(N) = \sum_{0 \leq n_0+\ldots+n_m \leq N} \prod_{i=0}^{m} f_i(n_i). \tag{12}$$

PROOF: The existence of the steady-state probability distribution of equation (7) is proved by observing that we are dealing with a Markov process of finite number of states, mutually inter-communicating (cf. Cox and Miller [53], pp. 183 - 185). Therefore, the normalized solution of equation (8) is a proper and unique probability distribution. When N becomes infinite for the existence of the steady-state distribution we must have $\Lambda_i/(\mu_i k_i) < 1$, for i=0,...,m, and the above theorem reduces to the one given in [40], where the processor stages can be treated statistically independently.

Let us assume that the steady-state probabilities satisfy the following relations:

$$P(n_0,\ldots,n_m) = B(N) f_0(n_0)\ldots f_m(n_m) \tag{13}$$

where

$$f_i(n) = (\Lambda_i)^n \prod_{m=1}^{n} \frac{1}{\mu_i \alpha_i(m)} \qquad \text{for } 0 \leq n \leq N \tag{14}$$

Then from (4), (13) and (14) we have,

$$\frac{P(n_0,\ldots,n_i-1,n_{i+1},\ldots,n_m)}{P(n_0,\ldots,n_m)} = \frac{\mu_i \alpha_i(n_i)}{\Lambda_i} \tag{15}$$

$$\frac{P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_m)}{P(n_0,\ldots,n_m)} = \frac{\Lambda_i}{\mu_i \alpha_i(n_i+1)} \tag{16}$$

$$\frac{P(n_0,\ldots,n_i+1,n_{i+1},\ldots,n_j-1,n_{j+1},\ldots,n_m)}{P(n_0,\ldots,n_m)} = \frac{\Lambda_i \mu_j \alpha_j(n_j)}{\Lambda_j \mu_i \alpha_i(n_i+1)} \tag{17}$$

Dividing both sides of the steady-state equation, i.e. equation (8), by $P(n_0,\ldots,n_m)$ and utilizing relations (15) to (17), we have,

$$C \sum_{i=0}^{m} \lambda_i + \sum_{i=0}^{m} \mu_i \alpha_i(n_i) = \sum_{i=0}^{m} \lambda_i \varepsilon(n_i) \frac{\mu_i \alpha_i(n_i)}{\Lambda_i} + \sum_{i=1}^{m} \alpha_0(n_0+1) \mu_0 r_{0i} \varepsilon(n_i) \frac{\Lambda_0 \mu_i \alpha_i(n_i)}{\Lambda_i \mu_0 \alpha_0(n_0+1)} \tag{18}$$

$$+ \sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i r_{i0}\varepsilon(n_0) \frac{\Lambda_i \mu_0 \alpha_0(n_0)}{\Lambda_0 \mu_i \alpha_i(n_i+1)}$$

$$+ C \sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i(1-r_{i0}) \frac{\Lambda_i}{\mu_i \alpha_i(n_i+1)} \tag{18}$$

From equation (9) and the special form of the routing probabilities we have,

$$\Lambda_i = \lambda_i + r_{0i}\Lambda_0 \qquad \text{for } i=1,\ldots,m \tag{19}$$

$$\Lambda_0 = \sum_{i=1}^{m} r_{i0}\Lambda_i \ .$$

Then

$$\begin{aligned} \sum_i \lambda_i &= \sum_i \Lambda_i - \Lambda_0 \sum_i r_{0i} \\ &= \sum_i \Lambda_i - \Lambda_0 \\ &= \sum_i (1-r_{i0})\Lambda_i \ . \end{aligned} \tag{20}$$

Using (20) we simplify (18),

$$\sum_{i=0}^{m} \mu_i \alpha_i(n_i) = \sum_{i=1}^{m} \lambda_i \varepsilon(n_i) \frac{\mu_i \alpha_i(n_i)}{\Lambda_i}$$

$$+ \sum_{i=1}^{m} r_{0i}\varepsilon(n_i) \frac{\Lambda_0 \mu_i \alpha_i(n_i)}{\Lambda_i}$$

$$+ \frac{\varepsilon(n_0)\mu_0\alpha_0(n_0)}{\Lambda_0} \sum_{i=1}^{m} r_{i0}\Lambda_i$$

$$= \sum_{i=1}^{m} \varepsilon(n_i)\mu_i\alpha_i(n_i)\frac{(\lambda_i+r_{0i}\Lambda_0)}{\Lambda_i} + \varepsilon(n_0)\mu_0\alpha_0(n_0)$$

$$= \sum_{i=0}^{m} \varepsilon(n_i)\mu_i\alpha_i(n_i)$$

As $\varepsilon(n_i)\alpha_i(n_i) = \alpha_i(n_i)$, from (4) and (5), the R.H.S. equals the L.H.S. This completes the proof.

THEOREM 2: The marginal steady-state probabilities of queue length $P_i(n_i)$, where n_i equals the size of the queue at processor stage i, of the system defined by equation (8) is given by,

$$P_i(n_i) = f_i(n_i)B(N) \qquad 0 \leq \sum_{r\neq i} n_r \leq N-n_i \qquad \prod_{\substack{j=0 \\ j\neq i}}^{m} f_j(n_j) \qquad (21)$$

for $i=0,\ldots,m$ and $n_i=0,\ldots,N$

PROOF: The proof of the above theorem follows from Theorem 1, equation (10). By fixing n_i and then summing over all the possible states we obtain (21).

The form of the equation (21) suggests the following relation: we define a new constant,

$$B_i^{-1}(N-n_i) \triangleq \sum_{0 \leq \sum_{r\neq i} n_r \leq N-n_i} \prod_{\substack{j=0 \\ j\neq i}}^{m} f_j(n_j) \qquad (22)$$

which is similar in form to $B^{-1}(N)$ with the ith term missing from the product. Then, the marginal steady-state probability of the queue length being n_i at stage i is given by equation (21) can be written as,

$$P_i(n_i) = f_i(n_i)\ B(N)/B_i(N-n_i)$$

$$\text{for} \quad 0 \leq i \leq m \quad \text{and} \quad 0 \leq n_i \leq N. \tag{23}$$

3.0 CLOSED SYSTEM MODEL

The closed network of queues were defined in Chapter I where there are no arrivals and departures from the system. Thus, in a multiprogramming computer system, a fixed number of programs (customers) circulate in the system being routed from one stage of processor, after receiving service, to another with a probabilistic policy (a deterministic routing policy is a special case of probabilistic policies).

Hence the routing probabilities in a close system of m+1 processor stages is given by

$$\sum_{j=0}^{m} r_{ij} = 1 \quad \text{for } 0 \leq i \leq m. \tag{24}$$

As there are no external arrivals to the system $\lambda_i \equiv 0$ for $0 \leq i \leq m$.

If we assume that the processing length distributions are mutually independent for the m processor stages and

distributed exponentially as given in Section 2.0 and by equation (6), then system equations for such a system model can be obtained from specializing the equation (7) by dropping the appropriate terms.

Then the steady-state equation of a closed system model for multiprogramming computer system can be shown to be given by,

$$\begin{aligned} 0 = &-\left[\sum_{i=0}^{m} \alpha_i(n_i)\mu_i\right] P(n_0,\ldots,n_m) \\ &+ \sum_{i=1}^{m} \alpha_0(n_0+1)\mu_0 r_{0i}\varepsilon(n_i)\, P(n_0+1,n_1,\ldots,n_i-1,n_i,\ldots,n_m) \\ &+ \sum_{i=1}^{m} \alpha_i(n_i+1)\mu_i r_{i0}\varepsilon(n_0)\, P(n_0-1,n_1,\ldots,n_i+1,n_{i+1},\ldots,n_m) \end{aligned} \tag{25}$$

The solution of equation (25) follows the same lines as the equation for the open system, equation (8), and we have the following theorem:

THEOREM 3: The joint steady-state probability $P(n_0,\ldots,n_m)$ of the closed finite storage system defined by equation (25), that there are n_i programs in the processor stage i, for i=0,...,m, is given by,

$$P(n_0,\ldots,n_m) = B'(N) f_0(n_0)\ldots f_m(n_m) \tag{26}$$

for $\sum_{i=0}^{m} n_i = N$

where

$$f_i(n_i) = (\Lambda_i)^{n_i} \prod_{m=1}^{n_i} \frac{1}{\mu_i \alpha_i(m)} \tag{27}$$

or

$$f_i(n_i) = \begin{cases} (\Lambda_i/\mu_i)^{n_i}/n_i! & \text{for } n_i = 0,\ldots,k_i \\ (\Lambda_i/\mu_i)^{n_i}/(k_i! k_i^{n_i-k}) & \text{for } n_i = k_i,\ldots,N. \end{cases}$$

The normalization constant B'(N) is determined from the normalization condition to be

$$[B'(N)]^{-1} = \sum_{\substack{m \\ \sum_{i=0} n_i = N}} \prod_{j=0}^{m} f_j(n_j). \tag{28}$$

PROOF: The existence and the uniqueness of the steady-state distribution is given by the same reasoning as of Theorem 1. The proof also follows similar lines.

Theorem 3 is given before, independently by Jackson [41] and Gordon and Newell [49], here we obtain it by specialization of the open system with finite storage.

In a similar way, let us define a new constant $B_i'(N-n_i)$ as,

$$[B_i'(N-n_i)]^{-1} = \sum_{\substack{m \\ \sum_{\substack{r=0 \\ r\neq i}} n_r = N-n_i}} \prod_{\substack{j=0 \\ j\neq i}}^{m} f_j(n_j). \tag{29}$$

Then, the steady-state probability of finding n_i programs in stage i is given by,

$$P_i(n_i) = f_i(n_i) \quad B'(N)/B_i'(N-n_i) \tag{30}$$

for $0 \leq i \leq m$ and $0 \leq n_i \leq N$.

The closed system model given in [41, 49] was utilized by Buzen [36], Baskett and Gomez [37] and Baskett and Muntz [38], for the multiprogramming system. As shown above, their results form special cases of our model which represents the computer system as a network of queues with finite storage space, where there is a variable number of programs.

4.0 DETERMINATION OF ARRIVAL RATES

The average arrival rates in the model of a multiprogramming computer system as described in previous sections are the only unknown in the steady-state probability distribution of the number of programs at each stage.

First we define $R \triangleq \{r_{ij}\}$, $\Lambda \triangleq \{\Lambda_i\}$ and $\lambda \triangleq \{\lambda_i\}$ where R is a (m+1) by (m+1) square matrix, Λ and λ are column vectors of dimension (m+1), we rewrite equation (9) as,

$$\begin{bmatrix} 1 & -r_{10} & -r_{20} & \cdots & -r_{m0} \\ -r_{01} & 1 & 0 & & 0 \\ -r_{02} & 0 & 1 & 0 & 0 \\ & & & & \\ -r_{0m} & 0 & & 0 & 1 \end{bmatrix} \begin{bmatrix} \Lambda_0 \\ \\ \\ \\ \Lambda_m \end{bmatrix} = \begin{bmatrix} 0 \\ \lambda_1 \\ \\ \\ \lambda_m \end{bmatrix} \tag{31}$$

or $$R\,\Lambda = \lambda \tag{32}$$

When the determinant of R is not identically zero, (31) can be uniquely solved for Λ_i's and it can be shown by recursive reasoning that the solution is given by,

$$\Lambda_0 = \frac{\sum_{i=1}^{m} r_{i0}\lambda_i}{1 - \sum_{i=1}^{m} r_{i0}r_{0i}} \tag{33}$$

$$\Lambda_i = \frac{\sum_{\substack{j=1 \\ j\neq i}}^{m} \lambda_j r_{j0} r_{0j} + \left(1 - \sum_{\substack{j=1 \\ j\neq i}}^{m} r_{j0}r_{0j}\right) \lambda_i}{1 - \sum_{j=1}^{m} r_{j0}r_{0j}} \tag{34}$$

for i=1,...,m.

In the case of closed system the arrival rates to each processor stage is given by the relation

$$\Lambda_i = \sum_j r_{ji}\Lambda_i \tag{35}$$

or specifically

$$\Lambda_0 = \sum_{j=1}^{m} r_{j0}\Lambda_j$$

$$\Lambda_i = r_{0i}\Lambda_0 \qquad \text{for } i=1,\ldots,m. \tag{36}$$

In this case Λ_i's can be determined within a multiplicative constant. It is important to make the distinction that Λ_i may be treated as the average arrival rates of programs to a processor stage i but it does not generally represent the flow rate of programs through this stage in steady-state. This can be seen clearly in the case when in a processor stage, the number in the queue becomes larger than the number of processors at that stage, the rate of flow through that stage becomes the processing rate of the processors. This situation exists in the case where $\Lambda_i/k_i\mu_i \geq 1$ in steady-state and it should be observed that this ratio is independent of the number in the queue and only depends on routing probabilities and the external arrivals. In such cases Λ_i can be thought of as the average number of appearances of stage i in particular routing pattern, [41].

5.0 SYSTEM MEASURES OF MERIT

In this section we are going to derive some system measures of practical interest to a system designer or an operator. The knowledge of how the system behaves under the variation of certain system parameters may

influence the decisions regarding the provisioning of the system resources.

Let us define a traffic intensity measure for stage i, ρ_i, $i=0,\ldots,m$, as

$$\rho_i \triangleq \frac{\Lambda_i}{k_i \mu_i} \tag{37}$$

a) The probability of finding the system empty in steady-state is given by,

$$P(0,\ldots,0) = B(N) \tag{38}$$

b) The probability of finding z programs in the system at steady-state is given by

$$P(n_0+\ldots+n_m = z) = B(N) \sum_{\sum_i n_i = z} \prod_{j=0}^{m} f_j(n_j)$$

$$= B(N)\ [B^{-1}(z) - B^{-1}(z-1)]. \tag{39}$$

c) Then, the average number of programs in the system is given by,

$$Q = \sum_{z=1}^{N} z\ P(n_0+\ldots+n_m = z)$$

$$= B(N) \sum_{z=1}^{N} z \sum_{\Sigma n_i = z} \prod_{j=0}^{m} f_j(n_j)$$

$$= B(N) \sum_{z=\Sigma n_i=1}^{N} z \prod_{j=0}^{m} f_j(n_j)$$

$$Q = B(N) \sum_{z=1}^{N} z[B^{-1}(z) - B^{-1}(z-1)]. \tag{40}$$

d) The average utilization of the main memory is given by the ratio Q/N.

e) The steady-state probability of overflow (an arriving program can not enter the system because it is already full), β, from (39), is given by,

$$\beta = P(\sum_{i} n_i = N) \tag{41}$$

$$= 1 - \frac{B(N)}{B(N-1)} \tag{42}$$

f) The average flow time (time from entrance to the open system, to the time of departure from the system) of a program through the system can be obtained from Little's result [59],

$$W = Q/\lambda_T \tag{43}$$

where

W = the average flow time of a program through the system,

Q = the average number of programs in the system, given by (40),

λ_T = the total average arrival rate of programs that joins to the system, i.e.

$$\lambda_T = (1-\beta) \sum_{i=1}^{m} \lambda_i \tag{44}$$

We remark here that equation (43) can be applied to find the flow time between any two points in the system, provided that we know the average queue size between these points and also the average rate of arrival of the programs.

g) Next, we would like to give an expression for the average length of queue formed in front of each processor stage in the system.

Let Q_i denote the average length of queue in processor stage i, then

$$Q_i = \sum_{z=1}^{N} z\, P_i(z) \qquad \text{for } 0 \leq i \leq m. \tag{45}$$

From equation (23), we have,

$$Q_i = B(N) \sum_{z=1}^{N} z\, f_i(z)/B_i(N-z) \tag{46}$$

All the system measures considered up to this point in this section are for open systems with finite storage. Similar expressions can be written for closed systems. In the case where $N \to \infty$, the network model can be decomposed into independent processor stages and the computational problems of system measures simplify considerably.

6.0 EXAMPLE

Here, we are going to apply the results to a numerical example. Consider a three stage system where stage 0

corresponds to a CPU and stages 1 and 2 to IO stages.

Let $k_0 = k_1 = k_2 = 1$, i.e. we have single processors at each stage.

Let the external average arrival rates be given as,

$\lambda_0 = 0$, $\lambda_1 = 2.2$ $\lambda_2 = 3.5$

and the routing probabilities be

$r_{00} = 0.$ $r_{10} = .6$

$r_{01} = .5$ $r_{11} = 0.$

$r_{02} = .5$ $r_{12} = 0.$

$r_{20} = .6$

$r_{21} = 0.$

$r_{22} = 0.$

The results are tabulated for different amount of common storage space, N, in the system, Table I and II. Also, varying the service rate of the processor in stage 2 its effect is observed.

TABLE 1

	Stage 0	Stage 1	Stage 2
Processing rates, μ_i	50	10	12
Average arrival rates, Λ_i	8.55	6.475	7.75
Traffic intensity, ρ_i	0.1710	0.6475	0.6479

		N = 6	N = 10	N = 14	N = 18
Average Queue Length per Stage, Q_i	0	0.1841	0.2008	0.2050	0.2060
	1	1.2465	1.6223	1.7719	1.8212
	2	1.2450	1.6198	1.7689	1.8179
Average total queue length, Q		2.6756	3.4429	3.7458	3.8451
Probability of finding system empty		0.1255	0.1077	0.1040	0.1031
Probability of finding system full		0.0836	0.0202	0.0047	0.0011
Average memory utilization		44.6%	34.4%	26.8%	21.4%
Average flow time, W		0.4694	0.6040	0.6572	0.6746

TABLE 2

	Stage 0	Stage 1	Stage 2
Processing rates, μ_i	50	10	6
Average arrival rates, Λ_i	8.55	6.475	7.75
Traffic intensity, ρ_i	0.1710	0.6475	1.2958

		N = 6	N = 10	N = 14	N = 18
Average Queue Length per Stage, Q_i	0	0.1358	0.1470	0.1503	0.1514
	1	0.780	0.9235	0.9726	0.9896
	2	3.3118	6.3924	9.9040	13.6639
Average total queue length, Q		4.2276	7.4629	11.0269	14.8050
Probability of finding system empty		0.0275	0.0081	0.0027	0.0009
Probability of finding system full		0.2970	0.2495	0.2355	0.2308
Average memory utilization		70.5%	74.6%	78.8%	82.3%
Average flow time, W		0.7417	1.309	1.935	2.597

CHAPTER III

DYNAMIC MEMORY SHARING

IN

MULTIPROGRAMMING COMPUTER SYSTEMS

1.0 GENERAL

In the previous chapter we have given a basic network of queues model for the multiprogramming computer systems. In this model we have assumed that a new program is permitted to enter the system when there is an idle unit space (a page) in the finite main memory. Also, it has been assumed that a program in the computer occupies one unit space in the main memory as long as it is in the system. When it departs from the system, by way of an IOP, it releases the memory space it occupies and that unit space becomes available immediately to another new program.

In this chapter we are going to include the effects of different memory requirements of programs in our analysis. We are going to take into account the varying memory requirements of programs on entering and during the processing steps in the system. Again, the model we utilize is going to be the network of queues with the appropriate extensions of the basic model.

The theory behind the analysis we are going to present has wide applications in the fields other than

the multiprogramming systems such as in the network of computers, transportation, inventory systems and job-shop type systems. We leave the generalizations to Chapter IV to gain continuity and clarity in the treatment of multiprogramming computer systems.

2.0 MATHEMATICAL MODEL

The configuration of the model for general purpose multiprogramming system we are going to consider here is similar to the one in Figure 1 of Chapter II. We have $m + 1$ stages (groups) of processors where stage i has $k_i \geq 1$ identical processors, $0 \leq i \leq m$.

The new programs belonging to class r are assumed, in the open system, to arrive to the ith IOP stage in the form of time-homogeneous Poisson processes with the average arrival rate $\lambda_{ri} \geq 0$ for $1 \leq r \leq R$ and $0 \leq i \leq m$.

Let there be N units of main memory space in the system (measured in some convenient units, possibly in pages). Assume that a program from class j is assigned j units of space when permitted to enter the system. We assume $R \leq N$ to be non-trivial. When a new program from class j arrives and finds the main memory has less than j units of free space, the program balks, never to return. A program releases the space it occupies when leaving the system after having obtained the necessary processing.

During the processing steps the memory requirements of the programs may change. To accomodate this fact we assume in the model that a class r program, after completing processing at processor group i, goes to processor group j and joins class s with the probability p' (r,i;s,j). In the open system model, the programs from class r after completing service at processor group i departs from the system with the probability p^*_{ri}, where

$$p^*_{ri} = 1 - \sum_s \sum_j p'(r,i;s,j) \qquad (1)$$

for

$$1 \leq r \leq R \quad \text{and} \quad 0 \leq i \leq m.$$

In the case of closed networks as there is no departures from the system $p^*_{ri} = 0$ for $1 \leq r \leq R$ and $0 \leq i \leq m$.

In the closed network model we consider that there is always a fixed number of programs, say L, in the system. The sum of the space requirements of all the programs may vary between L and N where $L \leq N$.

In our model we assume that the programs do not experience any delay in transition between the processing stages for the sake of simplicity. The transition delay can be introduced into the model rather easily (by inserting an extra stage with unlimited numbers of processors on the transition path) as was done in [38] or in more general form in [44, 45].

2.1 THE PROCESSING DISCIPLINE

Initially we assume the length of processing for each program, in each processing stage to be a random variable distributed mutually independently with the exponential law, also independent of the arrival process. We assume that the mean instantaneous rate of processing (service completion) for a class r program in processing stage j is specified by,

$$\mu'_{rj}(\underline{n}_j) = \mu_{rj} n_{rj} \phi_{rj}(\underline{n}_j) \tag{2}$$

where $\underline{n}_j = (n_{1j},\ldots,n_{Rj})$ denotes the presence of n_{rj} number of class r programs in processing stage j and $\mu_{rj} > 0$ is a constant, for $1 \leq r \leq R$, $0 \leq j \leq m$.

Here,

$$\phi_{rj} \triangleq \begin{cases} 1 & \text{for } |\underline{n}_j| \leq k_j \\ k_j/|\underline{n}_j| & \text{for } |\underline{n}_j| \geq k_j \end{cases} \tag{3}$$

and

$$|\underline{n}_j| \triangleq \sum_r n_{rj}$$

where k_j is the number of processors at stage j, for $0 \leq j \leq m$.

The processing mechanism given by (2) describes a policy such that there is no idle processor if there is a program in the queue in that stage and there is no wasted work due to interruptions or set up times. The process completion rate of class r programs depend on the

stage which they are in and also on the number of programs present at that time and the number of processors in that stage. This type of service discipline allows for "first come first served", "random order of service", "pre-emptive resume" type of priority within the same class, "time slicing" "round-robin" etc. types of disciplines in the consideration of the queue lengths in the system. This can be seen from [45], where although the expression (2) is derived under the "first come first served" assumption it also represents the mean processing rate times the mean probability of finding certain class of programs, at a given time, in service at a given processor stage without regard to their arrival order.

2.2 STATE DEPENDENT ROUTING PROBABILITIES

In the model presented above the transition of the programs from one class to another having greater memory space requirements depends on the total occupied space in the system at the time. Therefore the routing probabilities are state dependent.

Accordingly, a program demanding more space in the system (a transition from class r to s where r s) can only be satisfied if there is enough (s-r units) free space available at the time. Otherwise, we make the assumptions that the program requesting more space stays in its original class but may still proceed to the next

processor group in its routing. It is obvious, if a program demands less or the same amount of space (the transition from class r to class s where r = s) it can always be accomodated.

Now, we define a binary variable C(r,s) as

$$C(r,s) \triangleq \begin{cases} 1 & \text{for } \sum_{r=1}^{R} \sum_{j=0}^{m} rn_{rj} \leq N-(s-r) \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

Then,

$$\begin{aligned} p'(r,i;s,j) &= p(r,i;s,j)\ C(r,s) \quad s \neq r \\ p'(r,i;r,j) &= p(r,i;r,j) + [1-C(r,s)]p(r,i;s,j) \end{aligned} \tag{5}$$

where $p(r,i;s,j)$ is constant for $1 \leq r, s \leq R$ and $0 \leq i, j \leq m$.

The assumption just made about the routing discipline of programs in a multiprogramming computer system is a practical one. In this way programs may grow and occupy more memory space when they need it and if there is idle space available. When there is no idle space, if a program needs to bring a new page from the auxiliary memory or add a new page to the active pages belonging to it in the main memory, it changes place with one of its own active pages. This policy may be accomplished by applying the "least recently used" algorithm, locally.

Also, it is important to note that the programs are allowed in the system when there is enough space available for that class of programs. The memory space required for each class may be equivalent to the "working set" of those types of programs determined in a priori [22]. This type of main memory management is general enough to cover the dynamic variable partitioning as well as fixed partitioning models.

3.0 SYSTEM EQUATION

The system described above can be modelled as a Markov process by relating the state of the system at time t+h to that of time t and considering the number of programs from each class at each processing stage.

The method we are going to follow is similar to the one employed in Chapter II. Therefore, we are going to skip some of the steps here.

Let $P(\underline{n}_0,\ldots,\underline{n}_m;t)$ be the state probability of finding the system in the state $(\underline{n}_0,\ldots,\underline{n}_m)$ at time t, where the vector $\underline{n}_j = (n_{1j},\ldots,n_{Rj})$.

We also borrow the following notation from [45];

$$(\underline{n}_j)_{r+} = (n_{1j},\ldots,n_{rj}+1,\ n_{(r+1)j},\ldots,n_{Rj})$$
$$(\underline{n}_j)_{r-} = (n_{1j},\ldots,n_{rj}-1,\ n_{(r+1)j},\ldots,n_{Rj}). \tag{6}$$

Also define

$$\varepsilon(n_{rj}) \triangleq \min \quad (1,\ n_{rj}) \tag{7}$$

for

$$1 \leq r \leq R \quad \text{and} \quad 0 \leq j \leq m.$$

Then the system equation can be written as follows:

$$\begin{aligned}
P(\underline{n}_0,\ldots,\underline{n}_m;t+h) = {} & \{1-[\textstyle\sum_r \sum_j C(0,r)\lambda_{rj}+\sum_r \sum_j \mu'_{rj}(\underline{n}_j)]\ h\} \\
& .P(\underline{n}_0,\ldots,\underline{n}_m;t) \\
& +\textstyle\sum_r \sum_j \varepsilon(n_{rj})\lambda_{rj}h \\
& .P(\underline{n}_0,\ldots,(\underline{n}_j)_{r-},\underline{n}_{j+1},\ldots,n_m;t) \\
& +\textstyle\sum_r \sum_s \sum_j \mu'_{r0}(\underline{n}_0)_{r+}hp'(r,0;s,j)\ \varepsilon(n_{sj}) \\
& .P((\underline{n}_0)_{r+},\underline{n}_1,\ldots,(\underline{n}_j)_{s-},\underline{n}_{j+1},\ldots,\underline{n}_m;t) \\
& +\textstyle\sum_r \sum_s \sum_j \mu'_{sj}(\underline{n}_j)_{s+}hp'(s,j;r,0).\varepsilon(n_{r0}) \\
& .P((\underline{n}_0)_{r-},\underline{n}_1,\ldots,(\underline{n}_j)_{s+},\underline{n}_{j+1},\ldots,\underline{n}_m;t) \\
& +\textstyle\sum_r \sum_j C(0,r)\ \mu'_{rj}(\underline{n}_j)_{r+}hp^*_{rj} \\
& .P(\underline{n}_0,\ldots,(\underline{n}_j)_{r+},\underline{n}_{j+1},\ldots,\underline{n}_m;t) + o(h).
\end{aligned} \tag{8}$$

The equation (8) is written for an open system; however, for a closed system, as there would be no arrivals and departures, the terms containing λ_{rj} and p^{*}_{rj} for $1 \leq r \leq R$ and $0 \leq j \leq m$ would drop as they would become equal to zero.

We form the difference - differential equation by transposing the $P(\underline{n}_0,\ldots,\underline{n}_m;t)$ term to the lefthand side of (8) and then taking the limit as $h \to 0$ after dividing both sides by h.

3.1 STEADY-STATE EQUATION

The steady-state distribution for the state probabilities of the difference - differential equation obtained from (8) is of our main interest. This steady-state distribution exists and it is unique and proper as we have a finite state Markov process, described by equation (8), with all the states mutually communicating (cf. Cox & Miller [53] pp. 183 - 185). To find the steady-state probabilities of the system we write the steady-state equation corresponding to the equation (8) as follows by the process described in Chapter II, Section 2.3;

$$\left[\sum_r \sum_j C(0,r)\lambda_{rj} + \sum_r \sum_j \mu'_{rj}(\underline{n}_j)\right] P(\underline{n}_0,\ldots,\underline{n}_m)$$

$$= \sum_r \sum_j \varepsilon(n_{rj})\lambda_{rj} P(\underline{n}_0,\ldots,(\underline{n}_j)_{r-},\underline{n}_{j+1},\ldots,\underline{n}_m)$$

$$+ \sum_r \sum_s \sum_j \mu'_{r0}(\underline{n}_0)_{r+} p'(r,0;s,j)\ \varepsilon(n_{sj})$$

$$\cdot P((\underline{n}_0)_{r+},\underline{n}_1,\ldots,(\underline{n}_j)_{s-},\underline{n}_{j+1},\ldots,\underline{n}_m)$$

$$+ \sum_r \sum_s \sum_j \mu'_{sj}(\underline{n}_j)_{s+} p'(s,j;r,0)\ \varepsilon(n_{r0})$$

$$\cdot P((\underline{n}_0)_{r-},\underline{n}_1,\ldots,(\underline{n}_j)_{s+},\underline{n}_{j+1},\ldots,\underline{n}_m) \qquad (9)$$

$$+ \sum_r \sum_j C(0,r)\mu'_{rj}(\underline{n}_j)_{r+}\ p^*_{rj}$$

$$\cdot P(\underline{n}_0,\ldots,(\underline{n}_j)_{r+},\underline{n}_{j+1},\ldots,\underline{n}_m).$$

Again, for a closed system, terms containing λ_{rj} and P^*_{rj} for $1 \leqq r \leqq R$ and $0 \leqq j \leqq m$ become zero.

In a network of queues where a common storage is shared dynamically by the programs (customers) in the system and where the arrival and service processes are as given in Section 2, the steady-state probabilities for the number of programs from each class at each processing stage is obtained as the solution of the steady-state equation (9). Thus, we have the following theorem;

THEOREM 1: The joint, steady-state probability $P(\underline{n}_0,\ldots,\underline{n}_m)$ of an open system with finite common storage space and several classes of programs dynamically sharing space, defined by equation (9), where vector $\underline{n}_i = (n_{1i},\ldots,n_{Ri})$ represents n_{ri} programs belonging to class r in processor stage i,

is given by,

$$P(\underline{n}_0,\ldots,\underline{n}_m) = E(N)\ g_0(\underline{n}_0)\ldots g_m(\underline{n}_m) \tag{10}$$

where

$$g_i(\underline{n}_i) = \frac{1}{\phi_i(\underline{n}_i)} \prod_{r=1}^{R} (\Lambda_{ri})^{n_{ri}} \prod_{m=1}^{n_{ri}} \frac{1}{\mu_{ri}^{m}} \tag{11}$$

for $0 \leq \sum_{i=0}^{m} |\underline{n}_i|^* \leq N$, $|\underline{n}_i|^* = \sum_{r=1}^{R} n_{ri} r$

and $|\underline{n}_i| = \sum_{r=1}^{R} n_{ri}$.

Here,

$$\phi_i(\underline{n}_i) \triangleq \begin{cases} 1 & \text{for } |\underline{n}_i| \leq k_i \\ \dfrac{k_i!\,(k_i)^{|\underline{n}_i| - k_i}}{|\underline{n}_i|!} & \text{for } |\underline{n}_i| \geq k_i \end{cases} \tag{12}$$

and E(N) is the normalization constant to be determined from,

$$E^{-1}(N) = \sum_{0 \leq \sum_{j=0}^{m} |\underline{n}_j|^* \leq N} \prod_{i=0}^{m} g_i(\underline{n}_i) \ . \tag{13}$$

Λ_{rj} is given by the well known relation,

$$\Lambda_{rj} = \lambda_{rj} + \sum_{s} \sum_{i} p'(s,i;r,j)\Lambda_{si} \tag{14}$$

for $1 \leq r \leq R$ and $0 \leq j \leq m$.

PROOF: The proof follows similar lines of the proof of Theorem 1 in Chapter II, Section 2.4. Therefore, here we are only going to give the outline of the proof to the above theorem.

From equations (10) and (11) we note the following relations for $|\underline{n}_i| \geq k_i$:

$$\frac{P(\underline{n}_0,\ldots,(\underline{n}_j)_{r-},\underline{n}_{j+1},\ldots,\underline{n}_m)}{P(\underline{n}_0,\ldots,\underline{n}_m)} = \frac{\mu_{rj} n_{rj} k_j}{|\underline{n}_j| \Lambda_{rj}} \tag{15}$$

$$\frac{P(\underline{n}_0,\ldots,(\underline{n}_j)_{r+},\underline{n}_{j+1},\ldots,(\underline{n}_i)_{s-},\underline{n}_{i+1},\ldots,\underline{n}_m)}{P(\underline{n}_0,\ldots,\underline{n}_m)}$$

$$= \frac{\mu_{si} n_{si} (|\underline{n}_j|+1) k_i \Lambda_{rj}}{|\underline{n}_i| k_j \Lambda_{si} \mu_{rj} (n_{rj}+1)} \tag{16}$$

$$\frac{P(\underline{n}_0,\ldots,(\underline{n}_j)_{r+},\underline{n}_{j+1},\ldots,\underline{n}_m)}{P(\underline{n}_0,\ldots,\underline{n}_m)} = \frac{(|\underline{n}_j|+1)\Lambda_{rj}}{\mu_{rj}(n_{rj}+1)k_j} \quad . \tag{17}$$

Similar relations can be written for $|\underline{n}_i| \leq k_i$.

Then, substituting these relations along with (2) into the steady-state equation (9) it can be verified that the expression given by Theorem 1 satisfies (9) and therefore, is the required solution.

It should be noted that Theorem 1 does not give us a closed form solution to (9) as the Λ_{ij}'s, to be determined from equation (14), depend on state-dependent routing probabilities p'(r,i;s,j). In this case an approximate solution of equation (9) may be provided by either neglecting state-dependent terms in equation (14) or through a numerical procedure between equations (11) and (14) we may determine $p(\underline{n}_0,\ldots,\underline{n}_m)$. Here we adopt the first alternative because of its simplicity. The numerical approach based on Newton-Raphson technique is outlined in the appendix.

Neglecting the state-dependency in equation (14) amounts to assuming C(r,s) = 1 always in equation (5). That is the routing probabilities are considered to be fixed. Then equation (14) can be written as;

$$\Lambda_{rj} = \lambda_{rj} + \sum_{s}\sum_{s} p(s,i;r,j)\Lambda_{si} \qquad (18)$$

for $1 \leq r \leq R$ and $0 \leq j \leq m$.

This assumption can be justified where all the traffic intensities for each stage are less than one, which is the normal way of operating a system. From equation (4) we see that C(r,s) = 0 when the main memory is nearly full or full, which usually has smaller probability than other states of the system. Then, in Section 5.0 we utilize the above approximation to determine Λ_{rj}'s i.e. equation (18).

3.2 DISCUSSION OF THE RESULTS

Theorem 1 given above also applies to a closed system. This can be shown by rewriting equation (9) with $\lambda_{rj} = 0$ and $p^{*}_{rj} = 0$ and then verifying that theorem 1 still satisfies the new equation with $L \overset{\leq}{=} \sum_{i} |\underline{n}_i|^{*} \overset{\leq}{=} N$.

If we restrict the probability of transitions from one class to the other to be equal to zero in our system, i.e. when $p(r,j;s,i)=0$ when $r \neq s$ then the model does not have dynamically changing space requirements per program in the system. Additionally, assuming single processor at each processor stage and also assuming that each program occupies a unit space in a closed system, our result, given by Theorem 1, becomes identical to the one given by Posner and Bernholtz [45] for the no time-lag case between the transitions from one processor stage to the other. If we further restrict our model to have the same service length distribution for every class of program at a given processor stage, we then obtain the result provided by Baskett and Muntz [38], for "type 1" service center.

Thus, our result is more general than the previously provided ones in treating network of queues, which was proved here for only a particular model of the multiprogramming computer systems. Its generalization, to networks with arbitrary configuration, is going to be treated in Chapter IV.

3.3 MARGINAL DISTRIBUTIONS

Here, we are going to provide expressions for the various marginal steady-state distributions for finding particular patterns at a given processor stage. The following theorems summarize our results.

THEOREM 2: The marginal, steady-state probabilities of queue length at processor stage i, $P_i(\underline{n}_i)$, where vector $\underline{n}_i = (n_{1i},\ldots,n_{Ri})$ represents n_{ri} programs belonging to class r in processor stage i, is given by,

$$P_i(\underline{n}_i) = g_i(\underline{n}_i)\ E(N)\ E_i^{-1}(N-|\underline{n}_i|^*) \qquad (19)$$

for $0 \leq i \leq m$ and $0 \leq |\underline{n}_i|^* \leq N$

where $E_i(N-|\underline{n}_i|^*)$ defined similar to E(N) with the particular function $g_i(\underline{n}_i)$ is missing from the summation, thus,

$$E_i^{-1}(N-|\underline{n}_i|^*) = \sum_{0 \leq \sum_{r\neq i} |\underline{n}_r|^* \leq N-|\underline{n}_i|^*} \prod_{\substack{j=0 \\ j\neq i}}^{m} g_j(\underline{n}_j) \qquad (20)$$

PROOF: Theorem 2 can be proved by fixing for stage i the R-tuple $\underline{n}_i$ in equation (10), the joint probability expression, and summing it for all the other possible arrangements of programs in the rest of the system.

The next theorem gives the marginal steady-state probability of finding z programs in processor stage i such that $|\underline{n}_i| = z$, where $0 \leq |\underline{n}_i|^* \leq N$ restriction should be observed.

THEOREM 3: The marginal, steady-state probabilities of queue length (the number of programs) at processor stage i, $P_i(|\underline{n}_i| = z)$, where vector $\underline{n}_i = (n_{1i},\ldots,n_{Ri})$ and z is a positive constant, is given by,

$$P_i(|\underline{n}_i| = z) = E(N) \sum_{\Sigma_r n_{ri}=z} E_i^{-1}(N-|\underline{n}_i|^*) g_i(\underline{n}_i) \qquad (21)$$

for $0 \leq i \leq m$ and $0 \leq |\underline{n}_i|^* \leq N$.

PROOF: The proof is obtained by simply summing all the probabilities $P_i(\underline{n}_i)$, given by equation (19), such that $|\underline{n}_i| = z$.

THEOREM 4: The marginal, steady-state probabilities of queue length (the number of memory spaces occupied) at processor stage i, $P_i(|\underline{n}_i|^* = z)$, where vector $\underline{n}_i = (n_{1i},\ldots,n_{Ri})$ and z is a positive constant, is given by,

$$P_i(|\underline{n}_i|^* = z) = E(N)\, E_i^{-1}(N-|\underline{n}_i|^*) \sum_{\Sigma_r r n_{ri}=z} g_i(\underline{n}_i) \qquad (22)$$

for $0 \leq i \leq m$ and $0 \leq |\underline{n}_i|^* \leq N$.

PROOF: The proof is simply obtained by summing all the probabilities $P_i(\underline{n}_i)$, given by equation (19), such that $|\underline{n}_i|^* = z$.

4.0 OTHER PROCESSING DISCIPLINES

If we assume processor sharing service discipline, [7], then we have for service completion rate,

$$\mu'_{rj}(\underline{n}_j) = \mu_{rj} \frac{n_{rj}}{|\underline{n}_j|} \tag{23}$$

for $1 \leq r \leq R$ and $0 \leq j \leq m$.

Then under the processor sharing service discipline we have the following theorem;

THEOREM 5: For the system described by equation (9) and for processor sharing service discipline given by equation (23) the steady-state probability $P(n_0,\ldots,\underline{n}_m)$ of the system is given by,

$$P(\underline{n}_0,\ldots,\underline{n}_m) = E'(N) h_0(\underline{n}_0)\ldots\underline{h}_m(\underline{n}_m) \tag{24}$$

where

$$h_i(\underline{n}_i) = |\underline{n}_i|! \prod_{r=1}^{R} \frac{1}{n_{ri}!} \left[\frac{\Lambda_{ri}}{\mu_{ri}}\right]^{n_{ri}} \tag{25}$$

for $0 \leq \sum_{i=0}^{m} |\underline{n}_i|^* \leq N$, $|\underline{n}_i|^* = \sum_{r=1}^{R} n_{ri} r$

and $|\underline{n}_i| = \sum_{r=1}^{R} n_{ri}$.

The normalization constant E'(N) is to be determined from,

$$[E'(N)]^{-1} = \sum_{0 \leq \sum_{j=0}^{m} |\underline{n}_j|^* \leq N} \prod_{i=0}^{m} h_i(\underline{n}_i) \ . \tag{26}$$

and Λ_{rj} is given by,

$$\Lambda_{rj} = \lambda_{rj} + \sum_s \sum_i p'(s,i;r,j)\Lambda_{si} \tag{27}$$

for $1 \leq r \leq R$ and $0 \leq j \leq m$.

PROOF: The proof is similar to that of Theorem 1. Also, one can obtain the marginal distributions in this case.

We should note that equation (25) is a special form of equation (11) and can be obtained directly from it.

Similarly, if we have processing stages where there is always a processor available to a program coming to that stage, that is if there is no queueing, this may represent a stage which introduces random delay into the transit time of a program path. If we choose to represent this delay by a different exponential distribution for different class of programs the exiting rate from this special stage is given by,

$$\mu'_{rj}(\underline{n}_j) = \mu_{rj} n_{rj} \ . \tag{28}$$

Then the following theorem gives the number of programs in each class at such a processing stage.

THEOREM 6: For the system described by equations (9) and (28) the steady-state probability

$$P(\underline{n}_0,\ldots,\underline{n}_m) = E''(N)\ \ell_0(\underline{n}_0)\ldots\ell_m(\underline{n}_m) \tag{29}$$

where

$$\ell_i(\underline{n}_i) = \prod_{r=1}^{R} \frac{1}{n_{ri}!}\left[\frac{\Lambda_{ri}}{\mu_{ri}}\right]^{n_{ri}} \tag{30}$$

for $0 \leq \sum_{i=0}^{m} |\underline{n}_i|^* \leq N$, $|\underline{n}_i|^* = \sum_{r=1}^{R} n_{ri}.r$

The normalization constant E"(N) is to be determined from

$$[E''(N)]^{-1} = \sum_{0 \leq \sum_{j=0}^{m} |\underline{n}_j|^* \leq N} \prod_{i=0}^{m} \ell_i(\underline{n}_i) \tag{31}$$

and Λ_{rj} is given by

$$\Lambda_{rj} = \lambda_{rj} + \sum_{s}\sum_{i} p'(s,i;r,j)\Lambda_{si} \tag{32}$$

for $1 \leq r \leq R$ and $0 \leq j \leq m$.

PROOF: The proof follows the similar procedure of the previous proofs.

Theorems 5 and 6 are similar to the ones given in [38], where our results are proved under the condition the programs in the system share the common memory space, while their storage requirements may change at the end of

each processing step. The corresponding results in [38] only take into account the number at a processor stage in open or closed systems. In open systems they only consider the case with infinite storage space.

These results show the generality of our treatment which is not restricted in application only to the systems where "local balance" exists, [38], as we start from the first principles and do not make any assumptions on the presence of local balance.

The problems where there are different processing distributions than exponential distribution are encountered can be treated with good approximation by extending the results provided by Erlang [47] and Cox [48]. With these results it is possible to approximate any distribution with a distribution which can be represented by a network of exponential stages. Thus, our results of exponential service distributions can be extended to cover cases with other distributions, at least approximately.

5.0 ARRIVAL RATES

Here, we are going to determine the arrival rates Λ_{rj} for $1 \leq r \leq R$ and $0 \leq j \leq m$. We are going to term Λ_{rj} as arrival rates with the reservations mentioned at the end of Section 4.0, Chapter II. A simple model is to be employed to illustrate the method which is not

restricted to the configuration to be used, Figure 1.

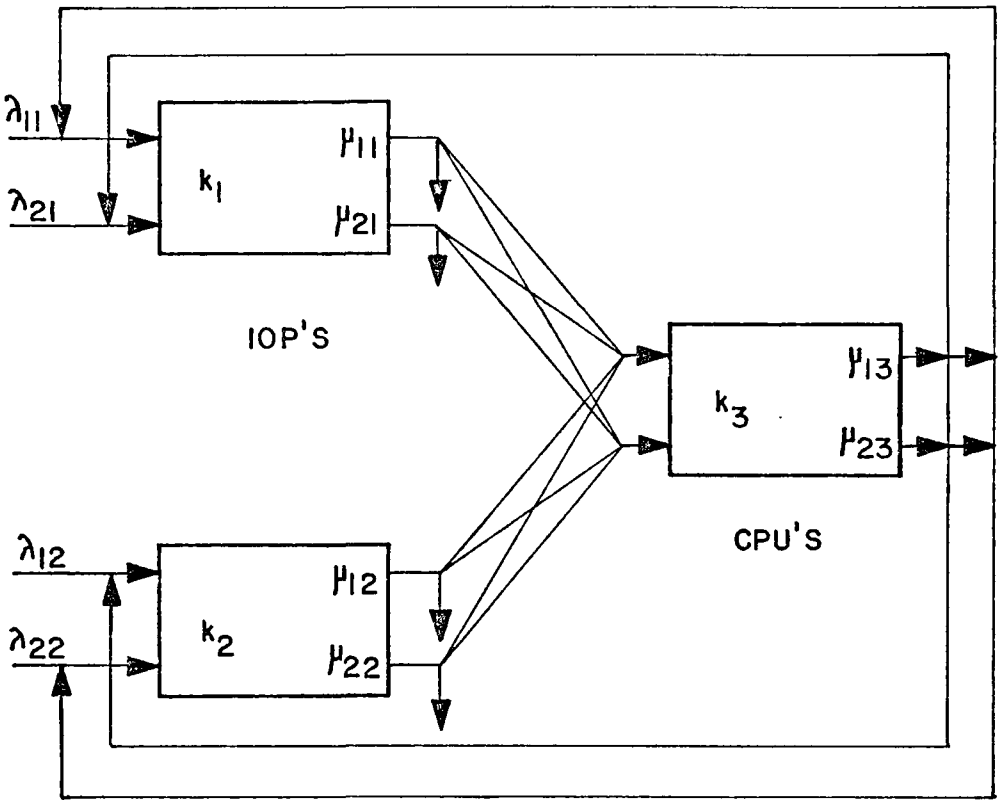


Figure 1 Configuration of the Model

Accordingly, the new arrivals come to the IOP stages (stages 1 and 2) only. The departures from the IOP stages either leave the system or proceed to the CPU stage (stage 0). The programs after leaving the CPU stage return to a particular IOP stage with a given probability.

First, we are going to determine arrival rates, Λ_{rj}, for $1 \leq r \leq 2$ and $0 \leq j \leq 2$. Λ_{rj}, for $1 \leq r \leq R$ and $0 \leq j \leq m$, are given by the approximation (18). Rewriting (18)

in open form and taking the configuration of the system into account we have for $1 \leq r \leq 2$ and $0 \leq j \leq 2$

$$\begin{aligned}
\Lambda_{1,0} &= p(1,1;1,0)\Lambda_{1,1} + p(2,1;1,0)\Lambda_{2,1} \\
&\quad + p(1,2;1,0)\Lambda_{1,2} + p(2,2;1,0)\Lambda_{2,2} \\
\Lambda_{2,0} &= p(1,1;2,0)\Lambda_{1,1} + p(2,1;2,0)\Lambda_{2,1} \\
&\quad + p(1,2;2,0)\Lambda_{1,2} + p(2,2;2,0)\Lambda_{2,2} \qquad (33) \\
\Lambda_{1,1} &= \lambda_{1,1} + p(1,0;1,1)\Lambda_{1,0} + p(2,0;1,1)\Lambda_{2,0} \\
\Lambda_{2,1} &= \lambda_{2,1} + p(1,0;2,1)\Lambda_{1,0} + p(2,0;2,1)\Lambda_{2,0} \\
\Lambda_{1,2} &= \lambda_{1,2} + p(1,0;1,2)\Lambda_{1,0} + p(2,0;1,2)\Lambda_{2,0} \\
\Lambda_{2,2} &= \lambda_{2,2} + p(1,0;2,2)\Lambda_{1,0} + p(2,0;2,2)\Lambda_{2,0}
\end{aligned}$$

From (33) we have,

$$\begin{aligned}
\Lambda_{1,0} &= \Lambda_{1,0} \sum_i \sum_k p(i,k;1,0)p(1,0;i,k) \\
&\quad + \Lambda_{2,0} \sum_i \sum_k p(i,k;1,0)p(2,0;i,k) \qquad (34) \\
&\quad + \sum_i \sum_k p(i,k;1,0)\lambda_{ik} \\
\Lambda_{2,0} &= \Lambda_{1,0} \sum_i \sum_k p(i,k;2,0)p(1,0;i,k)
\end{aligned}$$

$$+ \Lambda_{2,0} \sum_i \sum_k p(i,k;2,0)p(2,0;i,k)$$

$$+ \sum_i \sum_k p(i,k;2,0)\lambda_{ik} \tag{35}$$

Rewriting equations (34) and (35) as

$$(1-A)\Lambda_{1,0} - B\Lambda_{2,0} = Y \tag{36}$$

$$-C\,\Lambda_{1,0} + (1-D)\,\Lambda_{2,0} = Z \tag{37}$$

where

$$A = \sum_i \sum_k p(i,k;1,0)p(1,0;i,k) \tag{38}$$

$$B = \sum_i \sum_k p(i,k;1,0)p(2,0;i,k) \tag{39}$$

$$Y = \sum_i \sum_k p(i,k;1,0)\lambda_{ik} \tag{40}$$

$$C = \sum_i \sum_k p(i,k;2,0)p(1,0;i,k) \tag{41}$$

$$D = \sum_i \sum_k p(i,k;2,0)p(2,0;i,k) \tag{42}$$

$$Z = \sum_i \sum_k p(i,k;2,0)\lambda_{ik} \tag{43}$$

Then when $(1-A)(1-D) - CB \neq 0$ we have

$$\Lambda_{1,0} = \frac{Y(1-D) + BZ}{(1-A)(1-D) - CB} \tag{44}$$

$$\Lambda_{2,0} = \frac{Z(1-A) + CY}{(1-A)(1-D) - CB} \tag{45}$$

The rest of the arrival rates can then be determined through (33) by substituting (44) and (45). Although, Λ_{ij}'s are obtained here, for simplicity, for a system with two IOP stages and one CPU stage and only for two classes of programs, it can be repeated for more general systems through simple matrix algebra.

6.0 SYSTEM MEASURES OF MERIT

In this section we are going to derive some system measures of practical interest. These measures give us the means of quick comparison and insight into the system behaviour under the variation of the system parameters. It may be further utilized in an optimization to determine the most cost-effective system.

Let us first define a traffic intensity measure for stage i, ρ_i, i=0,...,m.

$$\rho_i \triangleq \sum_{r=1}^{R} \rho_{ri} \tag{46}$$

where

$$\rho_{ri} \triangleq \frac{\Lambda_{ri}}{k_i \mu_{ri}} \quad . \tag{47}$$

(a) The probability of finding the system empty in steady-state is given by,

$$P(0,\ldots,0) = E(N) \tag{48}$$

(b) The probability of finding z memory spaces occupied in the system, at steady-state is given by,

$$P\left(\sum_i |\underline{n}_i|^* = z\right) = E(N)\left[E^{-1}(z) - E^{-1}(z-1)\right] \tag{49}$$

where

$$0 \leq \sum_i |\underline{n}_i|^* \leq N \ .$$

(c) Then, the average number of memory spaces occupied in the system, Q_M, is given by

$$Q_M = \sum_{z=1}^{N} z\, P\left(\sum_i |\underline{n}_i|^* = z\right)$$

$$= E(N) \sum_{z=1}^{N} z\left[E^{-1}(z) - E^{-1}(z-1)\right] \tag{50}$$

(d) The average main memory utilization is given by the ratio Q/N.

(e) The probability of finding z programs in the system, at steady-state is given by,

$$P\left(\sum_i |\underline{n}_i| = z\right) = E(N) \sum_{\sum_i |\underline{n}_i| = z} \prod_{j=0}^{m} g_j(\underline{n}_j) \tag{51}$$

Let us denote the summation term in (51) as

$$E_p(z) \triangleq \sum_{\sum_i |\underline{n}_i| = z} \prod_{j=0}^{m} g_j(\underline{n}_j) \ . \tag{52}$$

(f) The average number of programs in system, Q_p, is given by (51) and (52) as

$$Q_p = E(N) \sum_{z=1}^{N} z \, E_p(z) \tag{53}$$

We can give similar expressions for each stage.

(g) The average number of memory spaces utilized by processor stage i, Q_{iM}, is given by,

$$Q_{iM} = \sum_{z=1}^{N} z \, P_i(|\underline{n}_i|^* = z) \tag{54}$$

$$= E(N) \sum_{z=1}^{N} z \, E_i^{-1}(N-z) \sum_{|\underline{n}_i|^* = z} g_i(\underline{n}_i) \tag{55}$$

for $0 \leq i \leq m$.

(h) The average number of programs that is found in processor stage i, Q_{iP}, is given by

$$Q_{iP} = \sum_{z=1}^{N} z \, P(|\underline{n}_i| = z) \tag{56}$$

$$= E(N) \sum_{|\underline{n}_i| = 1}^{N} z \, E_i^{-1}(N-|\underline{n}_i|^*) \, g(\underline{n}_i) \tag{57}$$

for $0 \leq i \leq m$.

(i) The steady-state probability of overflow (an arriving program can not enter the system because it is full), β, from (49) is given by,

$$\beta = P(\sum_i |\underline{n}_i|^* = N) \tag{58}$$

$$= 1 - \frac{E(N)}{E(N-1)} \quad . \tag{59}$$

(j) The average flow time between any two points in the system can be calculated from Little's result [59] by knowing the average queue length between these points and the average arrival rate to the subsystem involved.

Similar indicators for system performance can also be calculated for the closed systems. The expressions will be very similar to the ones obtained above.

7.0 EXAMPLE

Here, we are going to apply the results obtained to a numerical example. The system model we are going to consider is the same as Figure 1.

Let $k_0 = k_1 = k_2 = 1$, i.e. we have single processors at each stage.

Let the external arrival rates be given as follows:

$\lambda_{1,0} = 0.$ $\lambda_{1,1} = 2.$ $\lambda_{1,2} = 3.$

$\lambda_{2,0} = 0.$ $\lambda_{2,1} = 0.1$ $\lambda_{2,2} = 0.5$

Let the routing probabilities be given as follows:

$p(1,0;1,1) = 0.5$ $p(2,0;1,1) = 0.1$

$p(1,0;2,1) = 0.05$ $p(2,0;2,1) = 0.6$

$p(1,0;1,2) = 0.3$ $p(2,0;1,2) = 0.1$

$p(1,0;2,2) = 0.15$ $p(2,0;2,2) = 0.2$

$p(1,1;1,0) = 0.4$ $p(2,1;1,0) = 0.1$

$p(1,1;2,0) = 0.2$ $p(2,1;2,0) = 0.6$

$p(1,2;1,0) = 0.6$

$p(1,2;2,0) = 0.1$ $p(2,2;2,0) = 0.6$

otherwise $p(r,j;s,i)$ are equal to zero.

The above routing probabilities indicate that there is inter-class traffic.

The processing rates per class are taken as follows:

$\mu_{1,0} = 60.$ $\mu_{1,1} = 20.$ $\mu_{1,2} = 12.$

$\mu_{2,0} = 30.$ $\mu_{2,1} = 10.$ $\mu_{2,2} = 8.$

The results are tabulated, Table 1, for varying amounts of common storage space, N, in the system.

TABLE 1

Average arrival rates	$\Lambda_{1,0} = 5.54$	$\Lambda_{1,1} = 5.27$	$\Lambda_{1,2} = 5.16$
	$\Lambda_{2,0} = 4.99$	$\Lambda_{2,1} = 3.37$	$\Lambda_{2,2} = 2.33$
Traffic intensity	$\rho_{1,0} = 0.092$	$\rho_{1,1} = 0.264$	$\rho_{1,2} = 0.430$
	$\rho_{2,0} = 0.166$	$\rho_{2,1} = 0.337$	$\rho_{2,2} = 0.291$
	$\rho_0 = 0.258$	$\rho_1 = 0.601$	$\rho_2 = 0.721$

		N = 2	N = 6	N = 10
Average Queue Length per Stage, Q_i†	0	0.1676	0.3874	0.4903
	1	0.4020	1.1410	1.6641
	2	0.5083	1.4868	2.2732
Average total queue length, Q†		1.078	3.015	4.428
Probability of finding system empty		0.3310	0.1397	0.1025
Probability of finding system full		0.4088	0.1295	0.0518
Average memory utilization		53.9%	50.3%	44.3%
Average flow time, W		0.4708	4.1588	15.2570

† Here, queues refer to the memory spaces occupied rather than the number of programs.

7.1 DISCUSSION OF THE EXAMPLE

Some of the comments we are going to make are common to the results obtained here and also in Chapter II.

The first step in the computation of the required probabilities is the determination of the arrival rates. When there is no interclass traffic, the computation is as straight forward as in single (Chapter II) or multiclass traffic. In the case where there is interclass traffic, we make use of the approximation given by equation [18]. To check the closeness of this approximation one can look at the probability of finding the system full. If this probability is much smaller than the other probabilities in the system then the results obtained through the approximation should be very close to the actual values. In the cases when this approximation is not good enough, we have to utilize the numerical method given in the appendix. We should point out here that the numerical approach given in the appendix does not have to make use of the special dependency relation of the routing probabilities on the states, therefore, it is of a more general nature. As long as this dependency relation between the states of the system and the routing probabilities can be expressed explicitly then the problem can be treated numerically.

In the computation of most of the system measures, the normalization constant E(N) is necessary; but in computing

E(N), one may employ the recursive nature of the expression and also at the same time through the intermediate values, obtain other terms necessary in the computations. This can be seen from equations [49, 50, 59] and the following relations, from (13) we write

$$E(X) = S^{-1}(X). \tag{60}$$

Define

$$D(Z) = \sum_{\sum_{j=0} |\underline{n}_j| = Z}^{*} \sum_{i=0}^{m} g_i(\underline{n}_i) \tag{61}$$

then

$$S(1) = 1 + D(1)$$

$$S(2) = S(1) + D(2)$$

$$\vdots$$

$$S(Z) = S(Z-1) + D(Z). \tag{62}$$

The system behaves as expected, as the memory space, N, is increased the queue lengths become longer before each processor stage, thus keeping each stage proportionally busy for a longer time. The probability of finding the system empty decreases with increasing N and would eventually level out asymptotically. The probability of finding the system full (overflow conditions) decreases with increasing N as expected, which is also true for the utilization of the main memory. Here, we see that the average flow time of a single

page, that is, the average time a page stays busy in the system, increases with N. This is due to the increased number of programs waiting in front of each processor stage, therefore, it takes more time to go through each processor stage.

CHAPTER IV

GENERALIZATION OF THE RESULTS AND AREAS FOR FURTHER RESEARCH

1.0 GENERAL

In this chapter we are going to generalize the results presented in Chapters II and III, to more general networks which are applicable as models in other fields in addition to the multiprogramming computer systems.

The other application fields of the network of queues model can be found in the "job shop-like" systems [40, 41], in the transportation systems [44, 45], in the mining industry [31, 44, 45], in the storage (inventory or dam) systems and in the communication (computer, message, etc.) networks [1, 60 - 62]. The identifying characteristics of the models common to all the above fields is that discrete units (customers, programs, loads or packets) arrive at instants governed by a stochastic process to a service center. A unit after arrival to a service center joins the queue there and gets served when its turn comes and may or may not proceed to the next service center depending on its requirements (routing).

The units the system processes may consist of several classes depending on their service and space requirements.

An example of this can be seen in computer communication networks. Let us suppose that messages sent from one computer (or terminal) to the other is divided into small "packets" for the convenience of processing and transmission through the network. Each packet is transmitted independently over the network to its destination, sometimes each utilizing different paths. Each of the packets may or may not need the same amount of processing at each node (an intermediate processing point) of the network that it goes through in its journey to its destination. Each packet requires a storage space in the system, and there may be different sized packets in the system. Although it is difficult to visualize a common storage space for an entire computer communication network, there are proposals, [61, 62], to establish a fixed maximum amount of active packets in the system to control congestion.

Our models in this work are assumed to have a storage space common to all service centers. The arrival process for each class is governed by a time-homogeneous Poisson process and the service lengths for each class is distributed exponentially, which is mutually independent of other classes and the arrival processes.

2.0 MATHEMATICAL MODEL

Here, we are going to use the notation given in Chapters II and III.

We assume, there are m+1 service centers and R classes of units (jobs) offered to the system. The service center i has $k_i \geq 1$ identical servers (processors), $0 \leq i \leq m$.

The new units belonging to class r are assumed, in the open system, to arrive to the ith service center in the form of time-homogeneous Poisson processes with the average arrival rate $\lambda_{ri} \geq 0$ for $1 \leq r \leq R$ and $0 \leq i \leq m$.

We assume that a job from class i requires i units of storage space in the system. Let there be N units (measured in some convenient, discrete units) of common storage space available in the system. When a new job from class j arrives and finds the storage has less than j units of free space, it leaves the system, never to return. When a job departs from the system it releases the space it occupies.

After completion of service at a service center, the storage space requirements may change, if the total allowable space permits such a change. Otherwise, a job stays with the same class in the next center on its routing. Accordingly, we have the routing probabilities as defined in Chapter III, Section 2.2.

The service length distribution of class r units in the service center j is assumed to be a negative exponential, mutually independently distributed to the other classes,

stages and arrival processes, and has a mean instantaneous service completion rate.

$$\mu'_{rj}(\underline{n}_j) = \mu_{rj} n_{rj} \phi_{rj}(\underline{n}_j) \tag{1}$$

where $\underline{n}_j = (n_{1j},\ldots,n_{Rj})$ denotes the presence of n_{rj} number of class r units in service center j, and $\mu_{rj} > 0$ is a constant, for $1 \leq r \leq R$, $0 \leq j \leq m$. Here,

$$\phi_{rj} = \begin{cases} 1 & \text{for } |\underline{n}_j| \leq k_j \\ k_j/|\underline{n}_j| & \text{for } |\underline{n}_j| \geq k_j \end{cases} \tag{2}$$

and $|\underline{n}_j| \triangleq \sum_r n_{rj}$.

The properties of the service discipline specified by (2) have been elaborated Chapter III, Section 2.1.

3.0 SYSTEM EQUATION

The equation of the model described above can be formed following the procedure given in Section 3.0 of Chapter III. We are not going to repeat some of the intermediate steps here, as they have been explained in the previous chapters. Then, it can be shown that the steady-state equation of a generalized model is given by,

$$\left[\sum_r \sum_j C(0,r)\lambda_{rj} + \sum_r \sum_j \mu'_{rj}(\underline{n}_j)\right] P(\underline{n}_0,\ldots,\underline{n}_m)$$

$$= \sum_r \sum_j \varepsilon(n_{rj})\lambda_{rj} P(\underline{n}_0,\ldots,(\underline{n}_j)_{r-},\underline{n}_{j+1},\ldots,\underline{n}_m)$$

$$+ \sum_r \sum_i \sum_s \sum_j \mu'_{ri}(\underline{n}_i)_{r+} P(r,i;s,j)\varepsilon(n_{sj})$$

$$.P(\underline{n}_0,\ldots,(\underline{n}_i)_{r+},\underline{n}_{i+1},\ldots,(\underline{n}_{j+1},\ldots,\underline{n}_m). \qquad (3)$$

$$+ \sum_r \sum_j C(0,r)\mu'_{rj}(\underline{n}_j)_{r+} p^*_{rj}\ P(\underline{n}_0,\ldots,(\underline{n}_j)_{r+},\underline{n}_{j+1},\ldots,\underline{n}_m)$$

In a closed system, where there are no arrivals and departures, the steady-state equation is reduced to

$$\sum_r \sum_j \mu'_{rj}(\underline{n}_j)\ P(\underline{n}_0,\ldots,\underline{n}_m)$$

$$= \sum_r \sum_i \sum_s \sum_j \mu'_{ri}(\underline{n}_i)_{r+} p(r,i;s,j)\varepsilon(n_{sj}) \qquad (4)$$

$$.P(\underline{n}_0,\ldots,(\underline{n}_i)_{r+},n_{i+1},\ldots,(\underline{n}_j)_{s-},\underline{n}_{j+1},\ldots,\underline{n}_m)\ .$$

The solution of equations (3) and (4) is given by Theorem 1 in Chapter III. For the other processing disciplines the rest of the results of Chapter III follows.

Thus, we have shown that results obtained in Chapters II and III are also applicable to networks with arbitrary configuration.

Next, we are going to study systems with service centers having different service disciplines in the same network.

4.0 DIFFERENT SERVICE DISCIPLINES AT EACH CENTER

We are going to limit our discussion here, to a network of queues with two service centers. Having to consider only two centers, neither decreases our insight into the matter nor do we loose any generality, it only reduces the tediousness of the algebra.

Let us consider that service center 1 is governed by a processor sharing service discipline. Thus,

$$\mu'_{ri}(\underline{n}_1) = \mu_{r1} \frac{n_{r1}}{|\underline{n}_1|} \tag{5}$$

for $1 \leq r \leq R$

where

$$|\underline{n}_1| = \sum_{r=1}^{R} n_r \ .$$

And let service center 2 be governed by a service discipline as was described in Section 3.0 by equation (1).

Then, by specialization of equation (5) we have,

$$\begin{aligned} &\sum_r [C(0,r)(\lambda_{r1}+\lambda_{r2}) + \mu_{r1} \frac{n_{r1}}{|\underline{n}_1|} \\ &+ \mu_{r2} n_{r2} \phi_{r2}(\underline{n}_2)] P(\underline{n}_1, \underline{n}_2) = \sum_r \varepsilon(n_{r1}) \lambda_{r1} P((\underline{n}_1)_{r-}, \underline{n}_2) \\ &\qquad + \sum_r \varepsilon(n_{r2}) \lambda_{r2} P(\underline{n}_1, (\underline{n}_2)_{r-}) \\ &\qquad + \sum_r \sum_s \mu_{r1} \frac{n_{r1}+1}{|\underline{n}_1|+1} p'(r,1;s,2) \end{aligned} \tag{6}$$

$$\cdot \varepsilon(n_{s2}) P((\underline{n}_1)_{r+}, (\underline{n}_2)_{s-})$$

$$+ \sum_r \sum_s \mu_{r2}(n_{r2}+1)\phi_{r2}(\underline{n}_2)_{r+} p'(r,2;s,1)$$

$$\cdot \varepsilon(n_{s1}) P((\underline{n}_1)_{s-}, (\underline{n}_2)_{r+})$$

$$+ \sum_r C(0,r)\mu_{r1} \frac{n_{r1}+1}{|\underline{n}_1|+1} p^*_{r1} P((\underline{n}_1)_{r+}, \underline{n}_2)$$

$$+ \sum_r C(0,r)\mu_{r2}(n_{r2}+1)\phi_{r2}(\underline{n}_2)_{r+}$$

$$\cdot p^*_{r2} P(\underline{n}_1, (\underline{n}_2)_{r+}) \quad .$$

The solution of the equation is given,

$$P(\underline{n}_1, \underline{n}_2) = E(N) P_1(\underline{n}_1) \; P_2(\underline{n}_2) \tag{7}$$

where from equation (24) Chapter III,

$$P_1(\underline{n}_1) = |\underline{n}_1|! \prod_{r=1}^{R} \frac{1}{n_{r1}!} \left(\frac{\Lambda_{r1}}{\mu_{r1}} \right)^{n_{r1}} \tag{8}$$

and from equation (11) Chapter III,

$$P_2(\underline{n}_2) = \frac{1}{\phi_2(\underline{n}_2)} \prod_{r=1}^{R} (\Lambda_{r2})^{n_{r2}} \prod_{m=1}^{n_{r2}} \frac{1}{\mu_{r2^m}} \tag{9}$$

Here,

$$\phi_2(\underline{n}_2) = \begin{cases} 1 & \text{for } |\underline{n}_2| \leq k_2 \\ \dfrac{k_2!\,(k_2)^{\,|\underline{n}_2|-k_2}}{|\underline{n}_2|!} & \text{for } |\underline{n}_2| \geq k_2 \end{cases} \tag{10}$$

.

Normalization constant E(N) is to be determined from

$$E^{-1}(N) = \sum_{0 \leq \sum_{j=1}^{2} |\underline{n}_j|^* \leq N} P_1(\underline{n}_1)\, P_2(\underline{n}_2) \qquad (11)$$

where

$$|\underline{n}_i|^* = \sum_{r=1}^{R} r n_{ri} \quad \text{and} \quad |\underline{n}_i| = \sum_{r} n_{ri} \ .$$

The solution is easily verified by direct substitution of equations (7 - 10) into (6).

By the above treatment, it is demonstrated that our results hold for systems which have different service disciplines (provided that they belong to the families of disciplines described in Chapter III) at different service centers.

5.0 CONCLUSION AND AREAS OF FUTURE RESEARCH

In this work, our aim was to provide a comprehensive model for the multiprogramming computer system from the resource sharing point of view. The models developed interrelate the arrival process, the heterogeneous processors, the finite main memory size and several classes of programs that are characterized by processing lengths and memory space requirements. In the models, the dynamic variation of the processing lengths and the

memory space requirements of the programs in a paged main memory environment are allowed under various processing disciplines.

The modelling of multiprogramming computer systems was accomplished by first employing a single network of queues model with a simple class of programs to develop our tools of analysis. Then this method is extended to the case where there are several classes of programs which are characterized by their arrival rates, processing lengths and space requirements. The process and the space requirements of the programs allowed to change dynamically (after each process completion) during their life in the computer. In the case with several classes although a closed form solution has not been obtained an approximate solution, which is close to the actual solution in the light traffic, is given. In the appendix a numerical method to compute the exact values is outlined for all traffic situations.

In such systems, we examined the various queues forming in the common main memory space, in steady-state. This information yielded us the system behaviour under load, particularly the system bottlenecks.

Although our models assumed exponential service length distributions for each program, this assumption does not form a severe limitation because a large class of distributions can be approximated by a network of exponential stages (cf.[47]

and [48]). Our results do not change when the mean of the approximated distribution is the same as the mean of the exponential distribution used in our analysis. This has been demonstrated for restricted models by Baskett and Muntz, [38].

We have also shown that our results are applicable to other networks with arbitrary configurations which one may encounter in various other fields.

We have obtained practical system measures relating the behaviour of the system to the parameters (speed, size, quantitiy) chosen which may be employed in comparison of systems or cost-benefit optimization for a given application. Apart from this obvious extension there is a need for extending the theory to the networks where each service center has its own finite storage capacity.

Another area of future work is in the collection of data and measurement of traffic in the computer systems and networks. Starting with the knowledge of the behaviour of a network, the topic of congestion control may be explored. In this area, our work has direct relevance as was shown in [61, 62] where limitation of the maximum number of storage units [packets] in a network of computers was cited as one form of control mechanism.

The removal of Poisson assumption from the arrival processes, in the open networks, may increase the practicality of the theory; but difficulty is considerable.

We would like to point out that this type of analysis, though utilizing many simplifying assumptions, can be used as a starting point in the evaluation of complex systems. At least it can serve as a check to the more detailed simulations.

Finally, we quote Coffman and Denning [63] on the type of models of multiprogramming computer systems, which were extended in this study; "these results suggest that it is far more important for a model to reflect the true system structure rather than the detailed behaviour of the individual servers in the system".

APPENDIX

Here, we are going to outline a numerical method to solve the steady-state equation, (9) in Chapter III. The equation (10) and (14) in Chapter III give us a complete solution although not in explicit form in the case of state dependent routing probabilities. We rewrite equation (10) as follows:

$$P(\underline{n}') = F_1(\underline{n}', \underline{\Lambda}, \underline{\mu}, \underline{k}) \tag{1}$$

where

$$\underline{n}' = (\underline{n}_0,\ldots,\underline{n}_m) \text{ is an arbitrary but fixed vector} \tag{2}$$

$$\underline{\Lambda} = (\Lambda_{10},\ldots,\Lambda_{R0},\Lambda_{11},\ldots,\Lambda_{Rm}) \tag{3}$$

$$\underline{\mu} = (\mu_{10},\ldots,\mu_{R0},\mu_{11},\ldots,\mu_{Rm}) \tag{4}$$

$$\underline{k} = (k_1,\ldots,k_m) \ .$$

Now from equation (14) in Chapter III we observe that $\underline{\Lambda}$ is a function of $\underline{\lambda}$ and $\underline{p}'$ where

$$\underline{\lambda} = (\lambda_{10},\ldots,\lambda_{R0},\lambda_{11},\ldots,\lambda_{Rm}) \tag{5}$$

$$\underline{p}' = (p'(1,0;1,0),\ldots,p'(1,0;R,m),\ldots,p'(R,m;R,m)) \tag{6}$$

Then

$$P(\underline{n}') = F_1(\underline{n}', \underline{\lambda}, \underline{p}', \underline{\mu}, \underline{k}) \ . \tag{7}$$

In the case where $\underline{p}'$ is state dependent it is the function of $\underline{p}$ (constant routing probabilities) and $\underline{P}$ $(\underline{n})$ (all the permissible state vectors). Then

$$P(\underline{n}') = F_1(\underline{n}', \underline{\lambda}, \underline{p}, \underline{P}(\underline{n}), \underline{\mu}, \underline{k}) \tag{8}$$

as for a given system we can consider the parameters of the system as constants and we can write functionally

$$P(\underline{n}') = F_2(\underline{P}(\underline{n})) \ . \tag{9}$$

For all state vectors the left hand side of (9) becomes $\underline{P}(\underline{n})$ and we have

$$\underline{P}(\underline{n}) = F_2(\underline{P}(\underline{n})) \tag{10}$$

or for simplicity we write

$$\underline{P} = F_2(\underline{P}) \ . \tag{11}$$

In the system of equations (11) to solve for $\underline{P}$ we write it in the following form:

$$\Gamma = \underline{P} - F_2(\underline{P}) = 0 \tag{12}$$

Equation (12) can be solved by well-known Newton-Raphson method (64) in an iterative way. Let $\underline{P}^{(i)}$ denote the value of ith iteration, the algorithm to find $\underline{P}$ is given by

$$\underline{P}^{(i+1)} = \underline{P}^{(i)} - J^{-1}(\underline{P}^{(i)})\Gamma(\underline{P}^{(i)}) \tag{13}$$

$J^{-1}(x)$ is the inverse of the Jacobian computed at x.

The conditions for the convergence of (13) are given as follows:

$$1 - |\Gamma_i(\underline{P}^{(0)})| \leq A \quad \text{for all } i \tag{14}$$

where Γ_i denotes the ith row of Γ and $\underline{P}^{(0)}$ is the initial point which may be obtained by neglecting the state dependency of p' as was done in the text.

2 - The Jacobian is given by the matrix $\left.\frac{\delta \Gamma_i}{\delta x_j}\right|_{\underline{P} = \underline{P}^{(0)}}$ where x_j denotes the jth component of the vector-valued function Γ_i. The Jacobian should have a non-vanishing determinant D (ith absolute value $|D|$) and the absolute value of its cofactor $|A_{ij}|$, then

$$\max_i \frac{1}{|D|} \sum_j |A_{ij}| \leq B \; . \tag{15}$$

3 - The elements of the Hessian matrix are

$$\frac{\delta^2 \Gamma_i}{\delta x_j \delta x_k} \leq C \qquad \text{for all i, j, k .} \tag{16}$$

Then it can be shown that for an initial point chosen in a region defined by A, B and C and also the dimensions of the system, the algorithm converges [64].

REFERENCES

1 - Computer Communication Networks, N. Abramson and F.F. Kuo (editors), Prentice-Hall, 1973, Chapter 2.

2 - Watson, R.W., Time-sharing System Design Concepts, McGraw-Hill, 1970, Chapter 2.

3 - Chang, W., "Single-server queueing process in computing systems", IBM Syst. J. No. 1, 1970, pp. 36-71.

4 - Kingman, J.F.C., "The heavy traffic approximation in the theory of queues", Proc. of the Symposium on Congestion Theory, University of North Carolina, Chapel Hill, 1964, pp. 137-169.

5 - Bhat, U.N., "Sixty years of queueing theory", Mang. Sci., Vol. 15, No. 6, Feb. 1966, pp. B-280-B-294.

6 - McKinney, J.M., "A survey of analytical time-sharing models", Comp. Surveys, V.1, June 1969, pp. 106-116.

7 - Kleinrock, L., "Certain analytical results for time-shared processors", Info. Processing 68, North Holland Pub. Co., Amsterdam, 1969.

8 - Hellerman, H., "Some principles of time-sharing schedular strategies", IBM Syst. J., Vol. 8, No. 2, 1969, pp. 94-117.

9 - Rasch, P.J., "A queueing theory study of RR scheduling of time-shared computer systems", J. ACM, Vol. 17, Jan. 1970, pp. 131-145.

10 - Coffman, E.G., Muntz, R.R., Trotter, H., "Waiting time distribution for processor sharing systems", J.ACM, Vol. 17, 1970, pp. 123-130.

11 - Sakata, M., Noguchi, S., Oizumi, J., "An analysis of the M/G/I queue under round-robin scheduling", Opns. Res. Vol. 19, March-April 1971, pp. 371-385.

12 - Bhat, N., Nance, R.E., "Busy period analysis of a time-sharing system modelled as a semi-Markov process", J.ACM, Vol. 18, April 1971, pp. 221-238.

13 - Adiri, I., "A dynamic time-sharing priority queue", J.ACM, Vol. 18, Oct. 1971, pp. 603-610.

14 - _____, and Avi-Itzhak, B., "A time-sharing queue", Manag. Sci., Vol. 18, 1969, pp. 639-657.

15 - Kleinrock, L., "Swap time considerations in time-shared systems", IEEE Trans. on Comp., Vol. C-19, June 1970, pp. 242-261.

16 - Heacox, H.C. and Purdon, P.W., "Analysis of two time-sharing queueing models", J.ACM, Vol. 19, June 1972, pp. 70-91.

17 - Nance, R.E., Bhat, U.N., and Claybrook, B.G., "Busy period analysis of a time-sharing system: Transform Inversion", J.ACM, Vol. 19, July 1972, pp. 453-463.

18 - Kleinrock, L. and Muntz, R.R., "Processor sharing queueing models of mixed scheduling disciplines for time-shared systems", (ibid), pp. 464-482.

19 - Belady, L.A., "A study of replacement algorithms for a virtual-storage computer", IBM Syst. J., Vol. 5, No. 2, 1966, pp. 78-101.

20 - Fine, G.H., McIsaac, P.V., and Jackson, C.W., "Dynamic program behaviour under paging", Proc. ACM 21st National Conf. 1966, pp. 223-228.

21 - Varian, L., and Coffman, E.G., "An empirical study of the behaviour of programs in a paging environment", Proc. of ACM Symp. on Operating Principles, 1967.

22 - Denning, P.J., "The working set model for program behaviour", Comm. ACM, Vol. 11, No. 5, May 1968, pp. 323-333.

23 - Oden, P.H. and Shedler, G.S., "A model of memory contention in a paging machine", Comm. ACM, Vol. 15, No. 8, Aug. 1972, pp. 761-771.

24 - Coffman, E.G. Jr., and Ryan, T.A. Jr., "A study of storage partitioning using a mathematical mocel of locality", Comm. ACM, Vol. 15, No. 3, March 1972, pp. 185-190.

25 - Heller, J., "Sequencing aspects of multiprogramming", J.ACM, Vol. 8, 1961, pp. 426-439.

26 - Ochner, B.P., "Controlling a multiprocessor system", Bell Laboratories Record, Feb. 1966, pp. 59-62.

27 - Manacher, G.K., "Production stabilization of real time task schedules", J.ACM, Vol. 14, July 1967, pp. 439-465.

28 - Graham, R.L., "Bounds for certain multiprocessing anomalies", Bell Syst. Tech. J., Vol. 45, 1966, pp. 1563-1581.

29 - Muntz, R.R. and Coffman, E.G., "Optimal pre-emptive scheduling on two-processor systems", IEEE Trans. on Comp., Vol. C-18, No. 11, Nov. 1969, pp. 1014-1020.

30 - Ramamoorthy, C.V., Chandy, K.M., Gonzalez, M.J., "Optimal scheduling strategies in multiprocessor systems", IEEE Trans. on Comp., Vol. C-21, 1972, pp. 137-146.

31 - Koenigsberg, E., "The cyclic queues", Opns. Res. Quart., Vol. 9, March 1958, pp. 22-35.

32 - Kleinrock, L., "Sequential processing machines analyzed with queueing theory model", J.ACM, Vol. 13, April 1966, pp. 179-193.

33 - Sherr, A.L., An Analysis of Time-Shared Computer Systems, MIT Press, Cambridge, Mass., 1967.

34 - Gaver, D.P., "Probability models for multiprogramming computer systems", J.ACM, Vol. 14, July 1967, pp. 423-438.

35 - Mitrani, I., "Non-priority multiprogramming system under heavy demand conditions", J.ACM, Vol. 19, No. 3, July 1972, pp. 445-452.

36 - Buzen, J., "Analysis of system bottlenecks using queueing network model", ACM SIGOPS workshop 1971, Hardward, pp. 82-103.

37 - Baskett, F. and Gomez, F.P., "Processor sharing in a central server queueing model of multiprogramming

with applications", Proc. 6th Annual Princeton Conf. on Information Sciences and Systems, 1972, pp. 598-603.

38 - Baskett, F. and Muntz, R.R., "Queueing network models with different classes of customers", Proc. Compcon 72, San Francisco, pp. 205-209.

39 - _____ "Network of queues", Proc. 7th Annual Princeton Conf. on Information Sciences and Systems, 1973.

40 - Jackson, J.R., "Networks of waiting lines", Opns. Res., Vol. 5, 1957, pp. 518-521.

41 - _____ "Jobshop-like queueing systems", Mang. Sci., Vol. 10, 1963, pp. 131-142.

42 - Jackson, R.R.P., "Random queueing process with phase-type service", J.R. Statist. Soc. Series B, 1956, pp. 129-132.

43 - Finch, P.D., "Cyclic queues with feedback", J.R. Statist. Soc., Series B, 1959, pp. 153-157.

44 - Posner, M. and Bernholtz, B., "Closed finite queueing networks with time lags", Opns. Res., Vol. 16, 1968, pp. 962-976.

45 - _____ "Closed finite queueing networks with time lags and several classes of units", Opns. Res., Vol. 16, 1968, pp. 977-985.

46 - Chandy, K.M., "Analysis and solution for general queueing networks", Proc. 6th Annual Princeton Conf. on Information Sciences and Systems, 1972.

47 - Brockmeyer, E., Halström, H.L. and Jensen, A., "The life and works of A. K. Erlang", Transactions of the Danish Academy of Technical Sciences, No. 2, Copenhagen, 1948.

48 - Cox, D.R., "A use of complex probabilities in the theory of stochastic process", Proc. Cambridge Philosophical Soc., Vol. 51, 1955, pp. 313-319.

49 - Gordon, W.J. and Newell, G.F., "Closed queueing systems with exponential servers", Opns. Res., Vol. 15, 1967, pp. 254-265.

50 - _____ "Acknowledgement", Opns. Res., Vol. 15, 1967, p. 1182.

51 - Feller, W., An Introduction to Probability Theory and Its Applications, Vol. 1, John Wiley & Sons, 1968.

52 - Takács, L., Introduction to the Theory of Queues, Oxford University Press, 1962.

53 - Cox, D.R. and Miller, H.D., The Theory of Stochastic Processes, Methuen & Co., 1965.

54 - Coffman, E.G. and Wood, R.C., "Interarrival statistics for time sharing systems", Comm. of the ACM, Vol. 9, 1966, pp. 500-503

55 - Chang, W., "Queues with feedback for time-sharing computer systems", Opns. Res., Vol. 16, 1968, pp. 613-627.

56 - Coffman, E.G., "Analysis of two time sharing algorithms designed for limited swapping".

57 - Coffman, E.G. and Kleinrock, L., "Feedback queueing models for time-shared systems", J.ACM, Vol. 15, 1968, pp. 549-576.

58 - Fife, D.W., "The optimum control of queues, with applications to computer systems", Technical report No. 170, Cooley Electronics Lab., 1965.

59 - Little, J.D.C., "A proof for the queueing formula $L=\lambda W$", Opns. Res., Vol. 9, 1961, pp. 383-387.

60 - Kleinrock, L., Communication Nets; Stochastic Message Flow and Delay, McGraw-Hill, New York, 1964.

61 - Davies, D.W., "The control of congestion in packet-switching networks", IEEE Trans. on Com., Vol. C-20,

June 1972, pp. 546-550.

62 - Price, W.L., "Simulation of packet-switching networks on isarithmic principles", Proc. Third Data-communications Symposium, Datacomm 73, Tampa, Florida, 1973, pp. 44-49.

63 - Coffman, E.G. and Denning, P.J., "Operating Systems Theory", Prentice-Hall, 1973, Chapter 1.

64 - Saaty, T.L. and Bram, J., "Nonlinear Mathematics", McGraw-Hill, 1964, Chapter 2.

VITA

Mehmet Akın Sencer: Born in Turkey, 1940. Obtained his B.Sc. and M.Sc. degrees in Electrical Engineering from the Middle East Technical University in Ankara, Turkey in 1963 and 1964 respectively.

From 1965 to 1966 he worked with the R & D Laboratories of Ministry of Defence in Ankara on communication systems. He then joined the Defence Research Telecommunications Establishment in Ottawa, Canada on a one year research grant on signal processing. From 1967 to 1969 he was with Bell Canada Radio and Transmission Group. In 1969 he transfered to the Bell Northern Research establishment and worked on the traffic analysis of switching systems. In 1972 he became a full time student at University of Ottawa.

Since November 1973 he has been with Bell Canada Computer Communications Group as Supervising Engineer in Systems Planning.