ANALYSIS AND OPTIMIZATION OF MULTIPROGRAMMED COMPUTER SYSTEMS USING STORAGE HIERARCHIES

Ashby Woolf

University of Michigan

FOREWORD

The research described in this technical report was accomplished under Contract F30602-69-C-0214, Job Order Number 55810000 at the University of Michigan, Ann Arbor, MI. Mr. Rocco F. Iuorno (ISIS) was the Rome Air Development Center project engineer.

This report has been reviewed by the Office of Information (OI) and is releasable to the National Technical Information Service (NTIS).

This technical report has been reviewed and is approved.

Rocco F. Iuorno

Approved: ROCCO F. IUORNO
Project Engineer
Software Sciences Section

[illegible]

Approved: FRANK TOMAINI
Ch, Info Processing Branch
Info Sciences Division

ABSTRACT

The research described in this report centers around the development and application of a general and comprehensive mathematical model of computer systems which use storage hierarchies consisting of 2, 3 or more levels and in which the storage management is carried out automatically (i.e. transparent to the user.) This model has been implemented in the form of a highly interactive computer program which provides the user with an animated view of the system performance as model parameters are varied.

There are roughly 50 independent variables in the model. (The number of independent variables is dependent on the hardware configuration.) These variables describe the system architecture, data paths and routing, storage device characteristics, CPU performance, and user program behavior.

The overall model can best be described in terms of several component models. The storage device models take into account the effects of queueing for the device itself (such as a disk drive), queueing for channel service, seek time, latency time, and transfer time. The specific parameters may vary according to the specific device being modeled. Models for a core or random access device, drum, disk and data cell are described and implemented. The model which describes user program behavior is dependent on the storage allocated at each level in the hierarchy, the logical record or page size at each level, the size of the user program, and five other system independent variables. The user program model and the storage device models are linked together by a final model which relates user program behavior and the storage device performance. The macroscopic model includes effects of system architecture, data paths, queueing for the CPU, CPU lookahead and system software overhead.

The solution of the combined system model involves numerical techniques. A complete solution requires approximately 1 sec., based on IBM 360/67. When the model is used in the context of optimization, an evaluation can be obtained in approximately 50 ms. In practical terms this means that in the case of analysis the printing of results is more expensive than the analysis itself. In the case of optimization over 1000 systems can be examined in 1 minute or over 60,000 per hour.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

Chapter 1

Introduction

The design and application of large computing systems involves considerable risk. One can draw a parallel between those who have designed and applied today's computing systems and those who, in the early days of flight, strapped wings to their backs and jumped from the roofs of barns. Neither had adequate methods of predicting system performance. The work reported here attempts to contribute to computer system technology through the development of improved methods of predicting system performance.

1.1 A General Discussion

Throughout the development of computing systems there has been an effort to make a given hardware technology perform better as a system. The primary emphasis in the early days of development was on hardware technology. However, as computing systems have progressed, there has been a continued and growing interest in the design of systems which make the best and most efficient use of a given hardware technology.

One important method of achieving efficient use of a given hardware technology has been the use of storage hierarchies. Storage hierarchies are not unique to computer systems. Your pocket, your desk or dresser drawer and your basement represent a storage hierarchy. The basic idea is simply to store those items used most frequently in an easily accessible location. Those items seldom used may be stored in less accessible and correspondingly less costly locations.

The storage hierarchy was first implemented in a computing system developed at the University of Manchester in England in 1949. As computer system designed have progressed, two things have happened.

First the complexity of the hierarchy itself has increased. We commonly see systems with 5, 6, or more different kinds of storage hardware capability, for instance registers, cores, drums, disks, data cells, magnetic tapes, punched cards, and punched paper tape. A second and very important change has occurred in the management responsibility of the hierarchy.

The gains achieved by the storage hierarchy have imposed considerable burden upon the programmer. At the outset the programmer was responsible for deciding what information should be stored on what device at each instant of time. In addition the programmer was responsible for carrying out the necessary operations required to move the information about as required. Lastly and very importantly, the programmer was responsible for keeping track of how and where everything was stored as it was moved about in the system. As we have progressed and hierarchies have become more complex the operating systems and hardware have assumed these responsibilities in varying degrees. In some cases all responsibilities for management of a hierarchy have been assumed by the operating system and/or hardware and the hierarchy is invisible to the programmer.

Another area of computer systems development has been in the use of multiprogramming. A computer is often limited by its slowest (or most overworked) component, often an IØ device. One of the methods used to alleviate this problem has been multiprogramming or the practice of working with more than one user program at a time.

One of the effects of multiprogramming is to provide the system with a more balanced workload. If a system processes one program at a time, it will find some programs using a great deal of CPU resource and leaving the IØ devices idle, while others leave the CPU idle and do voluminous IØ. On the other hand, if a system is processing 10 or 20 programs at once, it is unlikely that the system will see this group of programs exhibit the wide variations in resource demand exhibited by the individual programs in the group. In other words system loading is more consistent and predictable when the sample space grows larger.

The second effect of multiprogramming is to introduce parallel paths in the workload as seen by the computing system. If a computing system must follow a single thread of execution, a delay in any part of the system holds up the entire system. Multiprogramming is one way of providing the necessary parallel paths of execution which, if properly used, can increase resource utilization.

The difficulty with multiprogramming is complexity. The problems of resource scheduling are difficult. The problems of protecting one user from another, protecting the operating system itself, and charging for resource use become most complex. However, the rewards for efficient resource utilization can be great. There has been considerable motivation for complexity in the effort to better utilize computing system resources. This has been added to by the response requirements of time-sharing and real time systems.

This need for complexity has placed the modern computing system outside the capability of man's unaided intuition.

1.2 The Nature of the Problem

In very few words the problem is that of predicting and controlling the behavior of complex computing systems using storage hierarchies and multiprogramming.

A computing system using a storage hierarchy generally involves 2 or more different types of storage devices. Different device types have widely varying performance characteristics. Drums, cores and disks all behave differently under load. A system's performance may be determined by a complex balance of workload throughout the system of devices or may be determined by the performance characteristics of a single overworked device.

User programs can be big or small. They can access data in a serial or random manner. They can do large quantities of IØ or almost none at all.

The data paths and routing of information influence the load seen by storage devices. The logical record sizes throughout the system also influence the load seen by storage devices and in turn their response times.

The number of programs running in a multiprogrammed system affects the storage allocations of user programs. This in turn influences the demands placed on the system by user programs.

All of these factors and many others combine to create an enormously complex and remarkably difficult analysis problem. It is difficult to merely determine what a given system will and will not do in a given circumstance. It is even more difficult to design such systems especially if some kind of optimal or near optimal design is required.

1.3 Objectives

The specific objective of this research is to develop and demonstrate a mathematical model of a computing system. The computing systems in question here fall into the class of those systems using storage hierarchies of 2, 3 or more levels and multiprogramming. The model will exhibit the following characteristics:

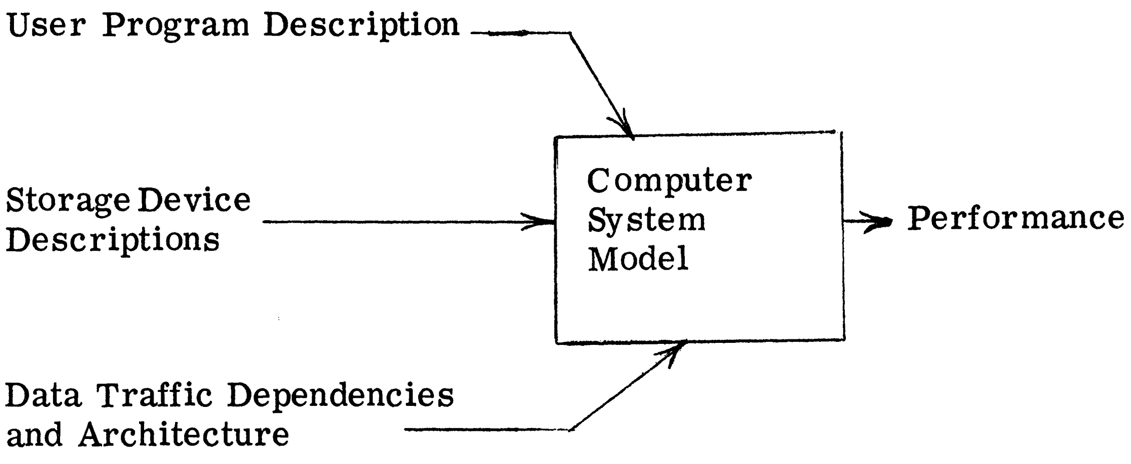
1. The model will include the effects of user program behavior, operating system characteristics and hardware performance.
2. The model will be versatile and easily applied to a wide range of system configurations. The model should be useful as a tool for the investigation of computing systems in general as well as applicable to the detailed investigation of a particular system.

3. The model will be useful as a tool for both analysis and optimization.
4. The results obtained from the model will approach the accuracy and realism of those obtained from simulation models.

1.4 A Preview

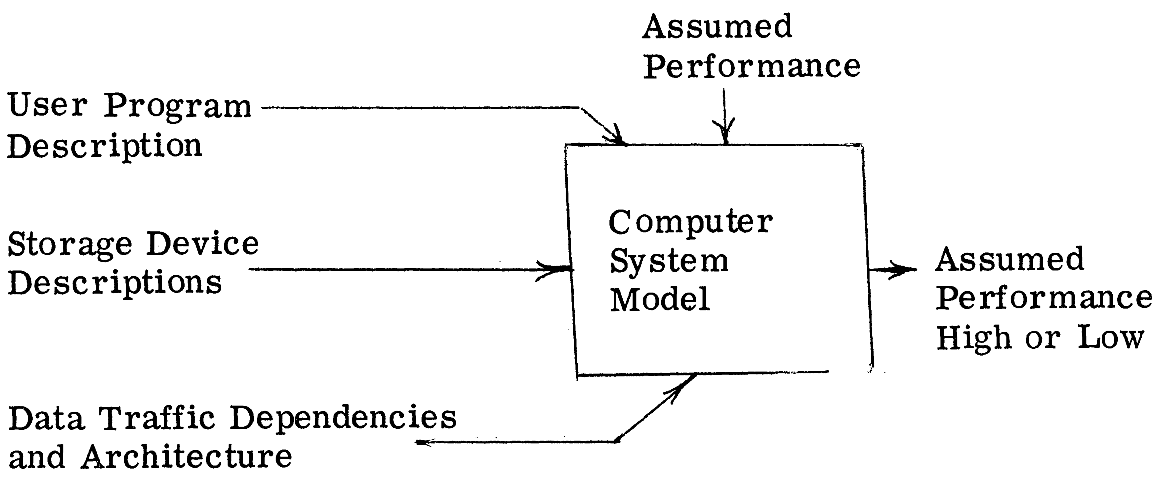
At this point we will attempt a broad preview of the coming chapters. It is hoped that this section will serve to place the contents of the chapters to follow in proper perspective and serve as a reader's guide.

The overall model for computer system analysis is shown in Figure 1.1. On the left we see 3 categories of independent variables. First we have the user program description. This includes independent variables such as the size of the user programs and other characteristics of user program behavior. Next is the storage device descriptions. This includes such items as drum RPM, disk seek time and variables relevant to storage device performance. Last we have data traffic dependencies and architecture. This includes CPU performance characteristics, logical record sizes, data transfer timing dependencies and other global system characteristics. As an output for the model we show performance. In the narrow sense, performance is defined as the mean rate at which the collection of user programs running on the system make reference or access to data and instructions. In the broad sense the performance also includes many


User Program Description
Storage Device Descriptions
Computer System Model
Performance
Data Traffic Dependencies and Architecture

System Analysis Model

Figure 1.1


Assumed Performance
User Program Description
Storage Device Descriptions
Computer System Model
Assumed Performance High or Low
Data Traffic Dependencies and Architecture

System Model

Figure 1.2

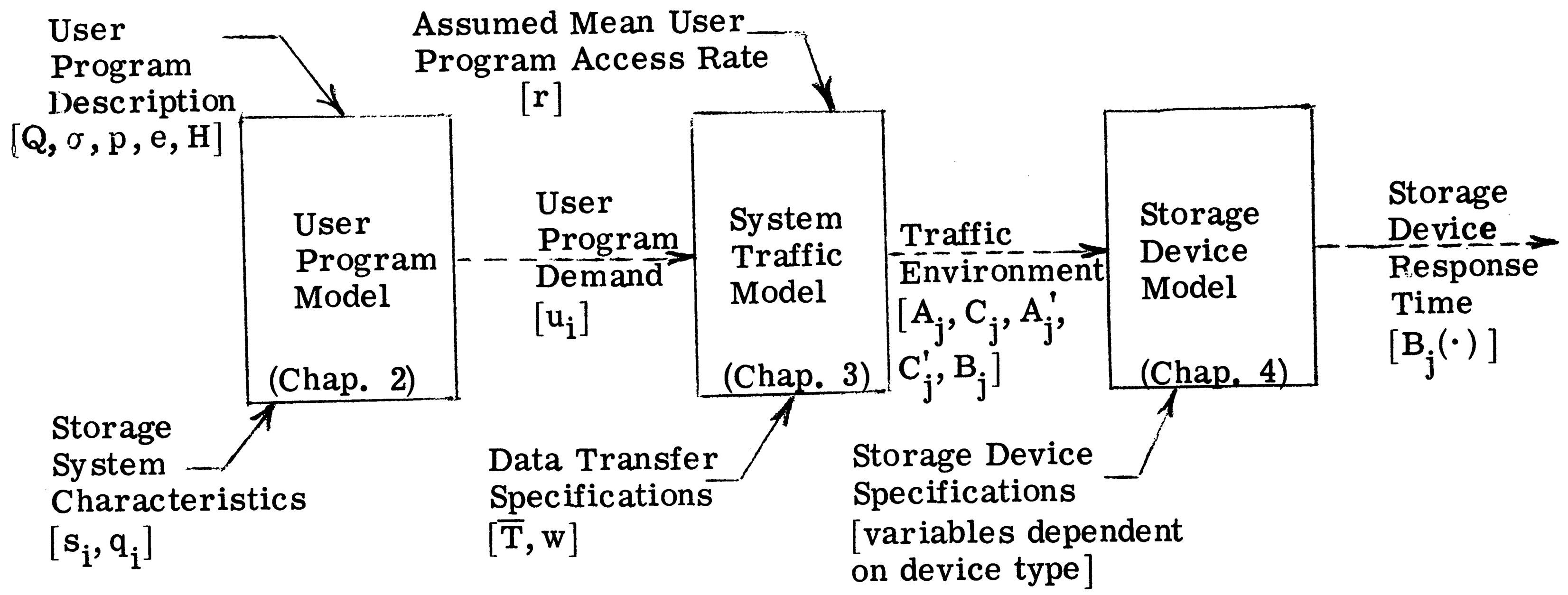
details such as mean queue lengths at various devices in the storage system and CPU utilization.

Figure 1.1 shows the model as seen by the designer when being used for analysis. Figure 1.2 shows the model in slightly greater detail and from a different point of view. It is this model that we will deal with in the next 3 chapters. We see that the independent variables on the left have not changed. However, the model shown here does not give us performance (mean user program access rate) directly but rather tells us if an assumed performance is greater than or less than a given system's capability. For analysis we will carry out a simple search to find the performance of a given system. The primary advantage of this particular approach to the problem occurs in optimization where systems are compared in a search for a system configuration with the greatest performance or mean user program access rate.

Figure 1.3 and 1.4 show a detailed breakdown of the model of Figure 1.2. Beginning at the left of Figure 1.3 we see the User Program Model. The independent variables supplied to this model fall into two classes, the user program description and the storage system characteristics. The user program description consists of 5 independent variables which describe the characteristics of the user program. The variables are shown in brackets here for reference in later chapters. The storage system characteristics consist of the storage allocation at each level in the hierarchy and the logical record size at each level. The user program model, given this information, determines what fraction of a user program's accesses will be directed to each level in the hierarchy of storage. This is referred to and shown in the figure as user program demand. For example, in a two level system with a core and a drum, this would be the fraction of user program references or accesses to storage which are satisfied at the core and the fraction which require drum activity.

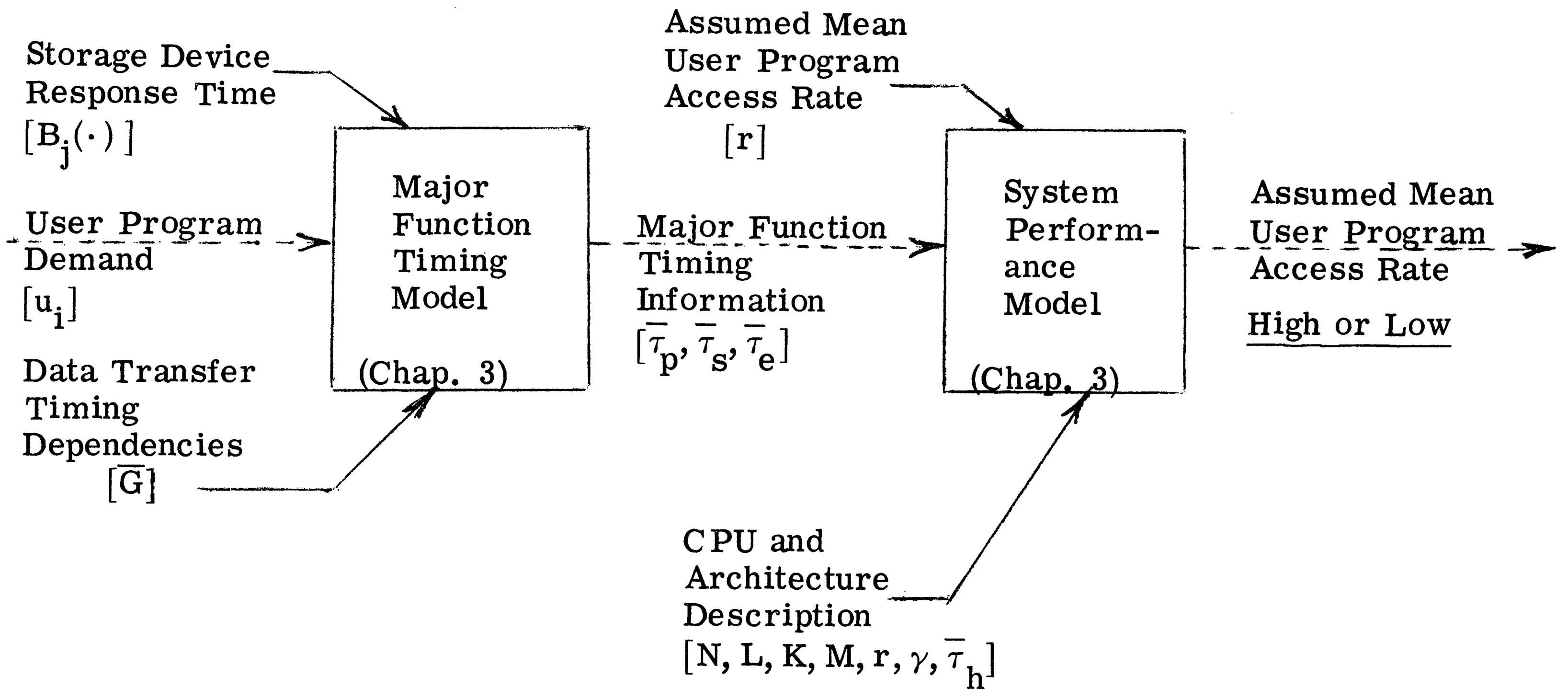
The user program model is discussed at length in chapter 2. The model is an outgrowth of the "lifetime function" concept proposed by Belady and Kuehner [6]. The model is also indebted to the "working set" concept introduced by Denning [21].

The next block in Figure 1.3 shows the System Traffic Model. This model has as its independent variables the assumed mean user program access rate (assumed performance), the user program demand and the data transfer specifications. The data transfer specifications describe what happens when a user program accesses a given level in the hierarchy. We are treating the user program access to a given level as a cause and the activity generated in the system as a result of that access as the effect. The effect takes the form of data transfers in the system. The data transfer specifications describe the data transfers which occur as a result of user program accesses to


User Program Description $[Q, \sigma, p, e, H]$
Assumed Mean User Program Access Rate $[r]$
User Program Model (Chap. 2)
User Program Demand $[u_i]$
System Traffic Model (Chap. 3)
Traffic Environment $[A_j, C_j, A'_j, C'_j, B_j]$
Storage Device Model (Chap. 4)
Storage Device Response Time $[B_j(\cdot)]$
Storage System Characteristics $[s_i, q_i]$
Data Transfer Specifications $[\overline{T}, w]$
Storage Device Specifications [variables dependent on device type]

Detailed System Model, Part 1

Figure 1.3



Storage Device Response Time $[B_j(\cdot)]$
User Program Demand $[u_i]$
Data Transfer Timing Dependencies $[\overline{G}]$
Major Function Timing Model (Chap. 3)
Major Function Timing Information $[\overline{\tau}_p, \overline{\tau}_s, \overline{\tau}_e]$
Assumed Mean User Program Access Rate $[r]$
System Perform-ance Model (Chap. 3)
CPU and Architecture Description $[N, L, K, M, r, \gamma, \overline{\tau}_h]$
Assumed Mean User Program Access Rate
High or Low

Detailed System Model, Part 2

Figure 1.4

some level in the hierarchy. Using this information along with the mean user program access rate and the fraction of user program accesses to each level (user program demand) the System Traffic Model generates a description of the traffic environment at each level in the hierarchy. The traffic environment is given in terms of the mean rate at which records are being read and written at each level, the mean record size being read and written at each level, and the maximum number of requests for service that can be generated by the system for each level. This System Traffic Model is discussed in detail in Chapter 3.

We arrive now at the Storage Device Model at the right hand side of Figure 1.3. The storage device model is actually a collection of models, any one of which can be used to represent the hardware at any level in the storage hierarchy. The device models give us the mean time required to read or write a record of a given size at a given level in the hierarchy. The model independent variables are the traffic environment seen by the level in question and device specification relevant to the device type at that level.

The storage device models are discussed at length in Chapter 4. Models for core or random access devices, drums, disks, and data cells have been developed. The models are not particularly unique or special in any real sense. They are based on finite Markov chains and require a numerical solution. The models were designed to provide generality of application, realistic results and rapid solution.

Backing away from the details of Figure 1.3, we can review what goes in and what comes out of this portion of the system model. We provide as independent variables a description of user program behavior and some storage system characteristics. From this we learn how the user program behaves in the system. Next, assuming a mean user program access rate and given the data trasfers which occur as a result of user program behavior, we compute the traffic flows at each level in the system. Given this and a description of the hardware at each level we compute the mean time required to read or write records of various sizes at each level. The principal result of the portion of the model shown in Figure 1.3 is simply the mean response time for each of the levels in the hierarchy.

We will now turn to Figure 1.4 where the second portion of the system model is shown. On the left we have the Major Function Timing Model which is discussed in detail in chapter 3. This model is responsible for timing information such as the mean time that a user program remains ineligible for execution following an access to some lower level of storage. The independent variables of this model are the storage device response times, the user program demand, and the data transfer timing dependencies. The data transfer timing dependencies require some explanation.

Imagine a system having a core and a drum. If a user program references some piece of data that happens to be on the drum a record or page will be read from the drum and written in core. There may also be a transfer from the core to the drum in order to make room for the new information coming into core. Here we see the possibility for 4 reads or writes. There is the operation of reading a record from the drum and writing that record in core and reading a record from core and writing it on the drum. However, in a typical system the only delay experienced by the executing program is that of reading the record from the drum. The other reads and writes do occur and do contribute to the congestion in the system but are generally carried out in such a way as to avoid a direct delay in execution. The data transfer timing dependencies specify precisely which reads and writes contribute directly to delays in program execution.

Finally on the right of Figure 1.4 we have the system performance model again covered in Chapter 3. This model has as its independent variables the assumed mean user program access rate, the major function timing information and certain items of a CPU and architecture description. This submodel determines if in fact the assumed mean user program access rate is too high or too low. This completes the model giving us the final result as shown in Figure 1.2.

This description of the model has been necessarily incomplete and simplified somewhat to aid in explanation. The detailed description of the various model components are found in Chapters 2, 3, and 4 as indicated. Chapter 5 is concerned with examples involving analysis and Chapter 6 with examples of optimization.

Chapters 2, 3, and 4 are long and concentrate on the detailed development of the various parts of the model described. The difficulties of placing the numerous but necessary details in proper perspective may be greatly reduced by becoming familiar with the examples of Chapters 5 and 6. Section 5.2 of Chapter 5 is of particular interest in this regard. A complete example is covered in Section 5.2 including some mention of all of the independent and dependent variables.

Chapter 2

A Model of Program Behavior

In an effort to identify and describe the characteristics of a computer program or process L. A. Belady and C. J. Kuehner [6] proposed the lifetime function. The lifetime function expresses a program's mean execution interval between references to secondary storage, as a function of the storage allocation in core or primary storage. Belady and Kuehner are not alone in attempting to model this aspect of a process. Very similar constructs have been considered by Peter J. Denning [21]. These efforts are at least in part attempting to determine what part of a program or process must be located in core to avoid an excessive number of transfers between core and drum.

2.1 A General Description of Program Behavior

It is generally agreed that most processes have a non-uniform storage access behavior and that it is meaningful to discuss the currently active part of a process. The concept of the currently active part of a process has been expressed in several different and useful ways. This idea is basic to the question of what part of a process should be in core.

P. J. Denning [21] refers to this currently active part as the working set. In Denning's words, "We define the working set of information $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval ($t-\tau, t$) ".

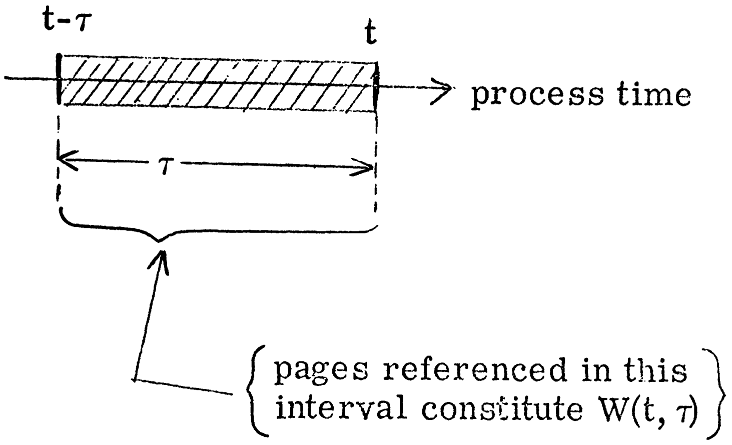


Figure 2.1 Definition of $W(t, \tau)$

There are many other ways to define a set of currently active pages. For instance we may modify the working set by considering t and τ of $W(t, \tau)$ to be real time instead of process time. We might consider a different parameterization such as the set $W'(t, n)$ defined as the n most recently used pages as a function of process time t. This is very similar in concept to the work by L. A. Belady and C. J. Kuehner [6]. Belady and Kuehner define the locality of storage references as a basic program property. "Locality is defined as the total range of storage references during a given execution interval." The remainder of the paper [6] if not the definition of locality would indicate that Belady's notion of locality is very similar to if not $W'(t, n)$.

The Two Level Hierarchy

The concept of a lifetime function was developed by Belady and Kuehner [6] in the context of a paging system consisting of a core and a drum. Here we will extend this concept in several ways but first we will consider it in a form close to Belady's original form as developed for two levels of storage.

When a program begins execution following a page fault (a reference requiring a page transfer from the drum) some of its pages will be in core and others generally will not. The precise information contained in core will determine the number of execution cycles required to generate another page fault. Since we are considering a paged system the information contained in core is determined by three factors:

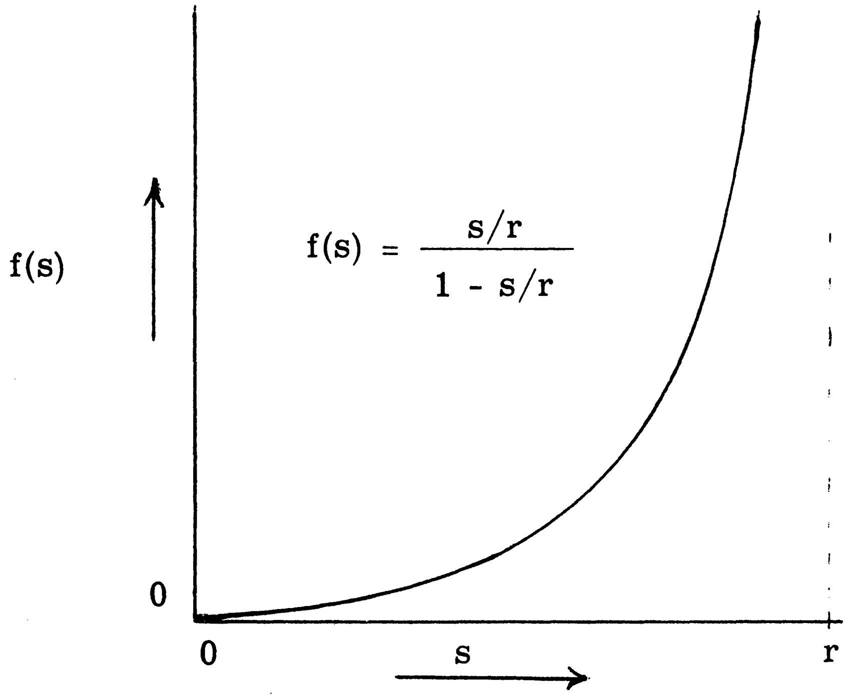
1. The method of selecting pages to be paged in and out.
2. The number of pages remaining in core.
3. The page size.

If we consider 1 and 3 to be fixed we may express the average number of execution cycles required to produce a page fault as a function of 2.

Belady defines his lifetime function as a relation between the size of the core allocation and the average length of an execution interval. In order to get a feel for the general shape of this function let us consider the simple case of completely random accesses. In this case the lifetime function is given in [6] as

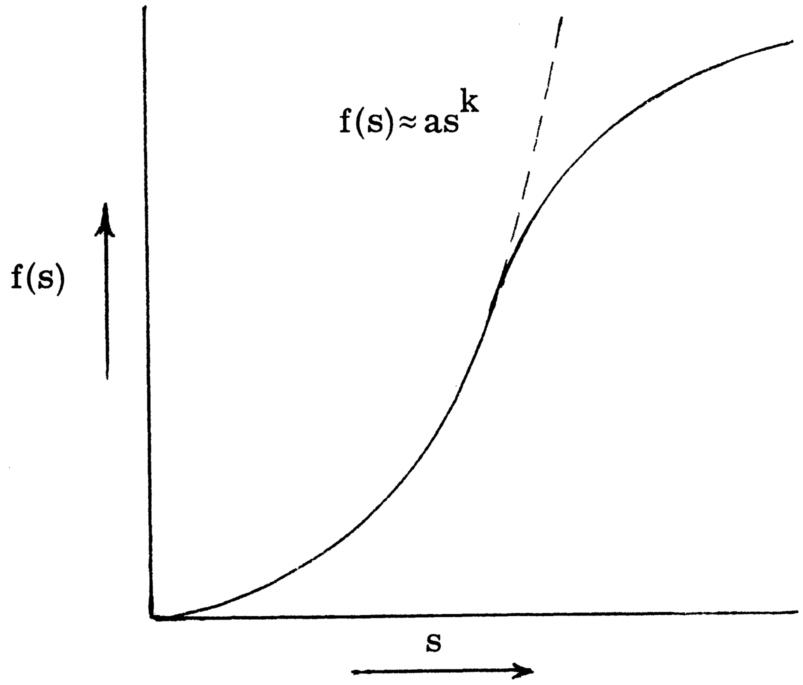
$$f(s) = \frac{s/r}{1 - s/r} = s/r + (s/r)^2 + (s/r)^3 + \ldots \qquad (2.1)$$

where r is the size of the program in bits and s is the core allocation in bits.



The lifetime function for Random Accesses

Figure 2. 2



General lifetime function

Figure 2. 3

This function is clearly convex and a curve of this form is shown in Figure 2.2. The general shape of a more realistic lifetime function is shown in Figure 2.3 with a solid line as well as an approximation proposed by Belady. Belady's approximating function is of the following form where a and k are constants to be adjusted to fit specific program behavior

$$f(s) = a\, s^{k} \qquad (2.2)$$

2.2 Basic Model Description

Belady's functional model for approximating program behavior has several deficiencies.

1. The model parameters a and k are not clearly related to properties of the program being modeled.
2. The model is valid only over part of the range of s for realistic programs.
3. The value of the lifetime function is expressed in units of execution time which limits its use in certain contexts.

In addition to removing the above deficiencies we would like to develop a model which considers the lifetime function as a function of both allocation, s, and page size, q.

We will refer to the model of program behavior developed here as the lifecycle function. This name reflects a change in units from execution time to the number of user program storage references required to produce a page fault.

conclusion by noting that with independent and equally probable random accessing, the specifics of what information is in core are irrelevant. Thus the lifecycle function is only a function of s and is independent of q.

As an aside we might consider what the optimal page size would be if programs did in fact behave in this manner. Since the lifecycle function is independent of q we have no motivation to make q larger than 1 word. A larger q and its associated information movement would be without effect on the lifecycle function and serve only to clutter channels and devices. This is of course an extreme case and has few representatives in the real world.

Now we will consider a second extreme case one which is the antithesis of the above. Consider a process which accesses storage sequentially from its first location to its last and then continues by repeating these accesses forming a large loop. Here accesses will be made proceeding sequentially through a page and into the next, producing a page fault. If the entire process is not in core (i. e. $s < q\lceil Q/q \rceil$ where $\lceil x \rceil$ is the smallest integer $\geq x$ and Q is the actual program size) the page needed will have always been paged out. This of course assumes a first in first out, least recently used or similar page out selection algorithm. Since we will produce a page fault at the end of each page we can write

$$f(s,q) = q \text{ for } s < q\lceil Q/q \rceil . \qquad (2.3)$$

Where Q is the actual program size $\lceil Q/q \rceil$ is the smallest integer $\geq$ Q/q and $q\lceil Q/q \rceil$ is the apparent program size as seen by a system with a page size q.

When $s \geq q\lceil Q/q \rceil$ we may write

$$f(s,q) = \sigma q \text{ for } s \geq q\lceil Q/q \rceil \tag{2.4}$$

where σ is the number of times the loop is executed.

Again we may stop to consider an optimal page size for processing such a process. It is clear that q should be large, in fact on the order of s, thereby refreshing the entire allocated storage with each page fault. Having considered these two extremes we will now turn to a more realistic model.

A Process Model

We will consider here a stochastic model of a process which is in some sense a combination of the two extremes just discussed. We may describe the access behavior as follows:

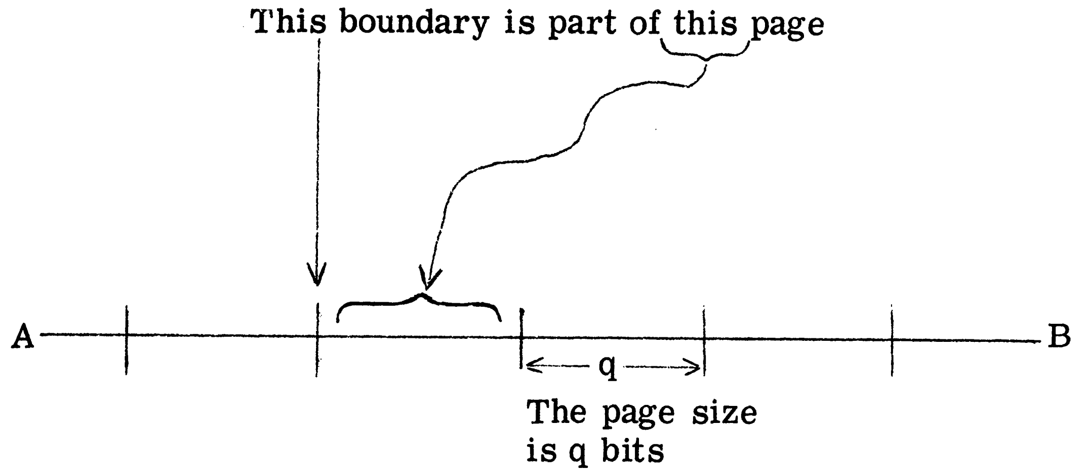
1. Randomly select a location α in the virtual address space of the process.
2. Randomly select a loop length Δ.
3. Access the Δ locations in the loop of 2 above sequentially σ times.
4. Repeat the above steps.

Where a loop which extends beyond the last virtual address of the process is continued at the beginning of the virtual address space.

Our pattern of access is then one of a series of loops where both the position and size of the loops change in a random fashion. Our immediate concern will be to develop the lifecycle function f(s, q) for such an access behavior.

2.3 Effects of Program Loop Lengths on Page Faults

The first step in developing this lifecycle function is to determine the number of distinct pages accessed when accessing the loop. This is a function of two things, the loop length and the position of the loop in virtual address space. The virtual address space can be represented as shown in Figure 2.4.

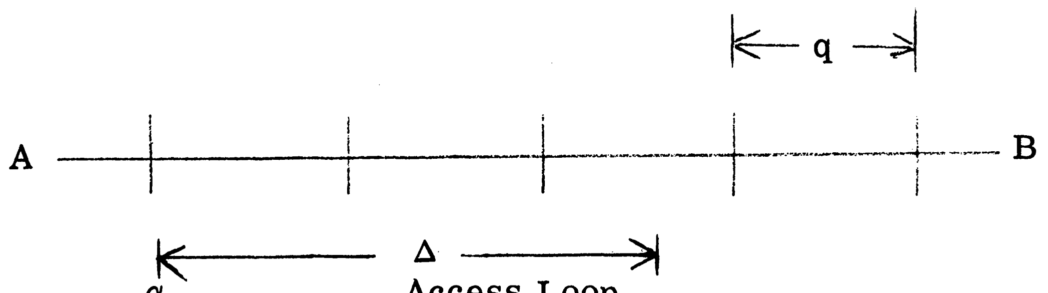


The Virtual Address Space

Figure 2.4

The line A-B in Figure 2.4 depicts a segment of virtual address space. The division lines represent page boundaries and the boundary is assumed to be part of the page to the right. The page size is q and is in bits.

We will now consider a loop of length Δ which begins on a page boundary at the beginning of a page.



Access Loop
Figure 2. 5

Here we may write the number of unique pages accessed in the loop as $\lceil \Delta/q \rceil$, where $\lceil x \rceil$ is defined as the smallest integer $\geq x$.

Examining Figure 2. 5 we can see that if we move the position of the loop α to the right the number of unique pages accessed will remain $\lceil \Delta/q \rceil$ until the right end of the loop crosses a page boundary. At this point the number of pages accessed will increase by 1 to $\lceil \Delta/q \rceil + 1$ and remain so until the left end of the loop crosses a page boundary. Thus the number of unique pages accessed by a loop of length Δ will be either $\lceil \Delta/q \rceil$ or $\lceil \Delta/q \rceil + 1$ depending upon the position of the loop in the virtual address space.

We will now consider what positions of the loop will involve $\lceil \Delta/q \rceil$ unique pages and what positions of the loop will involve $\lceil \Delta/q \rceil + 1$ unique pages. We will consider the position of the loop in virtual address space to be indicated by the starting address of the loop or the left end when diagrammed. We will first point out that if it is positioned or begins on any one of the first $\lceil \Delta/q \rceil q - \Delta$ bit positions a page $\lceil \Delta/q \rceil$ unique pages will be accessed by the loop.

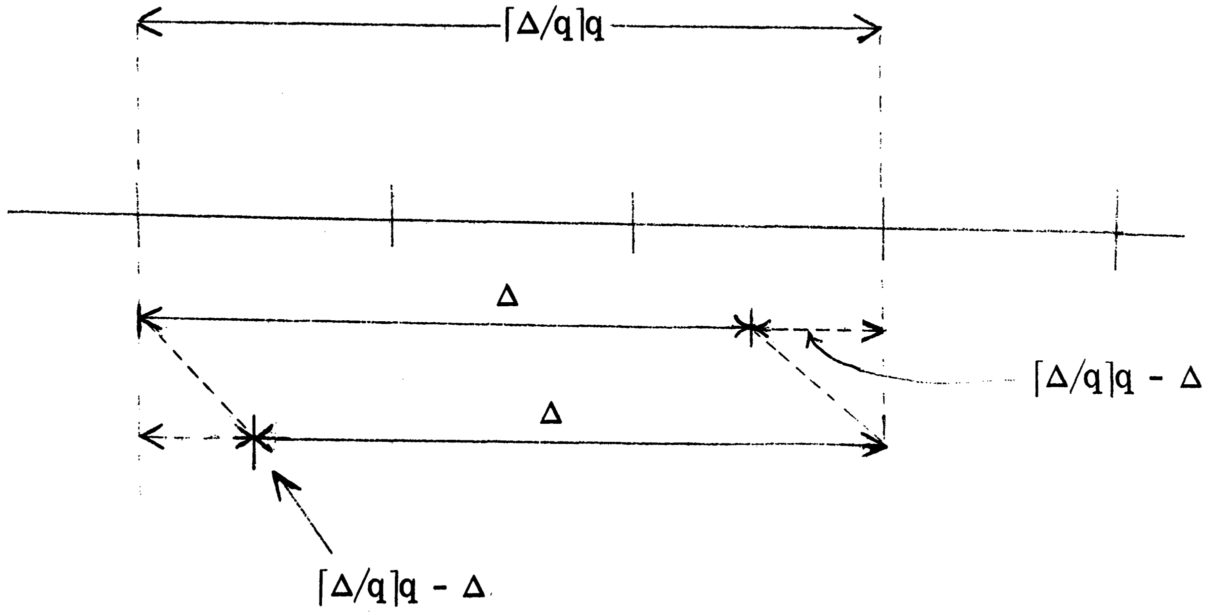


Figure 2.6 Loop Position

As shown in Figure 2.6 there are $\lceil \Delta/q \rceil q$ bits in the unique pages accessed by a loop of length Δ when the loop is positioned on a page boundary. Thus it is clear that in this case there are $\lceil \Delta/q \rceil q - \Delta$ bit positions not used by the loop. Thus one may position the loop on any one of the first $\lceil \Delta/q \rceil q - \Delta$ bit positions of any page and access only $\lceil \Delta/q \rceil$ unique pages. Correspondingly if the loop is positioned on any one of the $q - (\lceil \Delta/q \rceil q - \Delta)$ or $q - \lceil \Delta/q \rceil + \Delta$ bit positions of any page the loop will access $\lceil \Delta/q \rceil + 1$ unique pages.

We will consider α the position of each loop, to be a uniformly distributed random variable whose range is the entire virtual address space of the process. Thus we may write the probability of accessing any given number of unique pages in a loop as:

$$\begin{aligned} P(\text{accessing N unique pages}) &= \lceil \Delta/q \rceil - \Delta/q \qquad \text{for } N = \lceil \Delta/q \rceil \qquad (2.5) \\ &= 1 - \lceil \Delta/q \rceil + \Delta/q \qquad \text{for } N = \lceil \Delta/q \rceil + 1 \\ &= 0 \qquad \text{otherwise} \end{aligned}$$

Page Faults

Up to this point we have considered the number of unique pages accessed by a loop of length Δ in a system with a page size q. Here we will consider the number of page faults generated while accessing such a loop.

There are two important factors that must be taken into consideration here. We must determine if the number of unique pages accessed by the loop is greater than the space allocated s/q. If this is the case we will exhibit almost continuous paging. This is the simple result of the fact that as we access sequentially around the loop pages will be paged out before we complete the loop and access them again. Here one might say that the working set is larger than the allocated space.

A second condition occurs if the allocated space, s/q, is larger than or equal to the number of unique pages accessed in the loop. Here we will experience an initial burst of page faults as the pages associated with the loop are paged in and then a period of execution uninterrupted by additional page faults.

A second factor to be considered is that of finding needed pages in core (or some equivalent high level in the hierarchy) as a result of paging which occurred earlier. This means that as we begin each new loop of accesses we may find some of (and possibly all of) the needed pages resident in core.

Our immediate interest will be to find the ratio between the number of accesses made and the number of page faults which occur while accessing a loop Δ bits in length σ times. At least initially we will be interested in this ratio as a function of Δ the loop length. Thus we will define

$$g(\Delta) = \frac{\text{\# of accesses}}{\text{\# of page faults.}} \qquad (2.6)$$

In order to express $g(\Delta)$ it will be necessary to consider several special cases. We will begin by considering the two special cases

$$\begin{aligned} g(\Delta) &= g_1(\Delta) && \text{when } \alpha \text{ falls in the first } [\Delta/q]q - \Delta \\ &&& \text{bits of a page} \\ &= g_2(\Delta) && \text{when } \alpha \text{ falls in the last } q-[\Delta/q]q + \Delta \\ &&& \text{bits of a page} \qquad (2.7) \end{aligned}$$

where α is the virtual address position of the beginning of a loop.

The need for these two cases is generated by the different number of unique pages accessed in each case . In Case 1, for $g_1(\Delta)$, $\lceil\Delta/q\rceil$ and in Case 2, for $g_2(\Delta)$, $\lceil\Delta/q\rceil + 1$ unique pages are accessed. The development of $g_1(\Delta)$ and $g_2(\Delta)$ are quite similar with this difference of a single page being the only distinction. Thus we will develop $g_1(\Delta)$ in some detail and write $g_2(\Delta)$ by extending the same arguments.

An Expression for $g_1(\Delta)$

As we vary Δ we find that $g_1(\Delta)$ itself falls into two rather special cases. We will consider these two cases as Case 1a and Case 1b and write them as

$$\begin{aligned} g_1(\Delta) &= g_{1a}(\Delta) \qquad \text{for } \lceil \Delta/q \rceil \leq s/q \\ &= g_{1b}(\Delta) \qquad \text{for } \lceil \Delta/q \rceil > s/q \end{aligned} \tag{2.8}$$

To begin, notice that s/q is the number of pages allocated in core. In Case 1a the number of unique pages accessed is less than or equal to this allocation. Thus after the initial accessing of all the pages in the loop no further paging will be required. It should be clear that at most there will be $\lceil \Delta/q \rceil$ page faults during the accessing of this loop for after the loop has been accessed once all the required pages will be in core and repeated accesses in this loop will produce no further page faults.

Let us now consider Case 1b. Here $\lceil \Delta/q \rceil > s/q$, or more unique pages are accessed in the loop than the allocated space in core. This means that as we reach the end of the loop more pages will have been accessed than space has been allocated for. Assuming a least recently used, first in first out or similar page out selection scheme we may be sure that after accessing the first s/q unique pages in the loop a page fault will occur each time the sequence of accesses crosses a page boundary. This will continue as long as the loop is accessed. Since the loop will be accessed σ times a maximum of $\lceil \Delta/q \rceil \sigma$ page faults will occur.

2.4 Effects of Allocation During Initial Paging

Another factor to be considered is that of finding one or more of the $\lceil \Delta/q \rceil$ unique pages needed, in core. When the loop now being considered is begun there will be s/q pages in core as a result of one or more previous access loops. These pages may very well be some of those needed in the new loop and may remain in core (i.e. not be paged out) at the time of its first access in the new loop. It is clear at the outset that the number of page faults avoided in this manner is a random variable. It is also apparent that this random variable's mean and density function may be a function of all previous Δ's and α's. We will avoid these complications by making certain assumptions and replacing this random variable with its mean. Before we proceed with this, let us recall that our ultimate objective is not to accurately model the described process but to develop a function which may be used to approximate the lifetime functions of real programs. Thus we have a great deal of flexibility in the assumptions we may make as we proceed as long as the final result is suitable and useful for our purposes.

As we begin a new loop we will assume that the s/q pages remaining in core from previous accesses are scattered about randomly in the virtual address space of the process. Thus as we begin the new loop the

$$\text{probability of finding the first page in core} = \frac{s/q}{\lceil Q/q \rceil}$$

$$\text{probability of finding the second page in core} = \frac{s/q - 1}{\lceil Q/q \rceil - 1}$$

or in general

$$\text{probability of finding the ith page in core} = \frac{s/q - i+1}{\lceil Q/q \rceil - i+1} \qquad (2.9)$$

This only holds true for the first s/q pages or the number of pages accessed in the loop whichever is smaller. In the case of the first page accessed there are s/q pages in core and the process uses $\lceil Q/q \rceil$ pages. On each successive access the number of these left over pages is reduced by one either by the paging out process caused by a page fault or by its use and lack of availability. This of course assumes that $s < q\lceil Q/q \rceil$ for if $s \geq q\lceil Q/q \rceil$ no such paging out will occur and the probability of finding the i-th page in core will always be 1.

Thus we may write

$$\begin{aligned} \text{Prob}(i^{th} \text{ page in core}) &= \frac{s/q - i+1}{\lceil Q/q \rceil - i+1} & \text{for } s < q\lceil Q/q \rceil \\ &= 1 & \text{for } s \geq q\lceil Q/q \rceil \qquad (2.10) \end{aligned}$$

Unfortunately this expression of probability leads to excessive complications later when we consider a distribution of loop lengths, Δ, and integrate with respect to Δ. However we can successfully approximate using

$$\text{Prob}(i^{th}\text{ page in core}) = \begin{cases} \dfrac{s/q - i+1}{\lceil Q/q \rceil} & \text{for } s < q\lceil Q/q \rceil \\ = 1 & \text{for } s \geq q\lceil Q/q \rceil \end{cases} \qquad (2.11)$$

There are several reasons why this approximation serves us well. First it is reasonably accurate for small i and small i's predominate. The value of i cannot exceed s/q or $\lceil \Delta/q \rceil + 1$. This means that in order to be involved with large i's, we must have both a large allocation s and a long loop length Δ. Second we will be using a summation of these probabilities for $i = 1$ to $\lceil \Delta/q \rceil$ or in some cases $\lceil \Delta/q \rceil + 1$. The values of the terms in both Equation 2.10 and 2.11 decrease with increasing i. Thus the dominant terms in the series are those with small i for which the approximation is most accurate.

We may write the expected value of the number of such "free" pages which will occur with each new loop as

$$\sum_{i=1}^{K} \frac{s/q - i+1}{\lceil Q/q \rceil} = \frac{K(s/q + 1) - \sum_{i=1}^{K} i}{\lceil Q/q \rceil}$$

$$= \frac{K\,s/q + K - K(K+1)/2}{\lceil Q/q \rceil} \qquad (2.12)$$

for $s < q\lceil Q/q \rceil$

and

$$\sum_{i=1}^{K} 1 = K \qquad \text{for } s \geq q\lceil Q/q \rceil$$

where $K = \lceil \Delta/q \rceil$ in case 1a

and $K = s/q$ in case 1b

We will digress for a moment to insure that the source of the two values of K above is clear. In case 1a the number of pages referenced is less than or equal to the number allocated. In this case we must terminate the sum at the number of unique pages accessed which is $\lceil \Delta/q \rceil$. In case 1b the number of unique pages referenced is greater than the number allocated. The number allocated is s/q and we can only expect to find "free" pages during the first s/q unique page accessed.

Making these substitutions for case 1a and 1b:

Case 1a substituting $K = \lceil \Delta/q \rceil$

when $s < q\lceil Q/q \rceil$

$$\sum_{i=1}^{\lceil\Delta/q\rceil} \frac{s/q - i+1}{\lceil Q/q\rceil} = \frac{s/q\,\lceil\Delta/q\rceil + \lceil\Delta/q\rceil - \frac{1}{2}\lceil\Delta/q\rceil^2 - \frac{1}{2}\lceil\Delta/q\rceil}{\lceil Q/q\rceil}$$

$$= \frac{\lceil\Delta/q\rceil(s/q - \frac{1}{2}\lceil\Delta/q\rceil) + \frac{1}{2}\lceil\Delta/q\rceil}{\lceil Q/q\rceil} \qquad (2.13)$$

when $s \geq q\lceil Q/q\rceil$

$$\sum_{i=1}^{\lceil\Delta/q\rceil} 1 = \lceil\Delta/q\rceil \qquad (2.14)$$

Case 1b substituting $K = s/q$

when $s < q\lceil Q/q\rceil$

$$\sum_{i=1}^{s/q} \frac{s/q - i+1}{\lceil Q/q\rceil} = \frac{s/q\,(s/q + 1) - s/q\,(s/q + 1)/2}{\lceil Q/q\rceil}$$

$$= \frac{1}{2\lceil Q/q\rceil}\,(s^2/q^2 + s/q) \qquad (2.15)$$

Recalling that case 1b occurs when

$\lceil\Delta/q\rceil > s\lceil Q/q\rceil$, Equation (2.8), and that $\lceil\Delta/q\rceil \leq \lceil Q/q\rceil$

we see that in case 1b

$$s < q\lceil\Delta/q\rceil \leq q\lceil Q/q\rceil \qquad (2.16)$$

Thus in case 1b it is always the case that $s < q\lceil Q/q\rceil$

Here we will substitute the expected number of "free" pages for the actual number of the pages which is a random variable (except when $s \geq q\lceil Q/q \rceil$). By subtracting the number of "free" pages from the number of possible page faults we will obtain the number of actual page faults. There are several cases:

Case 1a:

$$\# \text{ of page faults (1a)} = \lceil \Delta/q \rceil - \frac{\lceil \Delta/q \rceil (s/q - \frac{1}{2} \lceil \Delta/q \rceil) + \frac{1}{2} \lceil \Delta/q \rceil}{\lceil Q/q \rceil}$$

$$= \lceil \Delta/q \rceil \left[1 - \frac{s}{q\lceil Q/q \rceil} - \frac{1}{2\lceil Q/q \rceil} + \frac{1}{2\lceil Q/q \rceil} \lceil \Delta/q \rceil \right] \qquad \text{for } s < q\lceil Q/q \rceil$$

$$= \lceil \Delta/q \rceil - \lceil \Delta/q \rceil = 0 \qquad \text{for } s \geq q\lceil Q/q \rceil$$

Case 1b: (2. 17)

$$\# \text{ of page faults (1b)} = \lceil \Delta/q \rceil \sigma - \frac{1}{2\lceil Q/q \rceil} (s^2/q^2 + s/q) \qquad (2.18)$$

$$\text{for } s < q\lceil Q/q \rceil$$

(Note s never equals $q\lceil Q/q \rceil$ in case 1b)

In all cases the number of accesses made is the same.

$$\# \text{ of accesses} = \Delta \, \sigma/q' \qquad (2.19)$$

where q' = # of bits in a word.

Thus we may write:

$$g_{1a}(\Delta) = \frac{\Delta\ \sigma/q'}{\lceil\Delta/q\rceil[(1 - \frac{s}{q\lceil Q/q\rceil} - \frac{1}{2\lceil Q/q\rceil} + \frac{1}{2\lceil Q/q\rceil}\lceil\Delta/q\rceil]} \tag{2.20}$$

for $\lceil\Delta/q\rceil \leq s/q$

and $s < q\lceil Q/q\rceil$

$$= \frac{\Delta\ \sigma/q'}{0} = \infty \tag{2.21}$$

for $\lceil\Delta/q\rceil \leq s/q$

and $s \geq q\lceil Q/q\rceil$

and

$$g_{1b}(\Delta) = \frac{\Delta\ \sigma/q'}{[\Delta/q]\sigma - \frac{1}{2[Q/q]}\ (s^2/q^2 + s/q)} \tag{2.22}$$

for $\lceil\Delta/q\rceil > s/q$

The expressions for $g_{2a}(\Delta)$ and $g_{2b}(\Delta)$ may be obtained directly from $g_{1a}(\Delta)$ and $g_{1b}(\Delta)$ by substituting $[\Delta/q] + 1$ for $\lceil\Delta/q\rceil$. This does not fall into the category of those things which are intuitively clear even to the most casual observer. However we will not go into a lengthy justification here because the development of case 2 is identical to case 1 which we have just considered. Any questions concerning case 2 may be resolved by referring to the discussion of case 1 and substituting 2a, 2b and $\lceil \Delta/q \rceil + 1$ in the appropriate places.

We may write $g_{2a}(\Delta)$ and $g_{2b}(\Delta)$ as follows.

$$g_{2a}(\Delta) = \frac{\Delta\ \sigma/q'}{(\lceil \Delta/q \rceil + 1)\left[\left(1 - \frac{s}{q\lceil Q/q \rceil} - \frac{1}{2\lceil Q/q \rceil}\right) + \frac{1}{2\lceil Q/q \rceil}(\lceil \Delta/q \rceil + 1)\right]} \qquad (2.23)$$

for $\lceil \Delta/q \rceil + 1 \leq s/q$

and $s < q\lceil Q/q \rceil$

$$= \infty \qquad (2.24)$$

for $\lceil \Delta/q \rceil + 1 \leq s/q$

and $s \geq q\lceil Q/q \rceil$

and

$$g_{2b}(\Delta) = \frac{\Delta\ \sigma/q'}{(\lceil \Delta/q \rceil + 1)\ \sigma - \frac{1}{2\lceil Q/q \rceil}(s^2/q^2 + s/q)} \qquad (2.25)$$

for $\lceil \Delta/q \rceil + 1 > s/q$

Note:

We will make a small alteration to the expressions for $g_{1a}(\Delta)$, $g_{1b}(\Delta)$, $g_{2a}(\Delta)$ and $g_{2b}(\Delta)$ at a later point in the text.

2.5 Consideration of a Distribution of Loop Lengths

In order to compute $E[g(\Delta)]$ we must have a density function for Δ. The question arises as to what should such a density function look like. One would imagine that a real program would be best modeled by a number of fixed length loops in addition to several variable length loops. The fixed length loops are generated by such things as actual DO loops. Variable length loops are generated by such activities as searching a list for some value. This would produce a density function containing a number of "spikes". However, the level of this model does not lend itself to such a detailed description of a program. We can however consider a density function with a single dominant peak with appropriate variables which allow control over the sharpness and the position of the peak. A function of the form

$$\frac{1}{a + (b - \Delta)^2}$$

provides us with the needed peak and control over the sharpness of the peak. Here the peak will occur at $\Delta = b$ and will have a maximum value of $1/a$.

For our purposes we will choose the constants in the equation to more closely suit our problem. Thus choosing

$$\frac{1}{\frac{1}{e^2} + \left(p - \frac{\Delta}{\lceil Q/q \rceil q}\right)^2} \qquad (2.26)$$

as our basic function we may position the peak at any point in the virtual address space by choosing a p in the range $0 \leq p \leq 1$. Thus if we wish to locate the peak of the distribution of Δ's at .2 of the length of the program then we choose p = .2. Notice that as we change the size of the program the peak will remain in the same relative position. We will consider this aspect in more detail later.

In order to make this function into a density function we must normalize it. Taking the integral

$$\int_o^{\lceil Q/q \rceil q} \frac{d\Delta}{\frac{1}{e^2} + (p - \frac{\Delta}{\lceil Q/q \rceil q})^2} = q\lceil Q/q \rceil e[\tan^{-1}(\frac{e}{q\lceil Q/q \rceil}\Delta - ep)]_o^{\lceil Q/q \rceil q}$$

$$= q\lceil Q/q \rceil e[\tan^{-1} e(1-p) + \tan^{-1} ep] \qquad (2.27)$$

and normalizing we have a density function for Δ

$$f_\Delta(\Delta) = \frac{1}{q\lceil Q/q \rceil e[\frac{1}{e^2} + (p - \frac{\Delta}{\lceil Q/q \rceil q})^2][\tan^{-1} e(1-p) + \tan^{-1} ep]} \qquad (2.28)$$

for $0 \leq \Delta \leq \lceil Q/q \rceil q$

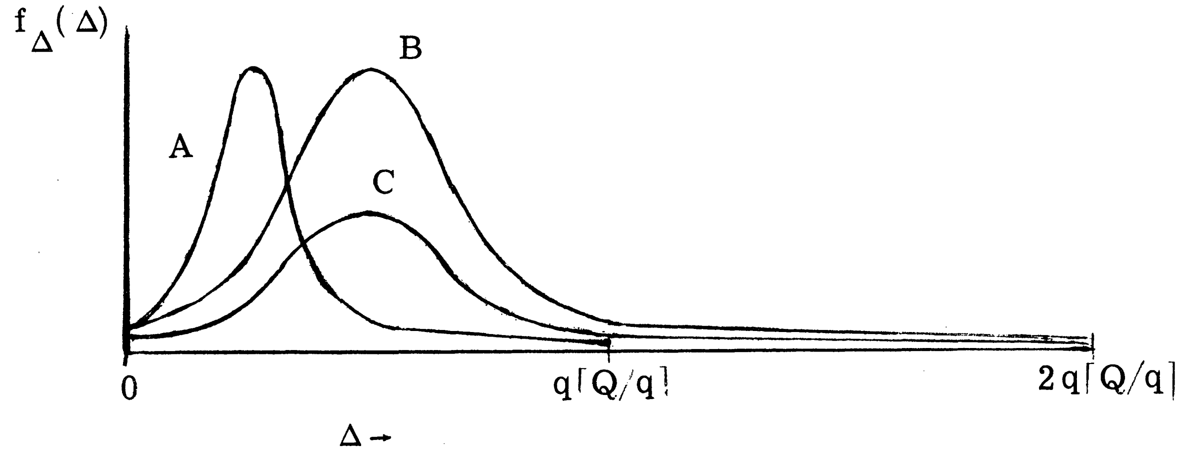
$= 0$ for $0 > \Delta > \lceil Q/q \rceil q$

Although this appears to be rather complex it is simply of the form

$$\frac{1}{a\Delta^2 + b\Delta + c}$$

where the constants a, b and c are rather cumbersome.

This density function was purposely constructed so that changes in the program size Q produce reasonable changes in the density function $f_\Delta(\Delta)$. The assumption has been made that as the size of a program increases the size of the access loops increase proportionally. The inclusion of $\lceil Q/q \rceil q$, the number of bits associated with the process, in this function performs exactly that function. We may draw a graph to illustrate this. Figure 2.7 shows what happened to the density function when the number of pages used by the program is doubled. Curve A depicts the density function for a program using $\lceil Q/q \rceil$ pages. Here $p = .25$ making the most frequent loop length 1/4 the size of the program. A program using twice as many pages $2\lceil Q/q \rceil$ but with the same p and e parameters would appear as curve C. Curve C may be obtained by



The Δ Distribution

Figure 2.7

first making a simple linear scale change to obtain curve B and then normalizing to obtain curve C.

There is another aspect of normalizing with respect to $q\lceil Q/q\rceil$ which must be considered. The term $q\lceil Q/q\rceil$ represents the apparent size of the program when the actual size is Q and the page size is q. Thus we see that the probability of a given loop length is affected by the page size. For instance if a program's actual size Q is "poorly" matched to the page size q the program will appear larger due to wasted space. We will assume that this wasted space is scattered at random throughout the programs virtual address space and thereby expands the length of loops being executed. This has one effect which must be corrected for. Earlier we said that the number of accesses made by a loop of length Δ traversed σ times would be $\Delta\sigma/q'$ where q' is the word size. However the process of normalizing the distribution of Δ to the apparent size of the program means that loops contain wasted space which does not contribute to actual program accesses. In order to account for this wasted space contained in loops we will express the number of program accesses in a given loop as:

$$\#\text{ of accesses} = \frac{\Delta\sigma}{q'} \; \frac{Q}{q\lceil Q/q\rceil} \qquad (2.29)$$

The term $\frac{Q}{q\lceil Q/q\rceil}$ can be recognized as the ratio between the actual size of the program and the apparent size of the program. Including this change the expressions for $g_{1a}(\Delta)$, $g_{1b}(\Delta)$, $g_{2a}(\Delta)$ and $g_{2b}(\Delta)$ become:

$$g_{1a}(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q\rceil)}{\lceil\Delta/q\rceil[(1 - \frac{s}{q\lceil Q/q\rceil} - \frac{1}{2\lceil Q/q\rceil} + \frac{1}{2\lceil Q/q\rceil}\lceil\Delta/q\rceil]} \qquad (2.30)$$

for $\lceil\Delta/q\rceil \leq s/q$

for $s < q\lceil Q/q\rceil$

$$= \frac{\Delta\ \sigma/q'}{0} = \infty \qquad (2.31)$$

for $\lceil\Delta/q\rceil \leq s/q$

and $s \geq q\lceil Q/q\rceil$

$$g_{1b}(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q\rceil)}{\lceil\Delta/q\rceil\sigma - \frac{1}{2\lceil Q/q\rceil}\ (s^2/q^2 + s/q)} \qquad (2.32)$$

for $\lceil\Delta/q\rceil > s/q$

$$g_{2a}(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q\rceil)}{(\lceil\Delta/q\rceil + 1)[(1 - \frac{s}{q\lceil Q/q\rceil} - \frac{1}{2\lceil Q/q\rceil}) + \frac{1}{2\lceil Q/q\rceil}(\lceil\Delta/q\rceil + 1)]} \qquad (2.33)$$

for $\lceil\Delta/q\rceil + 1 \leq s/q$

and $s < q\lceil Q/q\rceil$

$$= \infty \tag{2.34}$$

for $\lceil \Delta/q \rceil + 1 \leq s/q$

and $s \geq q\lceil Q/q \rceil$

and

$$g_{2b}(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q \rceil)}{([\Delta/q] + 1)\ \sigma - \frac{1}{2\lceil Q/q \rceil}\ (s^2/q^2 + s/q)} \tag{2.35}$$

for $\lceil \Delta/q \rceil + 1 > s/q$

2.6 Final Model Development

We may express the expected value of $g(\Delta)$ as

$$E[g(\Delta)] = \int_0^{q\lceil Q/q \rceil} [g_1(\Delta) P(\text{Case } 1|\Delta)\ f_\Delta(\Delta) \tag{2.36}$$

$$+ g_2(\Delta) P(\text{Case } 2|\Delta)\ f_\Delta(\Delta)]d\Delta$$

This integral must be divided into three separate integrals in order to accommodate the changes in functional representation which occur at $\Delta = qs\lceil Q/q \rceil - 1$ and $\Delta = qs\lceil Q/q \rceil$. Thus it will take the form

$$E[g(\Delta)] = \int_0^{(s-q)} [g_{1a}(\Delta)\ P(\text{Case } 1|\Delta) f_\Delta(\Delta) + g_{2a}(\Delta)\ P(\text{Case } 2|\Delta f_\Delta(\Delta)]d\Delta$$

$$+ \int_{s-q}^{s} [g_{1a}(\Delta)\ P(\text{Case } 1|\Delta) f_\Delta(\Delta) + g_{2b}(\Delta)\ P(\text{Case } 2|\Delta) f_\Delta(\Delta)]d\Delta$$

$$+ \int_s^{q\lceil Q/q \rceil} [g_{1b}(\Delta) P(\text{Case } 1|\Delta) f_\Delta(\Delta) + g_{2b}(\Delta) P(\text{Case } 2|\Delta) f_\Delta(\Delta)]d\Delta$$

(2.37)

The probabilities P(Case $1|\Delta$) and P(Case $2|\Delta$) may be taken from Equation (2. 5). They are:

$$P(\text{Case } 1|\Delta) = \lceil\Delta/q\rceil - \Delta/q \tag{2.38}$$

$$P(\text{Case } 2|\Delta) = 1 - \lceil\Delta/q\rceil + \Delta/q \tag{2.39}$$

We can at this point simplify our equations considerably. We will proceed by examining the terms in the above integrals in some detail. Beginning with the term in the first integral we will factor out $f_\Delta(\Delta)$ giving us

$$[g_{1a}(\Delta)\, P(\text{Case } 1|\Delta) + g_{2a}(\Delta)\, P(\text{Case } 2|\Delta)]\, f_\Delta(\Delta) \tag{2.40}$$

Let us now consider the function $g_{1a}(\Delta)$ in some detail. Rewriting it

$$g_{1a}(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q\rceil)}{\lceil\Delta/q\rceil[(1 - s - \frac{1}{2\lceil Q/q\rceil}) + \frac{1}{2\lceil Q/q\rceil}\lceil\Delta/q\rceil]} \tag{2.41}$$

$$\text{for } \lceil\Delta/q\rceil \leq s\lceil Q/q\rceil$$

$$\text{and } s \neq 1$$

Notice that between values where $\lceil\Delta/q\rceil$ changes abruptly $g_{1a}(\Delta)$ increases linearly with Δ. This is simply because only the numerator contains a Δ that is not in the form $\lceil\Delta/q\rceil$ and this Δ is in effect multiplied times a constant except in the neighborhood of points where Δ/q is an integer. Thus we would expect a graph of $g_{1a}(\Delta)$ to be a series of straight lines connected by discontinuities.

The form of $g_{2a}(\Delta)$ is the same as $g_{1a}(\Delta)$ except that $\lceil \Delta/q \rceil$ is replaced by $\lceil \Delta/q \rceil + 1$. It should be clear that our comments about $g_{1a}(\Delta)$ also apply to $g_{2a}(\Delta)$.

Another most important characteristic of these two functions is that

$$g_{2a}(\Delta) = \lim_{\epsilon \to 0} g_{1a}(\Delta + \epsilon) \text{ for } \Delta/q = \text{integer} \qquad (2.42)$$

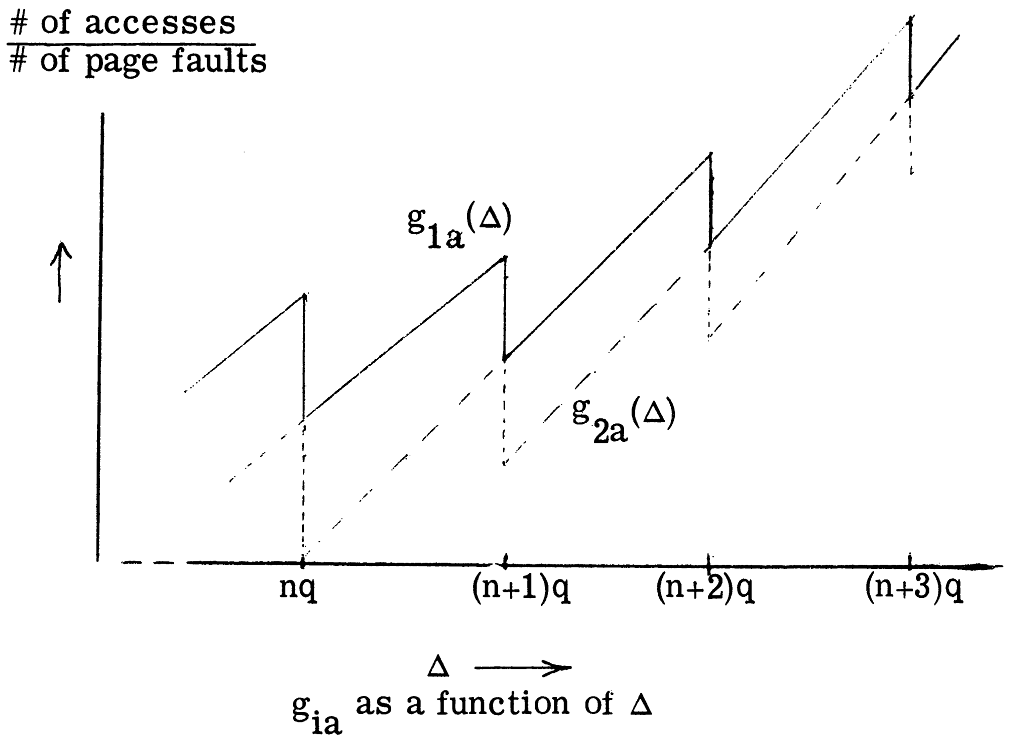
In Figure 2.8 we show a graph of $g_{1a}(\Delta)$ and $g_{2a}(\Delta)$ showing the characteristics we have just discussed.

Next we would like to turn to a detailed examination of the term

$$[g_{1a}(\Delta)\ P(\text{Case } 1|\Delta) + g_{2a}(\Delta)\ P(\text{Case } 2|\Delta)] \qquad (2.43)$$

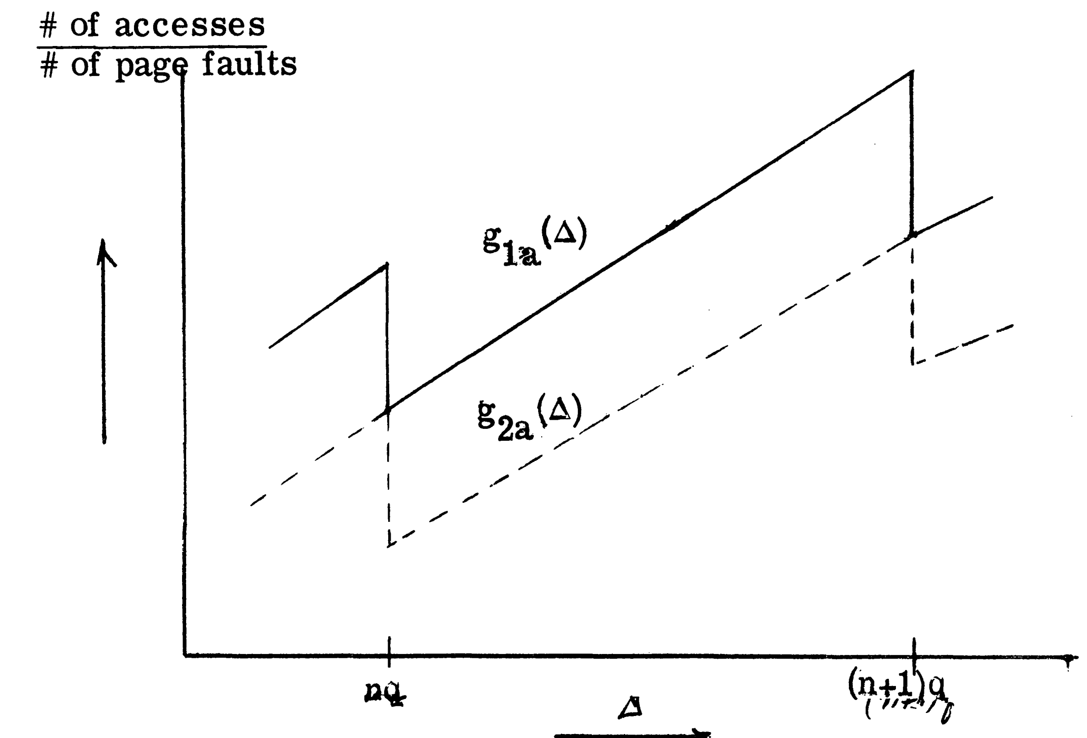
over a range of delta from nq to (n+1)q. Figure 2.9 shows three graphs one displayed a detailed graph of $g_{1a}(\Delta)$ and $g_{2a}(\Delta)$ and two others showing $P(\text{Case } 1|\Delta)$ and $P(\text{Case } 2|\Delta)$ over the same range.

If we examine these graphs and the term in question we can see that at a point just to the right of $\Delta = nq$ (or $\Delta = nq + \epsilon$ where ϵ is arbitrarily small) the Expression 2.43 is equal to $g_{1a}(\Delta)$. Correspondingly at the other end of the range where $\Delta = (n+1)q$ the expression is equal to $g_{2a}(\Delta)$. The value of the expression clearly begins on the left equal to $g_{1a}(\Delta)$ and progresses between the two function $g_{1a}(\Delta)$ and $g_{2a}(\Delta)$ equaling $g_{2a}(\Delta)$ at the page boundary. The expression is clearly never greater than $g_{1a}(\Delta)$ nor less than $g_{2a}(\Delta)$.

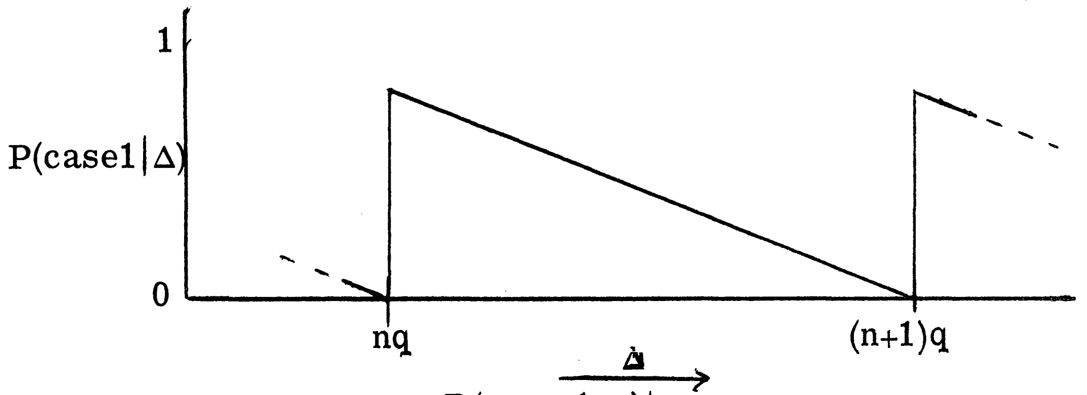


g_{ia} as a function of Δ

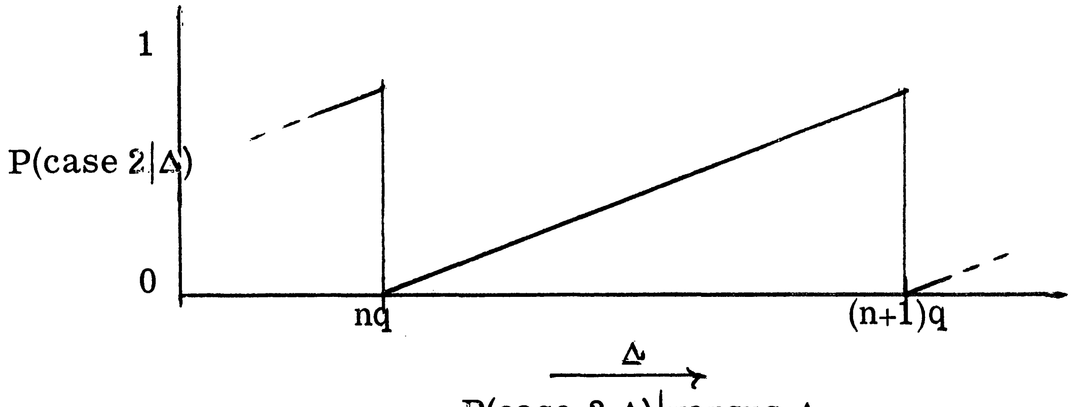
Figure 2. 8



g_{1a} (Δ) and g_{2a}(Δ) versus Δ



P(case 1 Δ)|versus Δ



P(case 2 Δ)|versus Δ

The Components of $H_1(\Delta)$
Figure 2.9

We will replace Expression 2.43 with the function

$$H_1(\Delta) = \frac{(\Delta\ \sigma/q')\ (Q/q\lceil Q/q\rceil)}{(\Delta/q + 1)[(1 - \frac{s}{q\lceil Q/q\rceil} - \frac{1}{2\lceil Q/q\rceil}) + \frac{1}{2\lceil Q/q\rceil}(\Delta/q + 1)]}$$

$$= \frac{2\,\sigma\, q\, Q}{q'} \times \frac{\Delta}{(\Delta + q)(2(q\lceil Q/q\rceil - s) + \Delta)} \qquad (2.44)$$

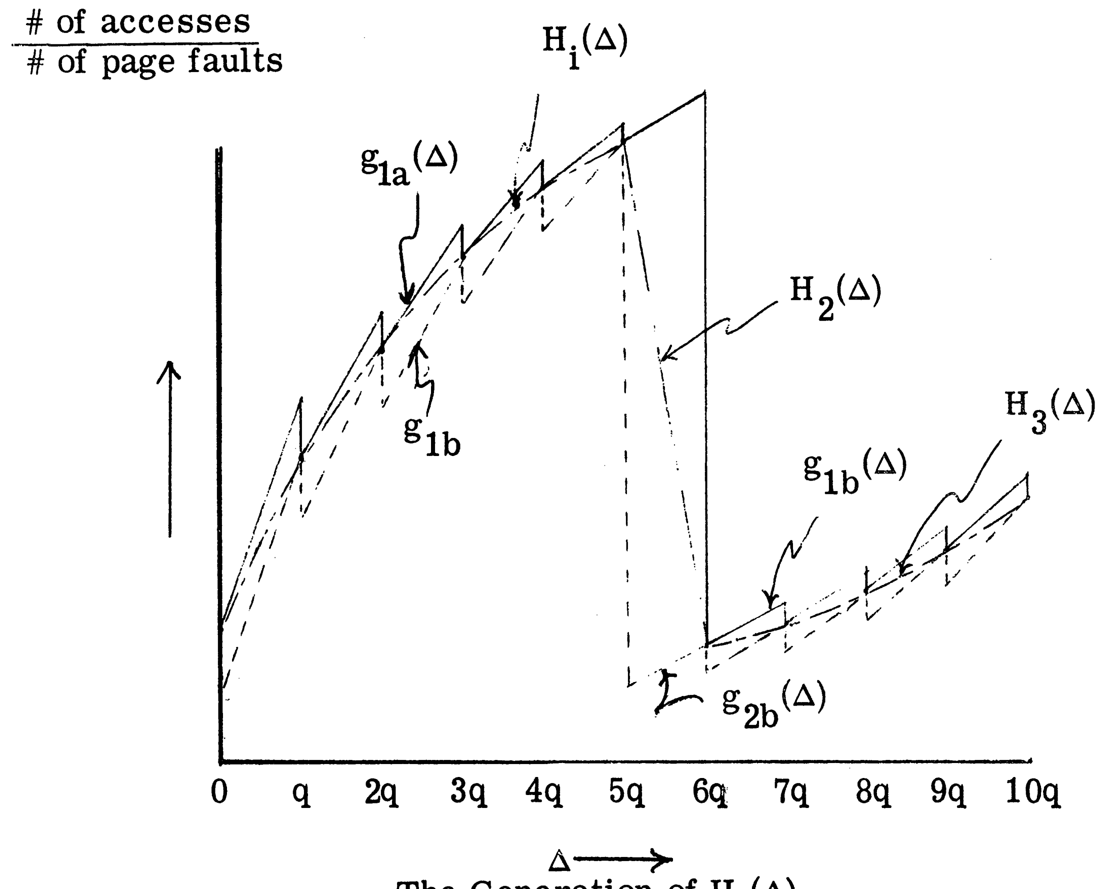
We can recognize $H_1(\Delta)$ as $g_{2a}(\Delta)$ where $\lceil \Delta/q\rceil$ has been replaced by Δ/q. $H_1(\Delta)$ is clearly an exact replacement for Expression 2.43 at page boundaries and is a smooth function which conforms to

$$g_{2a}(\Delta) \leq H_1(\Delta) \leq g_{1a}(\Delta). \qquad (2.45)$$

Keeping in mind that we will be integrating the product $H_1(\Delta)\,f_\Delta(\Delta)$ it is clear that any errors introduced by the use of $H_1(\Delta)$ are trivial when compared to the rather arbitrary choice of $f_\Delta(\Delta)$.

If we examine the other two integrals of Equation 2.37 we see that the situation in each is approximately the same. In each we have two functions which are straight lines between page boundaries and have discontinuities at the page boundaries. Figure 2.10 shows the general form of the functions $g_{1a}(\Delta)$, $g_{2a}(\Delta)$, $g_{1b}(\Delta)$ and $g_{2b}(\Delta)$. In this plot we have plotted these functions over the ranges where they appear in the integrals of Equation 2.37 in the case of $s = 5q$.

Turning to the third integral of Equation 2.37 we will replace the term



The Generation of $H_i(\Delta)$

Figure 2.10

$$[g_{1b}(\Delta)\ P(\text{Case } 1|\Delta) + g_{2b}(\Delta)\ P(\text{Case } 2|\Delta)] \qquad (2.46)$$

with

$$H_3(\Delta) = \frac{(\Delta\ \sigma/q')(Q/q\lceil Q/q\rceil)}{(\Delta/q + 1)\sigma - \frac{1}{2\lceil Q/q\rceil}(s^2/q^2 + s/q)} \qquad (2.47)$$

$$= \frac{Q}{q'\lceil Q/q\rceil} \times \frac{\Delta}{(\Delta + (q - \frac{q}{2\sigma\lceil Q/q\rceil}(s^2/q + s/q)))}$$

The function $H_3(\Delta)$ can be recognized as $g_{2b}(\Delta)$ where $[\Delta/q]$ has been replaced by Δ/q. Again $H_3(\Delta)$ is equal to the expression in question at page boundaries and satisfies

$$g_{2b}(\Delta) \leq h_3(\Delta) \leq g_{1b}(\Delta) \qquad (2.48)$$

In the case of the second integral of Equation 2.37 we will factor out the term

$$[g_{1a}(\Delta)\ P(\text{Case } 1|\Delta) + g_{2b}(\Delta)\ P(\text{Case } 2|\Delta)]. \qquad (2.49)$$

This term combines both cases a and b and is illustrated in Figure 2.10 in the range of Δ from 5q to 6q. We will replace this term with a straight line which connects the values of the term at the page boundaries. The general form of this approximation will be simply

$$H_2(\Delta) = (\text{slope})\Delta + (\text{constant}). \qquad (2.50)$$

Since the value of $H_1(\Delta)$ and $H_2(\Delta)$ is exact at the page boundaries we may compute the slope as

$$\text{slope} = \frac{H_3(s) - H_1(s-q)}{s - (s-q)}$$

$$= \frac{H_3(s) - H_1(s-q)}{q} \qquad (2.51)$$

and slope for the constant

$$\text{constant} = H_3(s) - (\text{slope})\, s$$

Expanding these terms we see that

$$H_2(\Delta) = \left[\frac{s(Q/q\lceil Q/q\rceil)}{q'(s + q - \frac{q}{2\sigma\lceil Q/q\rceil}(s^2/q^2 + s/q))} - \frac{2\,\sigma\, Q\,(s-q)}{q's(2(q\lceil Q/q\rceil - s) + s - q)} \right] \Delta$$

$$+ \left[\frac{((q-s)\,s)\,(Q/q\lceil Q/q\rceil)}{q'(s + q - \frac{q}{2\sigma\lceil Q/q\rceil}(s^2/q^2 + s/q))} - \frac{2\sigma Q(s-q)}{q'(2(q\lceil Q/q\rceil - s) + s - q)} \right]$$

(2.52)

We can now write $E[g(\Delta)]$ using the approximating functions.

$$E[g(\Delta)] = 1 \qquad (2.53)$$

$$\text{for } 0 \leq s \leq q$$

$$= \int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta + \int_{s-q}^{s} H_2(\Delta) f_\Delta(\Delta)\, d\Delta$$

$$+ \int_s^{q\lceil Q/q\rceil} H_3(\Delta) f_\Delta(\Delta)\, d\Delta \qquad (2.54)$$

$$\text{for } q \le s < q\lceil Q/q\rceil$$

$$= \infty \tag{2.55}$$

$$\text{for } s \ge q\lceil Q/q\rceil$$

This integral is evaluated in Appendix A and the results follow:

$$E[g(\Delta)] = 1 \tag{2.56}$$

$$\text{for } 0 \le s < q$$

$$\begin{aligned}
E[g(\Delta)] = {} & A_1\Big[K_1 \ln\Big(\frac{a_1 + s - q}{a_1}\Big) + K_2 \ln\Big(\frac{a_2 + s - q}{a_2}\Big) \\
& + K_3 \frac{1}{2a_5} \ln\Big(\frac{a_3 + a_4(s-q) + a_5(s-q)^2}{a_3}\Big) \\
& + \Big(K_4 - K_3\frac{a_4}{2a_5}\Big) q\lceil Q/q\rceil e\Big[\tan^{-1}\Big(\frac{e(s-q)}{q\lceil Q/q\rceil} - ep\Big) + \tan^{-1}(ep)\Big]\Big] \\
& + A_2\Big[K_5 \frac{1}{2a_5} \ln\Big(\frac{a_3 + a_4 s + a_5 s^2}{a_3 + a_4(s-q) + a_5(s-q)^2}\Big) \\
& + \Big(K_6 - K_5\frac{a_4}{2a_5}\Big) q\lceil Q/q\rceil e\Big[\tan^{-1}\Big(\frac{es}{q\lceil Q/q\rceil} - ep\Big) - \tan^{-1}\Big(\frac{e(s-q)}{q\lceil Q/q\rceil} - ep\Big)\Big]\Big] \\
& + A_3\Big[K_7 \ln\Big(\frac{a_6 + q\lceil Q/q\rceil}{a_6 + s}\Big) \\
& + K_8 \frac{1}{2a_5} \ln\Big(\frac{a_3 + a_4 q\lceil Q/q\rceil + a_5 q^2\lceil Q/q\rceil^2}{a_3 + a_4 s + a_5 s^2}\Big) \\
& + \Big(K_9 - K_8\frac{a_4}{2a_5}\Big) q\lceil Q/q\rceil e\Big[\tan^{-1} e(1-p) - \tan^{-1}\Big(\frac{es}{q\lceil Q/q\rceil} - ep\Big)\Big]\Big]
\end{aligned} \tag{2.57}$$

$$\text{for } q \leq s < q\lceil Q/q \rceil$$

$$E[g(\Delta)] = \infty \qquad (2.58)$$

$$\text{for } s \geq q\lceil Q/q \rceil$$

Where

$$A_1 = \frac{2\sigma(Q/\lceil Q/q \rceil}{q'\, e[\tan^{-1} e(1-p) + \tan^{-1}(ep)]} \qquad (2.59)$$

$$A_2 = \frac{Q}{q\lceil Q/q \rceil e[\tan^{-1} e(1-p) + \tan^{-1}(ep)]} \qquad (2.60)$$

$$A_3 = \frac{Q/q}{q'\lceil Q/q \rceil e[\tan^{-1} e(1-p) + \tan^{-1}(ep)]} \qquad (2.61)$$

$$a_1 = q \qquad (2.62)$$

$$a_2 = 2(q\lceil Q/q \rceil - s) \qquad (2.63)$$

$$a_3 = \frac{1}{e^2} + p^2 \qquad (2.64)$$

$$a_4 = -\frac{2p}{q\lceil Q/q \rceil} \qquad (2.65)$$

$$a_5 = \frac{1}{q^2\lceil Q/q \rceil^2} \qquad (2.66)$$

$$a_6 = q - \frac{q}{2\sigma\lceil Q/q \rceil}(s^2/q^2 + s/q) \qquad (2.67)$$

$$K_1 = \frac{-a_1}{(a_2 - a_1)(a_3 - a_4 a_1 + a_5 a_1^2)} \tag{2.68}$$

$$K_2 = \frac{-a_2}{(a_1 - a_2)(a_3 - a_4 a_2 + a_5 a_2^2)} \tag{2.69}$$

$$K_3 = -(K_1 + K_2) a_5 \tag{2.70}$$

$$K_4 = -K_1 \frac{a_3}{a_1} - K_2 \frac{a_3}{a_2} \tag{2.71}$$

$$K_5 = \frac{s}{q'(s + a_6)} - \frac{2\sigma q \lceil Q/q \rceil (s-q)}{sq'(a_2 + s - q)} \tag{2.72}$$

$$K_6 = \frac{(q - s) s}{q'(s + a_6)} + \frac{2\sigma q \lceil Q/q \rceil (s-q)}{q'(a_2 + s - q)} \tag{2.73}$$

$$K_7 = \frac{-a_6}{a_3 - a_4 a_6 + a_5 a_6^2} \tag{2.74}$$

$$K_8 = \frac{1 + K_7 \left(\frac{a_3}{a_6} - a_4 \right)}{a_6} \tag{2.75}$$

$$K_9 = -K_7 \frac{a_3}{a_6} \tag{2.76}$$

To begin, $E[g(\Delta)]$ is the expected ratio between the number of accesses and the number of page faults or in other words the average number of accesses required to produce a page fault. Thus we may express the lifecycle function as a seven variable function

$$f(s, q', q, Q, \sigma, p, e) = E[g(\Delta)] \qquad (2.77)$$

Before continuing we would like to add one additional consideration. We will define

$$H = \text{the number of storage accesses between I}\emptyset\text{ interrupts.} \qquad (2.78)$$

Here I∅ involves interaction outside of the storage system, possibly with a terminal user. The rate of I∅ interrupts per access is then $1/H$ and the rate of storage interrupts will be $\frac{1}{E[g(\Delta)]}$. We may express the combined rate as

$$\text{Interrupt rate} = \frac{1}{E[g(\Delta)]} + \frac{1}{H} \qquad (2.79)$$

We may now write an expression for an eight variable life cycle function which includes I∅ interrupts as

$$f(s, q', q, Q, \sigma, p, e, H) = \frac{1}{\frac{1}{E[g(\Delta)]} + \frac{1}{H}} \qquad (2.80)$$

where

s = absolute allocation in bits

$$0 \leq s$$

q' = word length in bits

$$0 < q' \leq q$$

q = page size in bits

$$q' \leq q$$

Q = program size in bits

$$0 < Q$$

σ = cycle repetition number (dimensionless)

$$1 \leq \sigma$$

p = the normalized mode of the cycle length

$$0 \leq p \leq 1$$

e = a control of the Δ distribution

$$0 < e$$

H = # of storage accesses between IØ interrupts

$$0 < H$$

2.7 Examples of Model Behavior With Varying Allocation

Here we want to study the lifecycle function and its behavior. We will consider a number of graphs of the function. To begin we will consider a specific case where:

q' = 32 bits

(a 32 bit word)

q = 16384 bits

(a 2048 byte or 512 word page)

Q = 163840 bits

(a 10 page program)

$\sigma = 20$

(loops are accessed 20 times before a random jump)

$p = .5$

(the most frequent loop length is 1/2 the length of the program size)

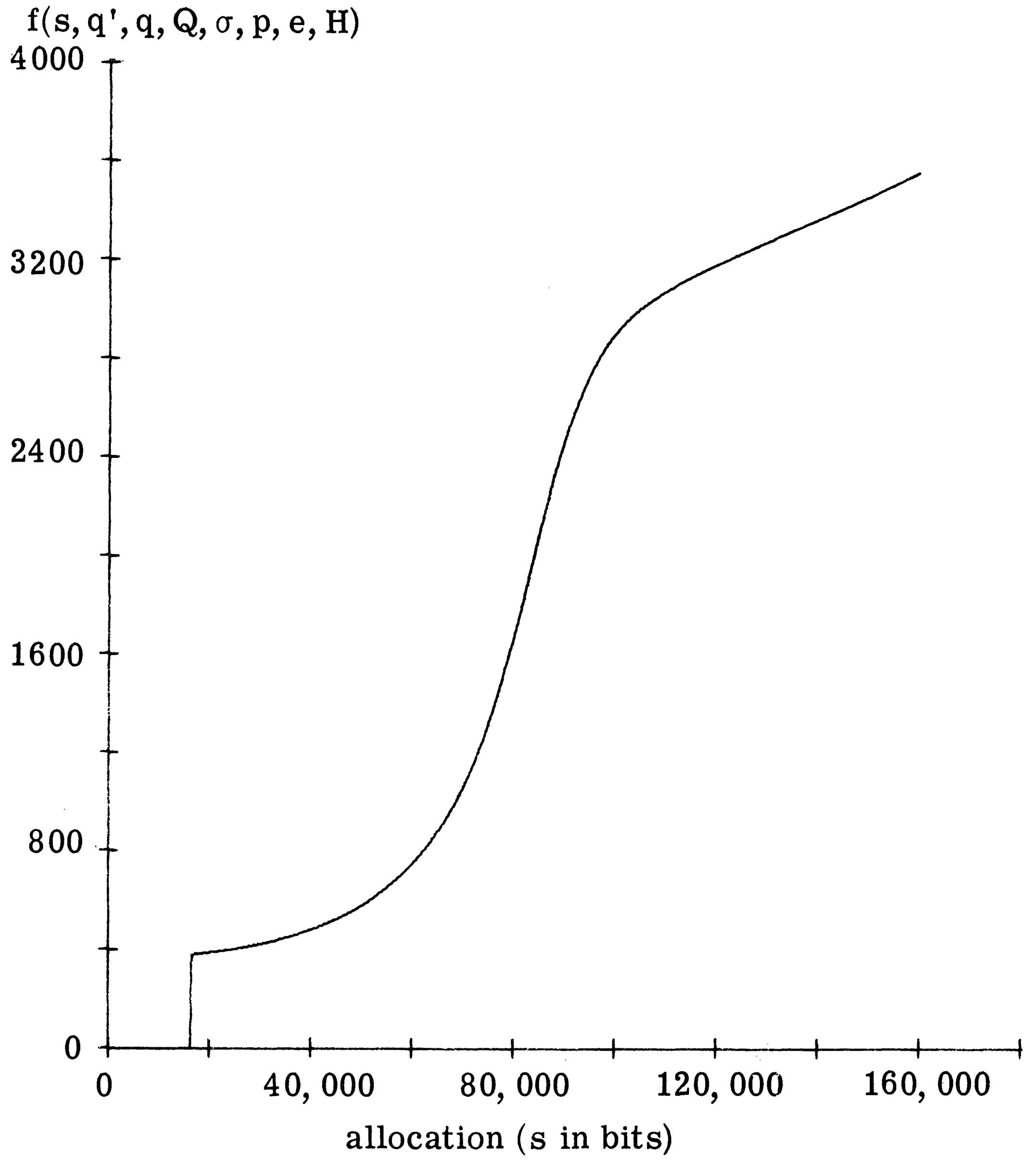
$e = 20$

(the distribution of loop lengths has a moderately sharp peak)

$H = 4000$

(IØ interrupts occur at a rate of 1 out of every 4000 accesses)

These parameters were selected with two goals in mind. First reality and second a demonstration of functional behavior as we varied the parameters away from this "standard" case. Examining Figure 2.11 we see a graph of this "standard" lifecycle function plotted as a function of the absolute allocation s. We may describe this graph in terms of three regions; the under allocated on the left, the fully allocated on the right, and the transition region in the center where we see that the lifecycle function grows rapidly with respect to allocation.



The Lifecycle Function Versus Allocation
for the "Standard" Case

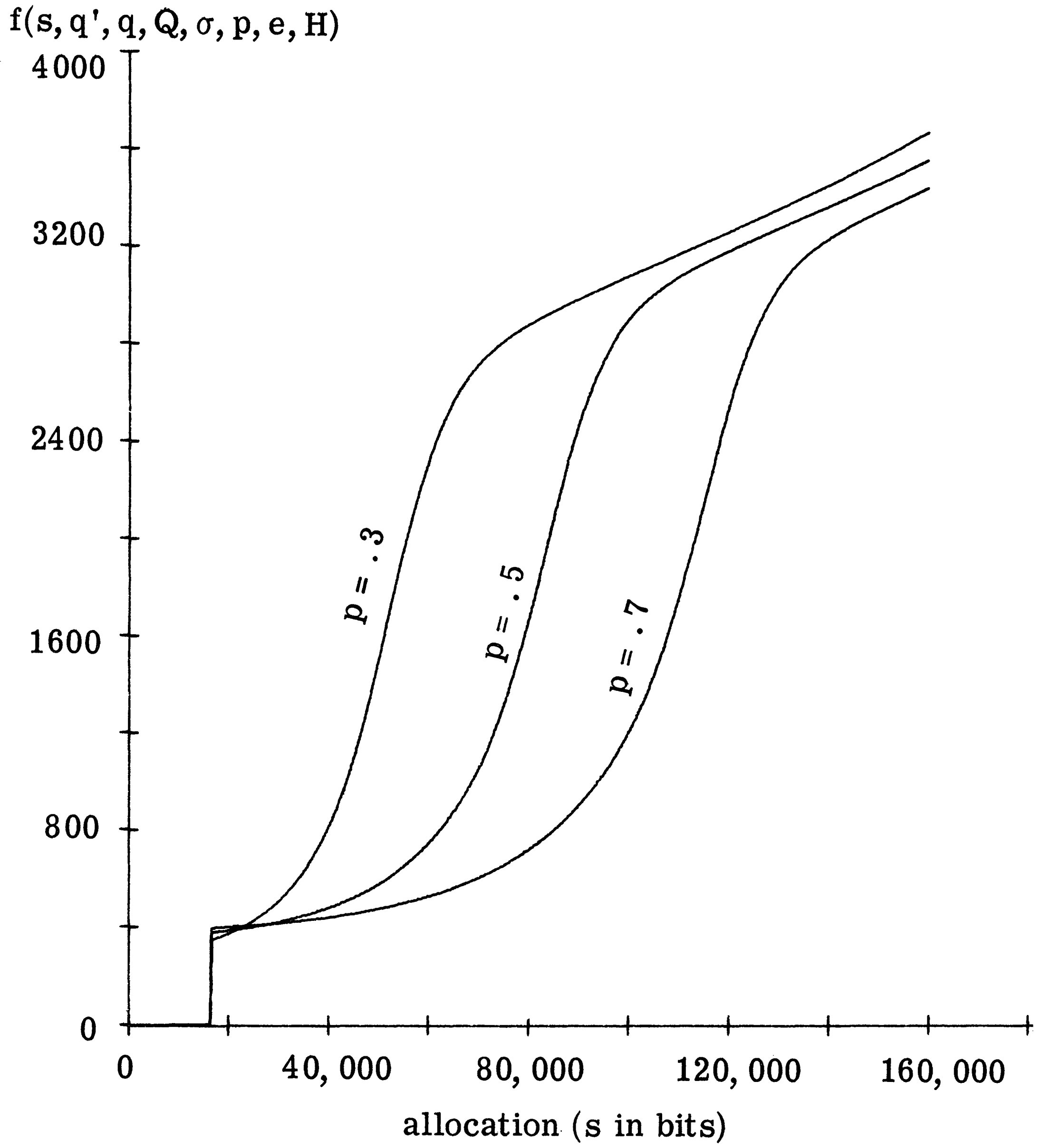
Figure 2. 11

In the under allocated region the mean number of accesses required to produce a page fault is most strongly affected by the page size q and the word size q'. In fully allocated region the number of times a loop is accessed before beginning a new loop, σ, and the number of accesses per IØ interrupt, H, have the greatest consequence. The location and sharpness of the transition region is affected most strongly by the variables that control the loop length distribution p and e.

In Figure 2. 12 a graph is displayed showing the lifecycle function for three different values of p (normalized mode of the loop length). All the other variables are the same as our "standard" case just considered and the plot for p = .5 is the "standard" case.

Here we see that as we increase p and thus increase the length of the loops the transition region moves to the right or to a higher allocation. Likewise reducing the value of p reduces the allocation required to reach the transition region. In terms of the working set we could say that increasing p increases the size of the working set and decreasing p decreases the size of the working set.

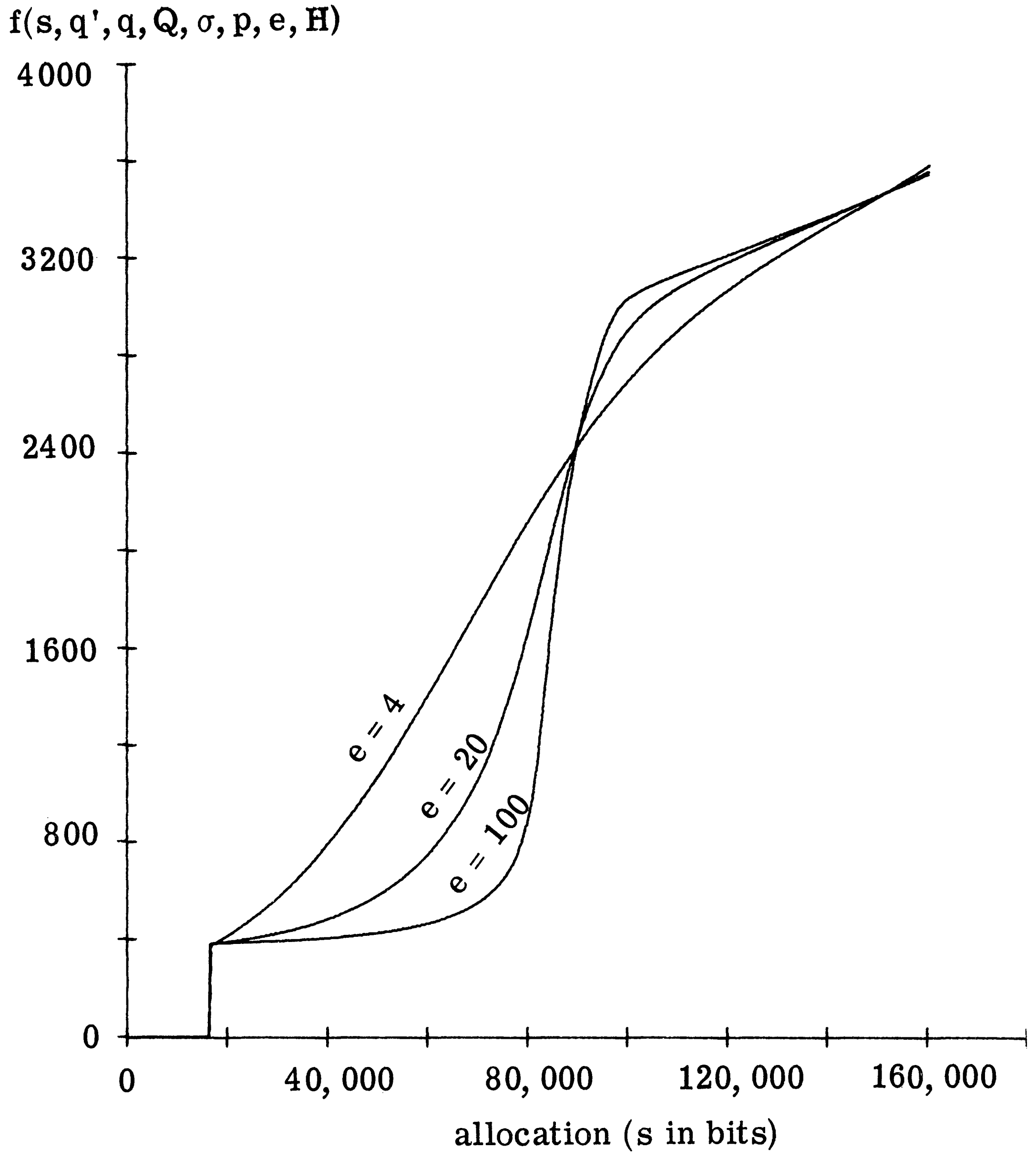
Turning to Figure 2. 13 we also see a graph containing three plots of the lifecycle function. Here we have varied e which controls the sharpness of the distribution of loop lengths. Small values of e produce a broad distribution of loop lengths and large values produce a sharply peaked distribution with low variance.



The Lifecycle Function Versus Allocation

With Variations in p

Figure 2. 12



The Lifecycle Function Versus Allocation
With Variations in e

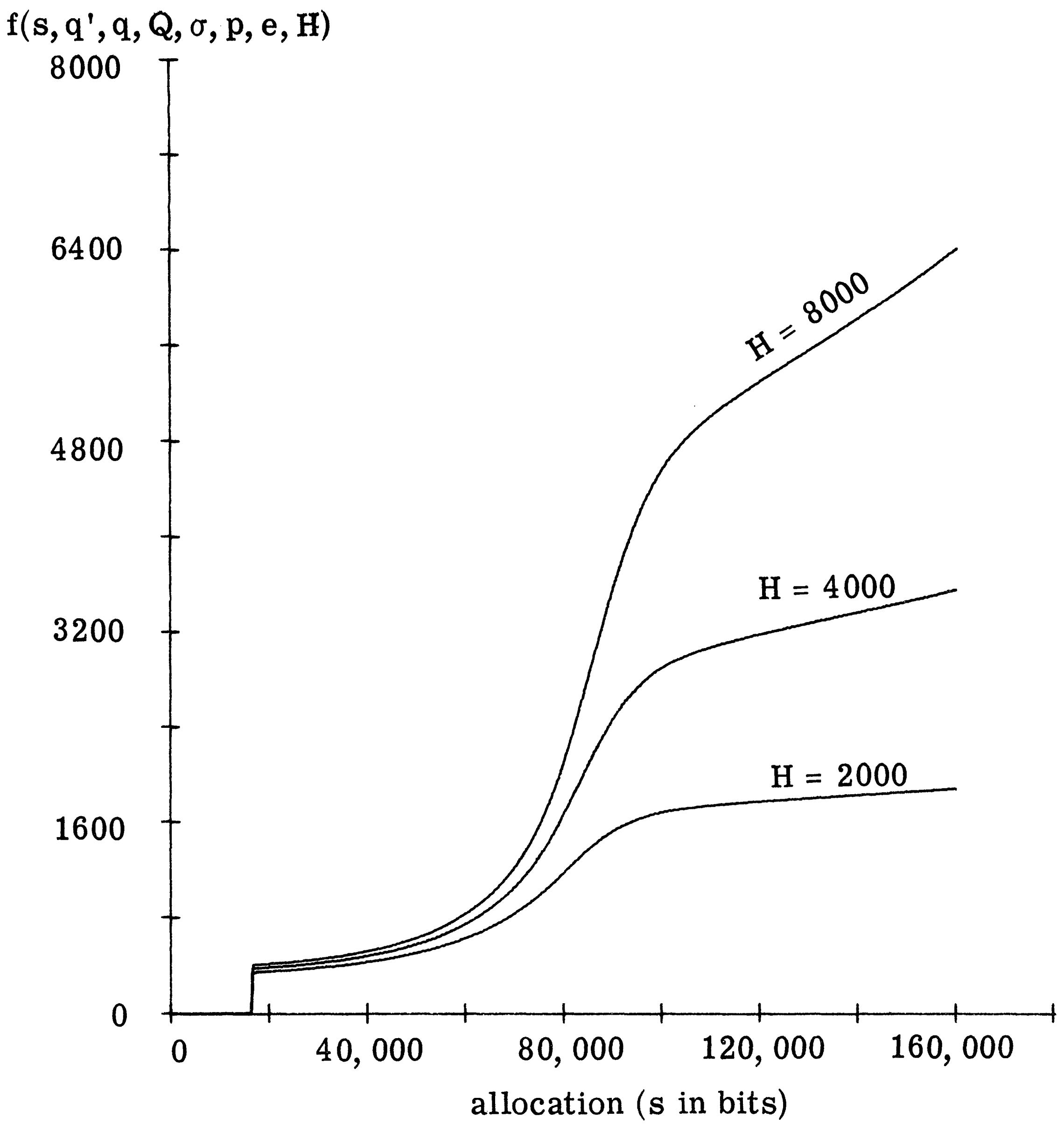
Figure 2. 13

Again we display the "standard" curve and two variations from it. The standard curve is in the center where e = 20.

Here we see that the broad distribution of loop lengths in the case of e = 4 produce a wide transition region. In the case of e = 100 we have a sharper distribution of loop lengths and corresponding a sharper transition region. In terms of the working set, if the size of the working set is sharply defined we can expect a sharp rise in the lifecycle function when the allocation exceeds the size of the working set. Correspondingly if the size of the working set varies as the program is executed and has a broad distribution we can expect a broad to non-existent transition region.

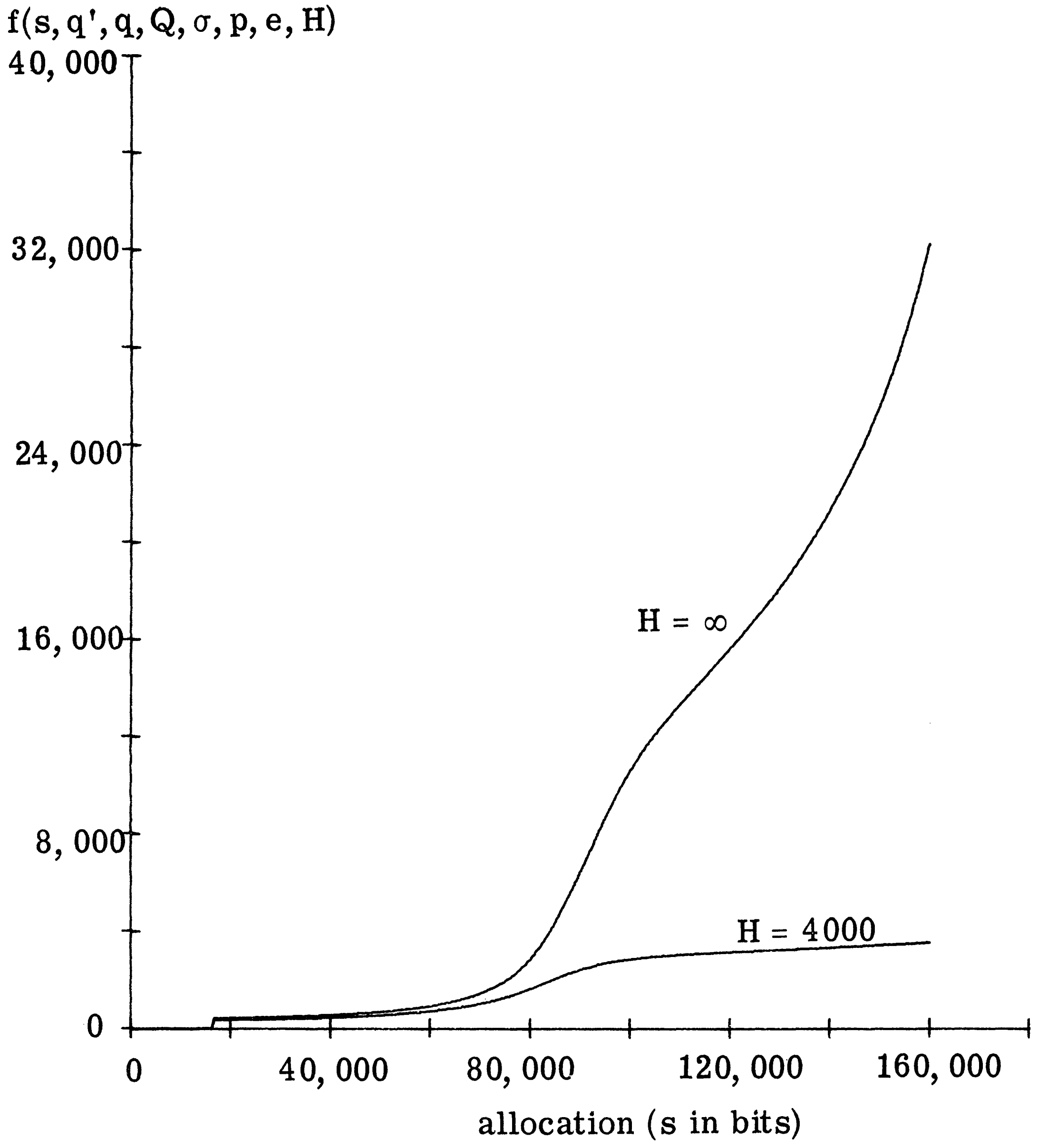
In the fully allocated region the two variables H and σ have a large effect. In Figure 2.14 a graph is shown with three plots of the lifecycle function. Here we have varied H, the number of accesses between IØ interrupts. The "standard" curve is as usual in the middle where H = 4000. The effect of varying H can be dramatic but is generally predictable. Notice that in the case of H = 8000 the curvature in the fully allocated region is different from the other two cases. In Figure 2.15 we show a case which is "standard" except that H = ∞. Notice here both the scale of the graph and the distinct change in curvature.

Figure 2.16 shows a graph of three cases where σ = 20 is the "standard" case. If we disregard the effects of IØ interrupt



The Lifecycle Function Versus Allocation
With Variations in H

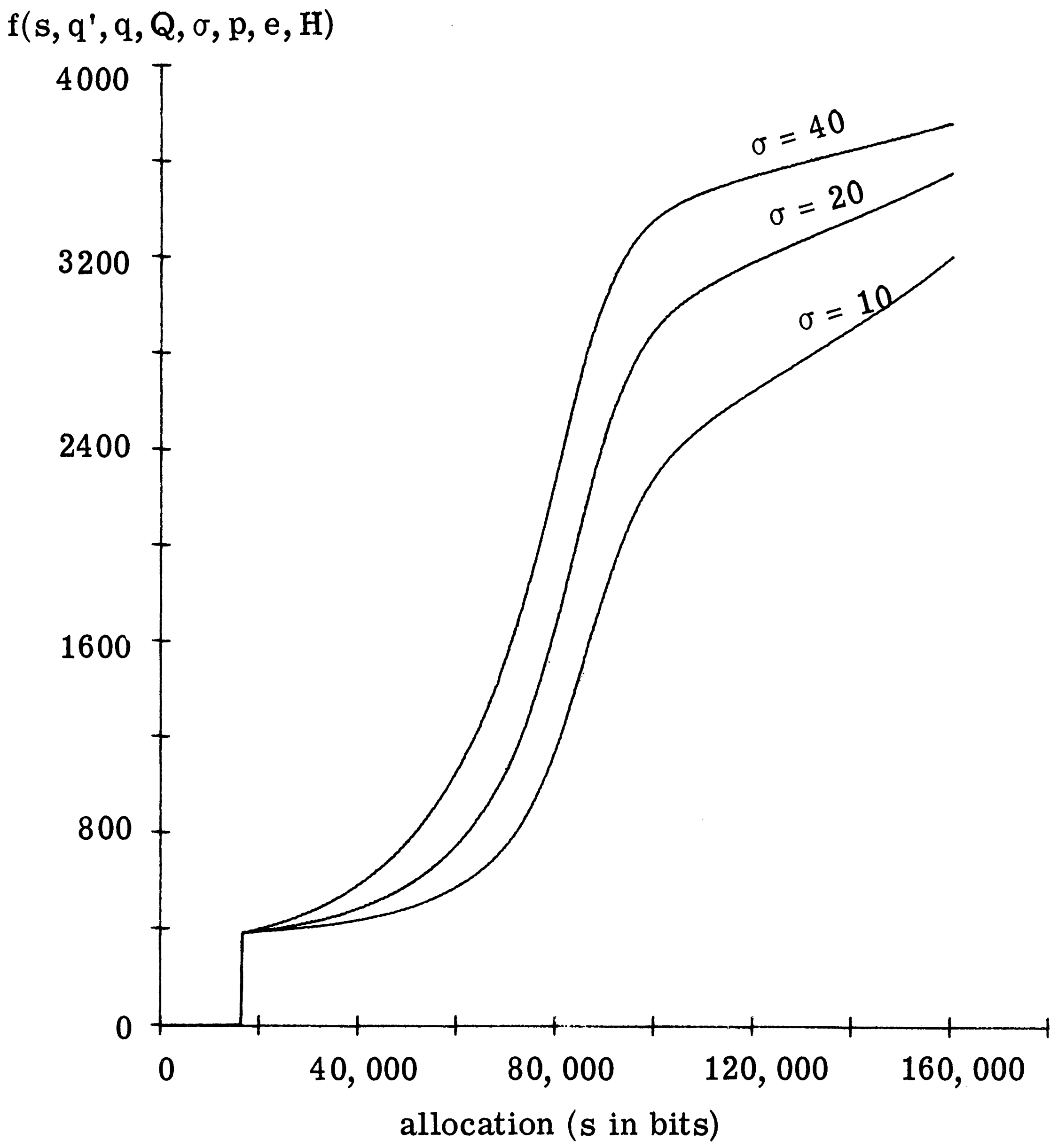
Figure 2. 14



The Lifecycle Function Versus Allocation

for H = ∞

Figure 2. 15



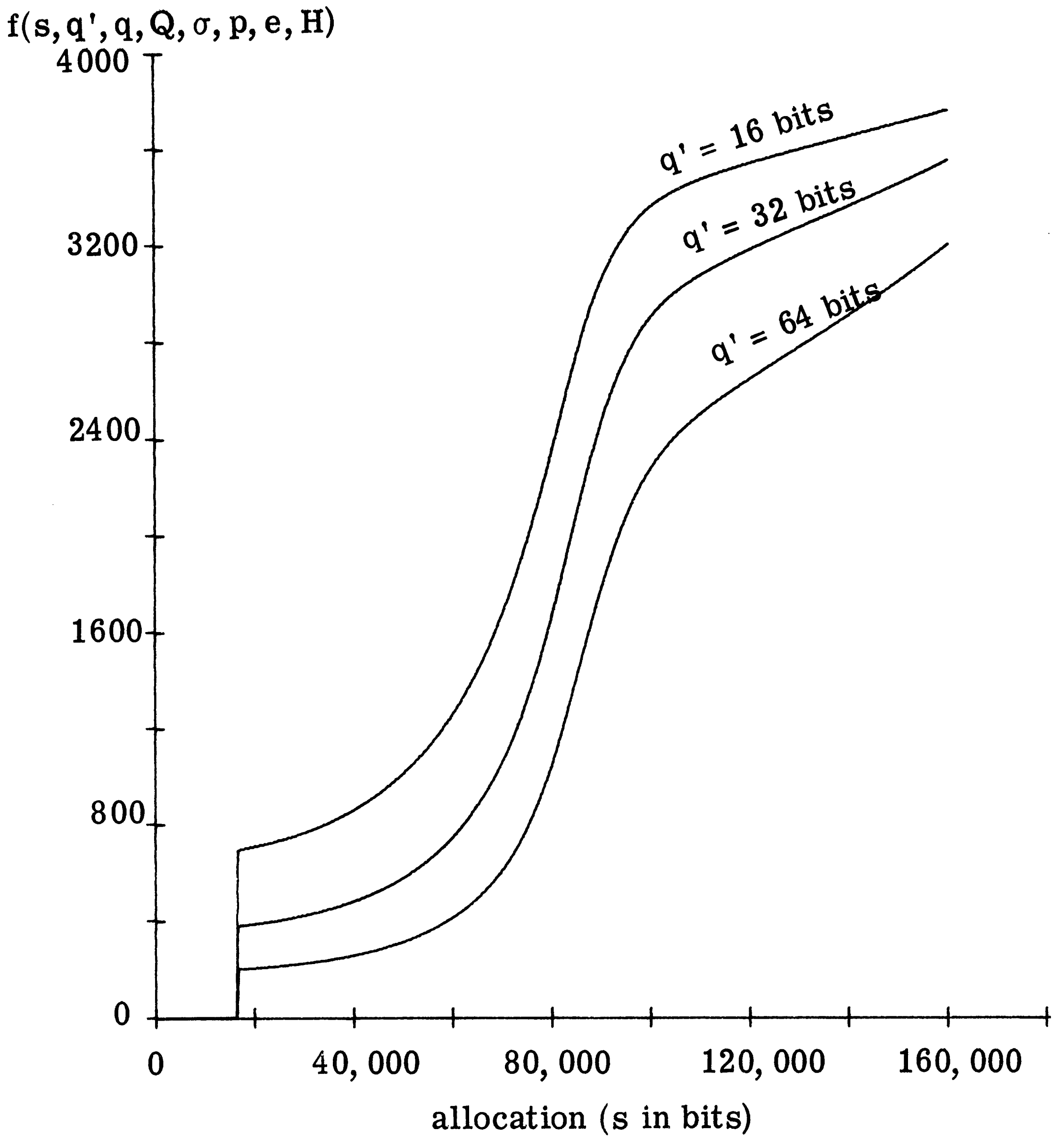
The Lifecycle Function Versus Allocation
With Variations in σ

Figure 2.16

we would expect that changes in the lifecycle function in the fully allocated region would be approximately equal to changes in σ. However in this case these changes are suppressed by the IØ interrupts. In fact since H = 4000 the lifecycle function cannot exceed 4000.

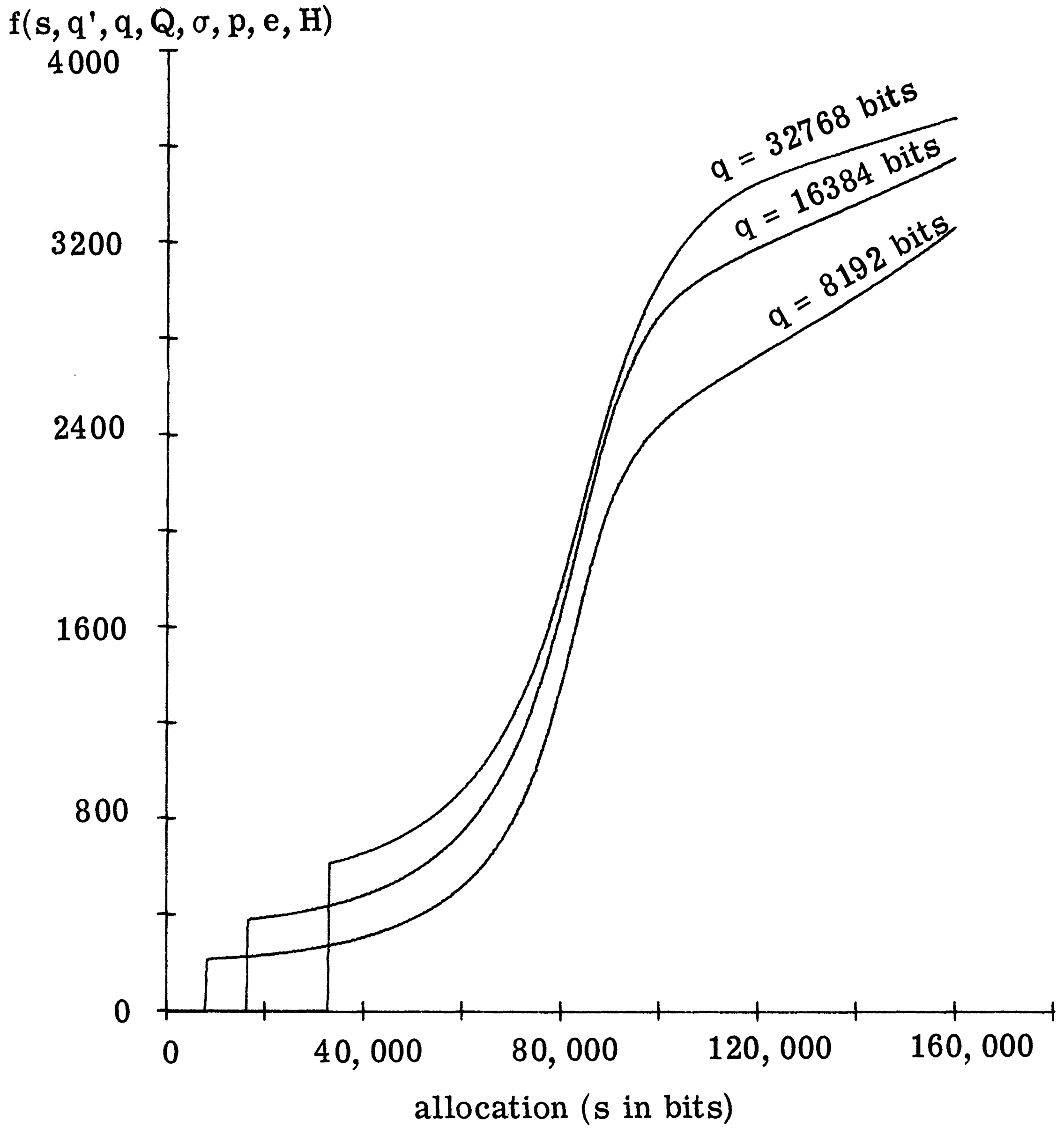
The variables σ and H cannot be described in terms of the working set concepts. Both σ and H describe what happens when we do allocate sufficient space for the working set.

Figures 2.17 and 2.18 show the effect of changing the word, q', and page, q, size. In general increased word size means a decrease in the lifecycle function. In the case of page size, an increased page size results in an increased lifecycle function. Great caution must be exercised in the interpretation of either of these results. In the case of page size, q, we are considering here pages which are exact divisors of the program size resulting in no wasted space. Also the three page sizes used here are considerably smaller than the allocation s. The effects of varying page size will be examined in considerable detail in the following pages. In the case of word size, q', the effects on the lifecycle function are representative but changes in word size are accompanied by other important system changes. For example we would expect the useful work per access and the number of accesses between IØ interrupts to change. The important point here is that the lifecycle function is a function of word size.



The Lifecycle Function Versus Allocation
With Variations in q'

Figure 2. 17



The Lifecycle Function Versus Allocation
With Variations in q

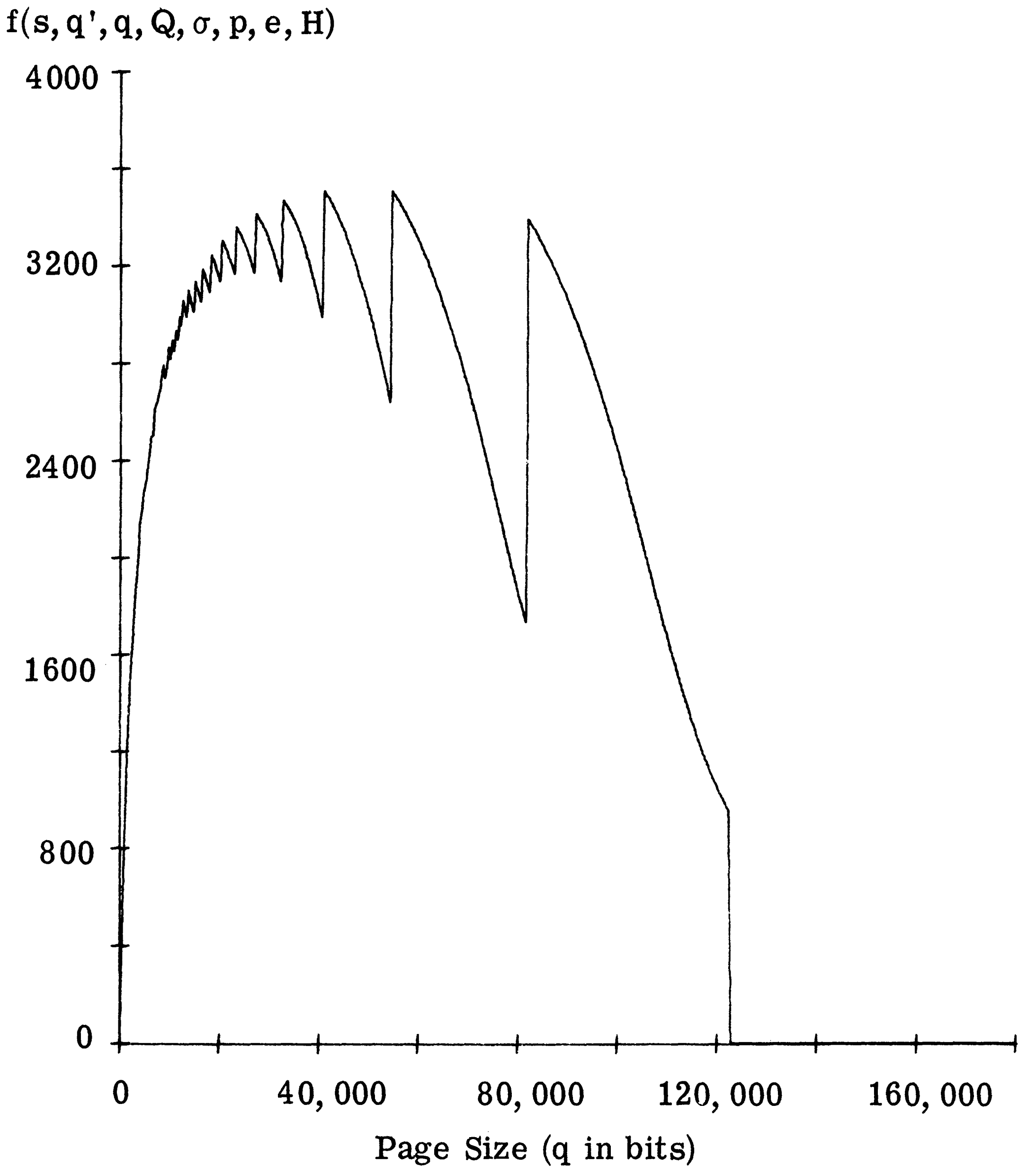
Figure 2. 18

2.8 Example of Model Behavior With Varying Page Size

We will now consider the effect of varying the page size, q, in detail. The graph of Figure 2.19 displays the lifecycle function as a function of page size. This represents the "standard" plot from which we will make variations. In our previous "standard" there was obviously no need to specify s as there is no need to specify q here. This "standard" here is identical to the previous standard except that we have chosen s = 122880. This means that the allocation is 3/4 of size of the program. This places us in a rather well allocated region or we might say that this is sufficient allocation to get the working set in core.

We will divide the graph of Figure 2.19 into three regions. On the left the life cycle function rises sharply with page size. We will refer to this region as the initial region. In the middle values of page size we see a sawtooth-like variation which increases with page size. We will refer to this region as the sawtooth region. On the right beyond a page size of 122880 bits we see a region where the value of life cycle function is suppressed. We will refer to this region as the suppressed region.

Beginning again on the left in the initial region we see a sharp rise in the value of the life cycle function with increasing page size. In this region of very small page sizes the life cycle function is dominated by the necessity to bring in an enormous number of



The Lifecycle Function Versus Page Size
for the "Standard" Case

Figure 2. 19

pages every time any new information is needed. In the case of a 32 bit page (the smallest value calculated here) the page and the word size are equal. Thus every time a new loop is started almost as many page faults will occur as there are words in the loop. In this case we have a 32 bit page, a 32 bit word and we access each loop 20 times, $\sigma = 20$. Thus we will get somewhat more than 20 accesses per page fault. The "somewhat more" is the result of finding desired pages in core from previous loops.

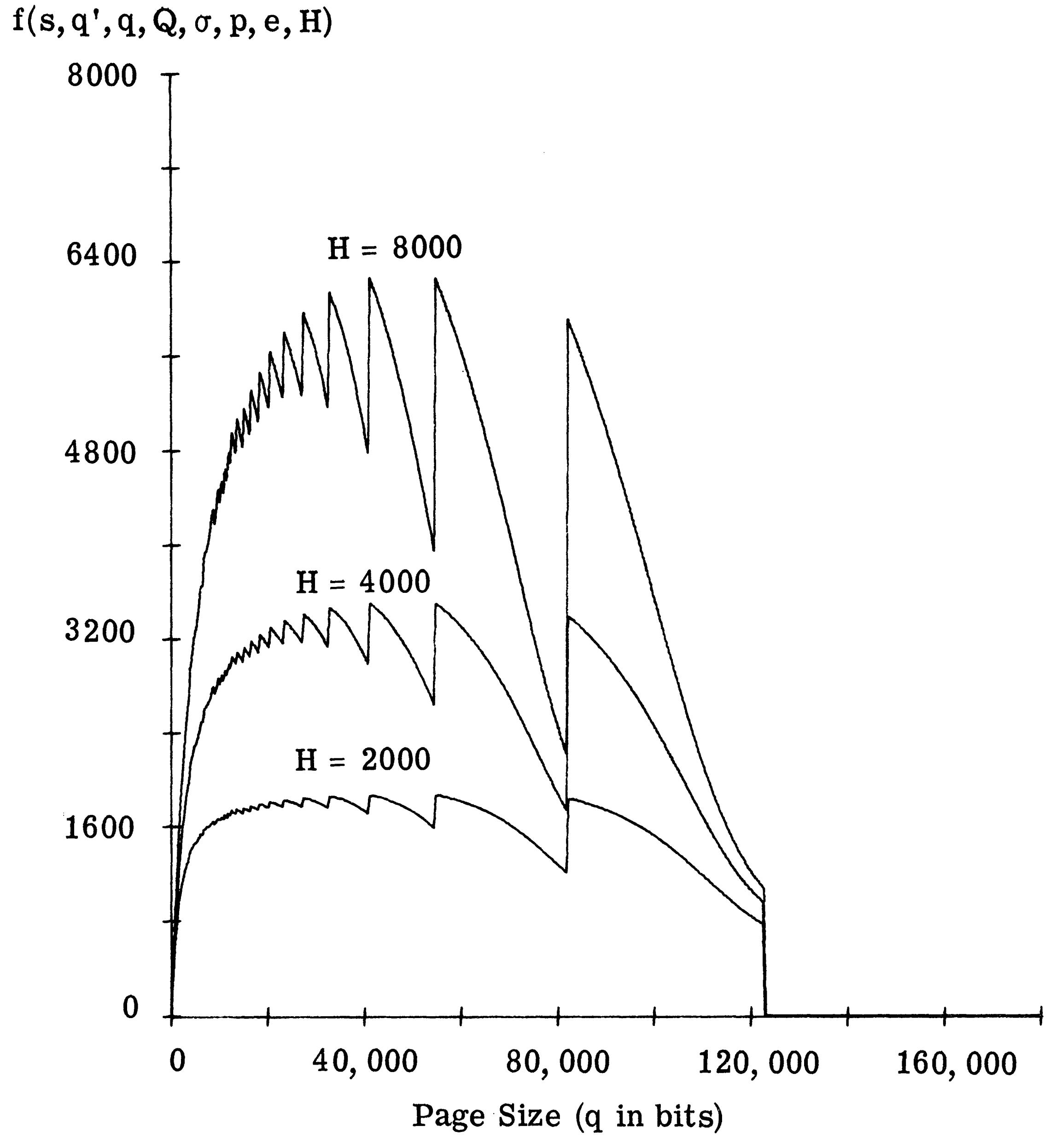
As we increase the page size farther the general shape of the curve begins to flatten out and a sawtooth variation appears. The sawtooth variation is caused by the discrete size of the pages. The peaks of the sawtooth represent page sizes for which some exact multiple of pages is equal to the size of the program. As we increase the page size from one of these peaks we see that the apparent size of the program increases. That is the program is spread out by the inappropriate page size and consumes more space. As we further increase the page size we reach another "perfect" fit and the value of the life cycle function again peaks.

If we increase the page size farther we reach a point where the page size exceeds the allocation. In this case this occurs at $q = 122880$ bits. From this point on the value of the life cycle function is suppressed and the program behaves as though it were severely under allocated. Notice that in these graphs the page size runs from the word size to 10% larger than the actual program.

Turning to Figure 2. 20 here we see the effect of varying the IØ interrupt rate. The "standard" curve is shown in the center where the value of H is 4000. In Figure 2. 21 we see the relative value of the life cycle function when $H = \infty$ and $H = 4000$ (the "standard" curve).

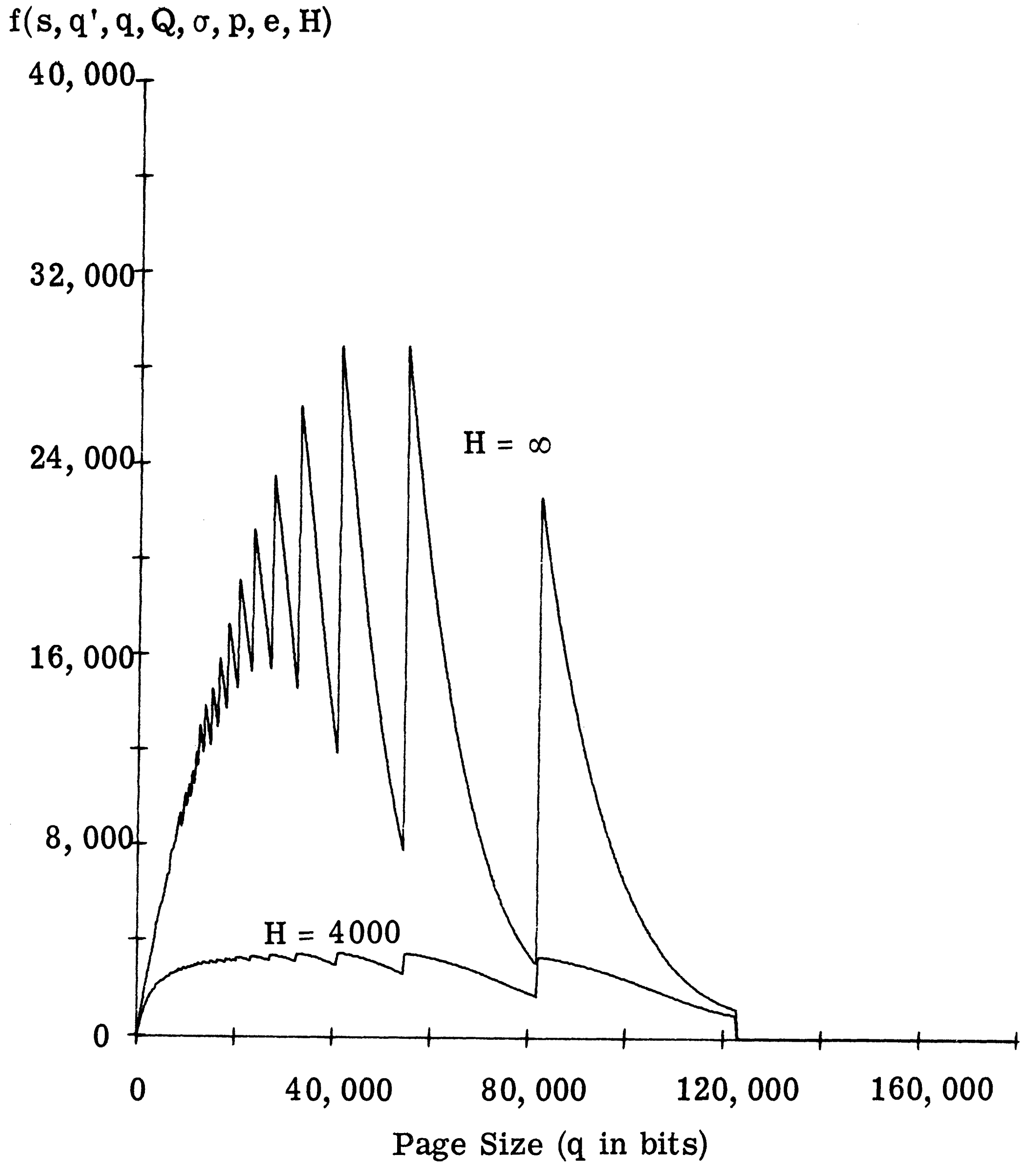
The curve of Figure 2. 21 with $H = \infty$ demonstrates an effect of page size which has been evident but not pronounced in the previous graphs. Let us consider the value of the life cycle function when the program size is multiple of the page size. That is at the peaks of the life cycle function. Notice that in the sawtooth region the value at the peaks first increases and then decreases. The increase can be explained easily as an extension of the behavior in the initial region, however the decrease has yet to be examined.

Considering this specific case we see that with a page size of 81920 bits we have a two page program. Likewise with a page size of 54614 bits we have a three page program and with a page size of 40960 bits we have a four page program. Notice that each of these page sizes "fits" the program either exactly or almost exactly. However in the case of the two page program only 1 of the two pages or 1/2 of the program can be allowed in core since the allocation is 3/4 of the total size of the program. Correspondingly,



The Lifecycle Function Versus Page Size
With Variations in H

Figure 2. 20



The Lifecycle Function Versus Page Size
for $H = \infty$

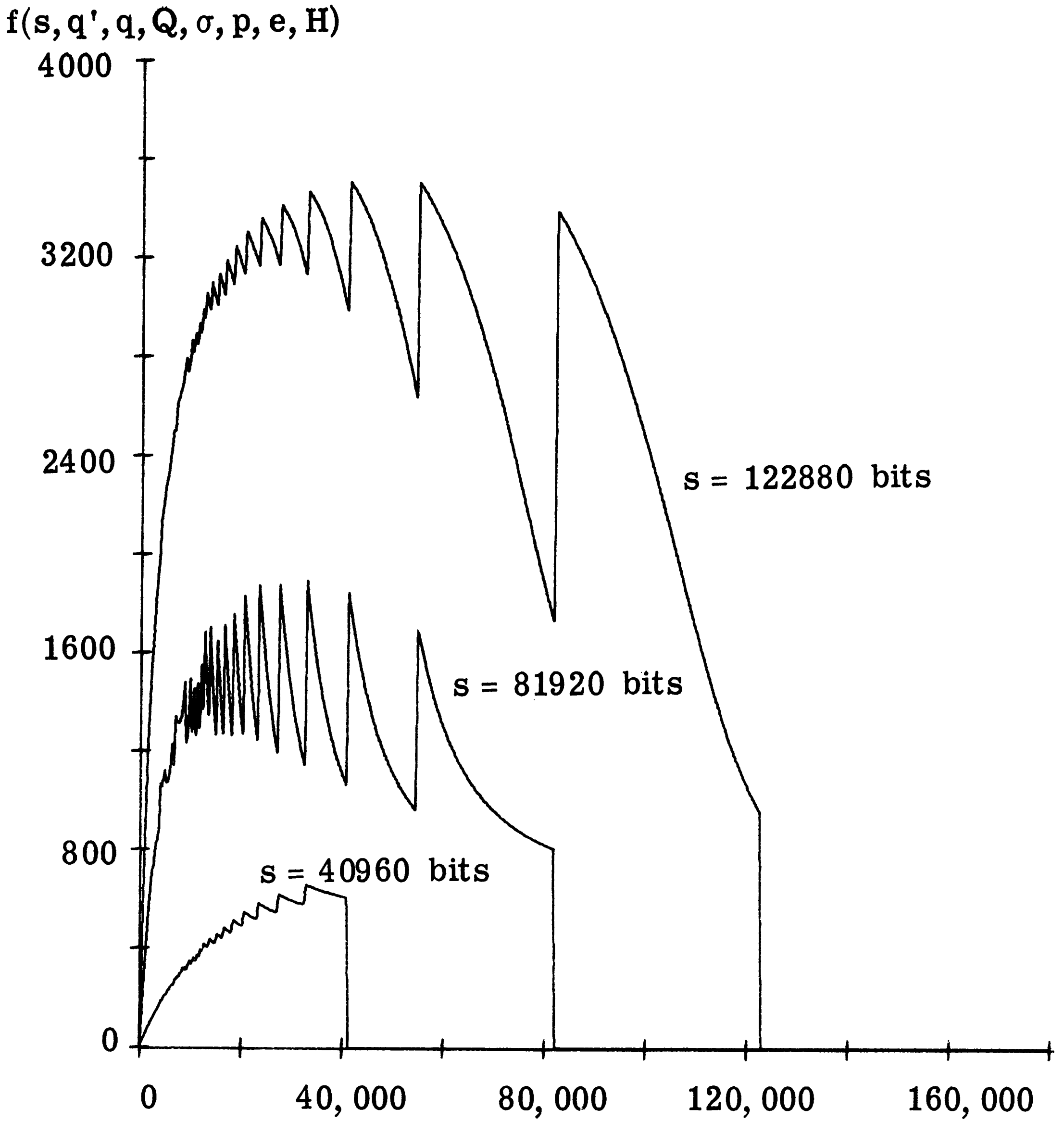
Figure 2. 21

in the case of the three page program we can keep two pages in core using 8/9 of the allocation and in the four page program we can use all of the allocated space. Thus in addition to the effects of "fitting" the program size we have some effects involving "fitting" the allocation.

Before going on to other graphs notice that the IØ interrupts are a major factor in the "standard" lifecycle function we have chosen and this will reduce some of the effects that we will observe as we proceed to examine changes in other variables.

In Figure 2. 22 we show the effect of varying the allocation s. Here our "standard" curve is the upper curve where s = 122880. There are several notable effects of changing s. First as we reduce the allocation we see the expected reduction in the value of the life cycle function. We also see that the suppressed region extends further to the left. That is, the page size exceeds the allocated space sooner. Notice that in the case of s = 40960 bits the program is under allocated and appears suppressed over the entire range of page sizes. The effect of not having the working set in core (i. e. most loops bigger than the allocation) is similar to the effect of the page size being larger than the allocation.

As we change the allocation we see a marked change in the curvature of the life cycle function between peaks in the sawtooth

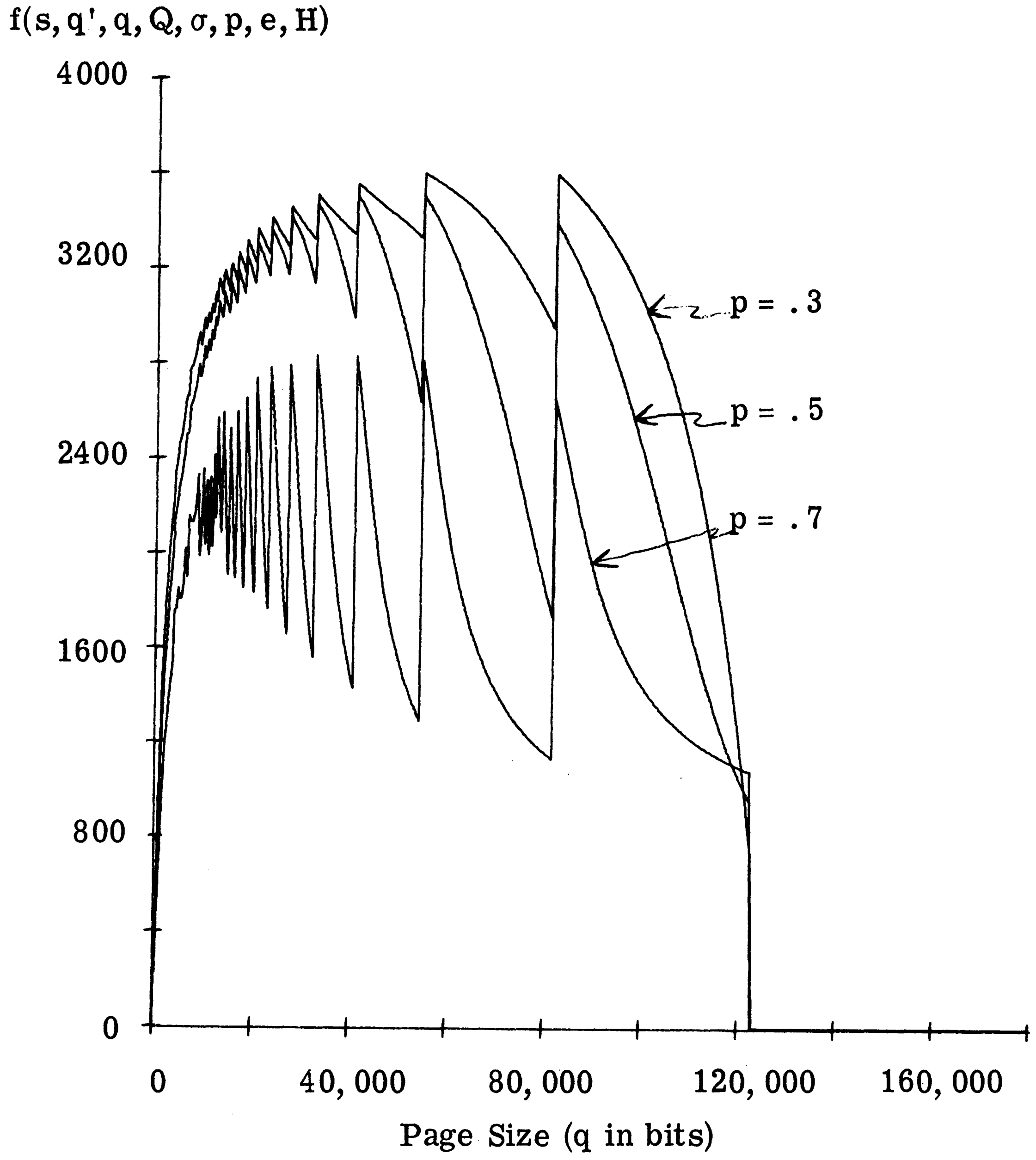


The Lifecycle Function Versus Page Size
With Variations in s

Figure 2. 22

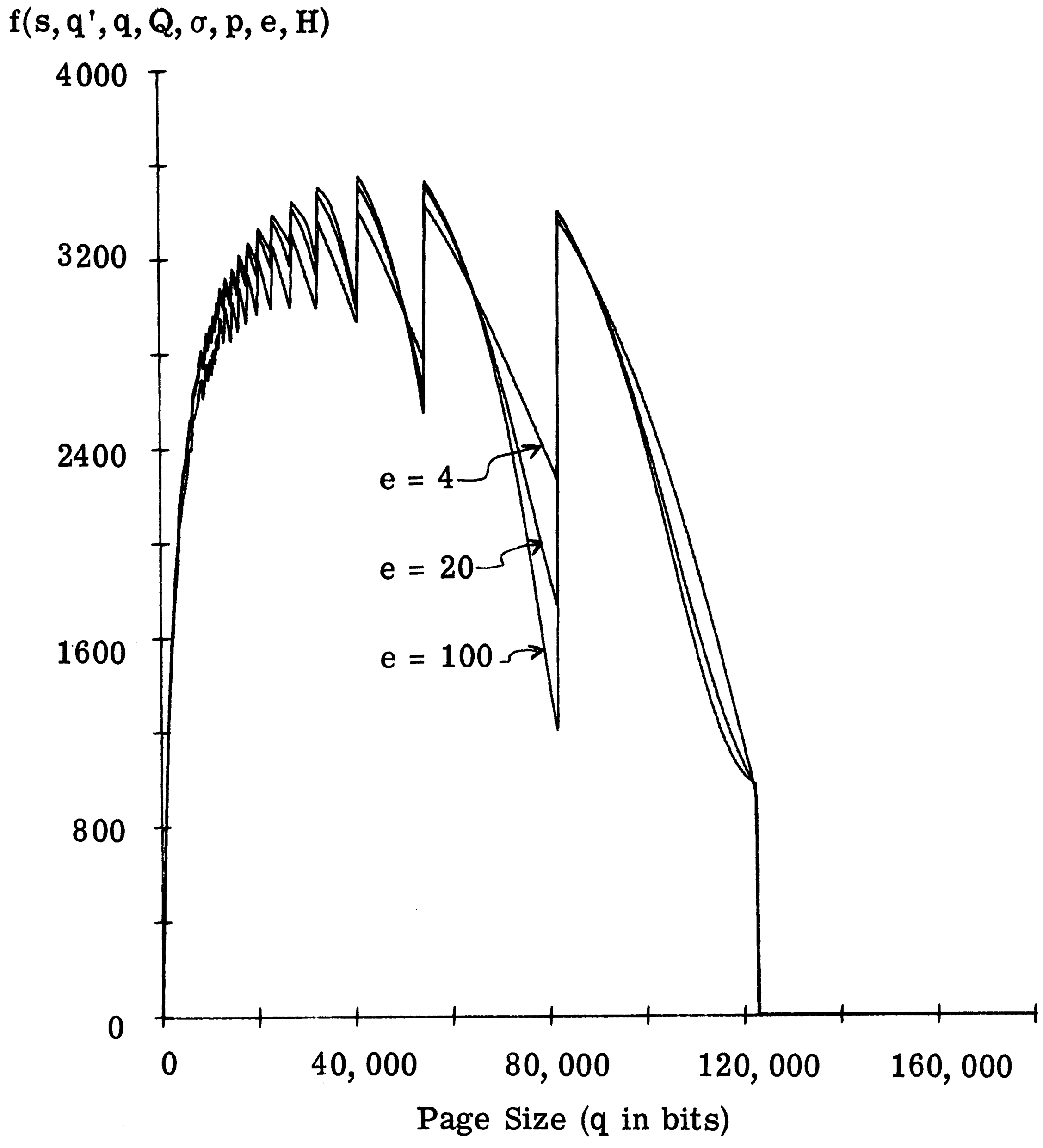
region. When we move from one peak to another increasing page size we are in effect expanding the apparent size of the program, an thus reducing the relative allocation. As we change the value of the absolute allocation s we are in effect picking a different point on the life cycle function versus s curve. We can think the variation which occurs between peaks in the sawtooth region as roughly corresponding to a variation in the allocation in the life cycle function versus s curve where an increased page size corresponds to decreased allocation. Thus when we choose s in a concave region of the life cycle function versus s curve we can expect a tendency for concave behavior between peaks in the sawtooth region and vice versa for s chosen in convex regions of the life cycle function versus s.

Turning to Figure 2. 23 we see a graph displaying three plots where we have varied the size of the mode of the loop length or in effect the working set. The "standard" curve is in the center where $p = .5$. Notice that the beginning of the suppressed region is the same for all cases. Here we see again large changes in the curvature. Notice also that the largest p, or mode of the loop size, represents a loop size .7 times the length of the program. In all of these cases we have an allocation of 122880 bits or .75 times the length of the program. Thus we do not show a sharply under allocated situation.



The Lifecycle Function Versus Page Size
With Variations in p

Figure 2. 23



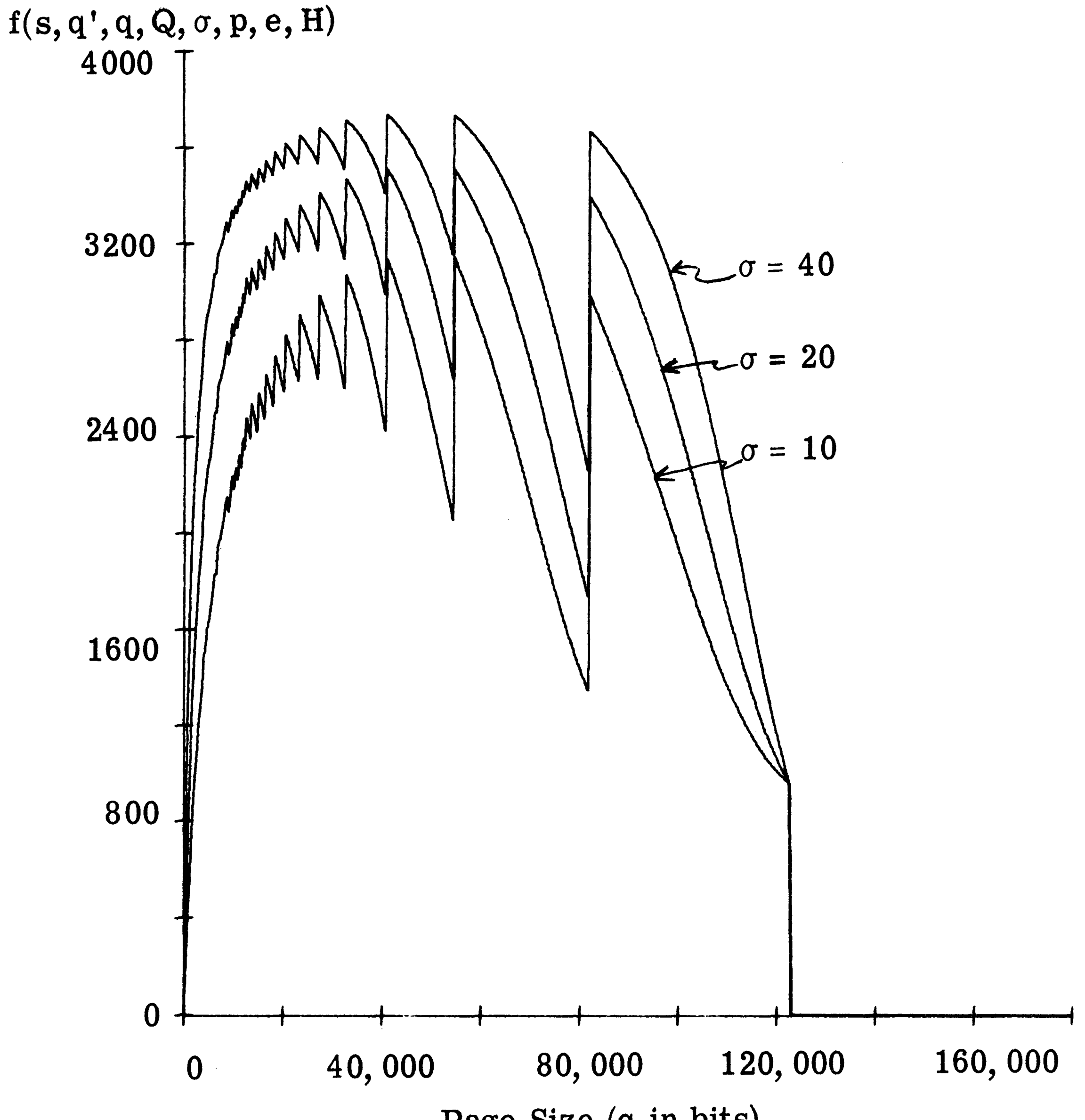
The Lifecycle Function Versus Page Size
With Variations in e

Figure 2. 24

In Figure 2. 24 we show the effects of changing e. Here the "standard" curve is shown with e = 20. It is interesting to compare these curves of Figure 2. 13. Note that the cross over points between peaks in the sawtooth region of Figure 2. 24 correspond to the cross over point in Figure 2. 13.

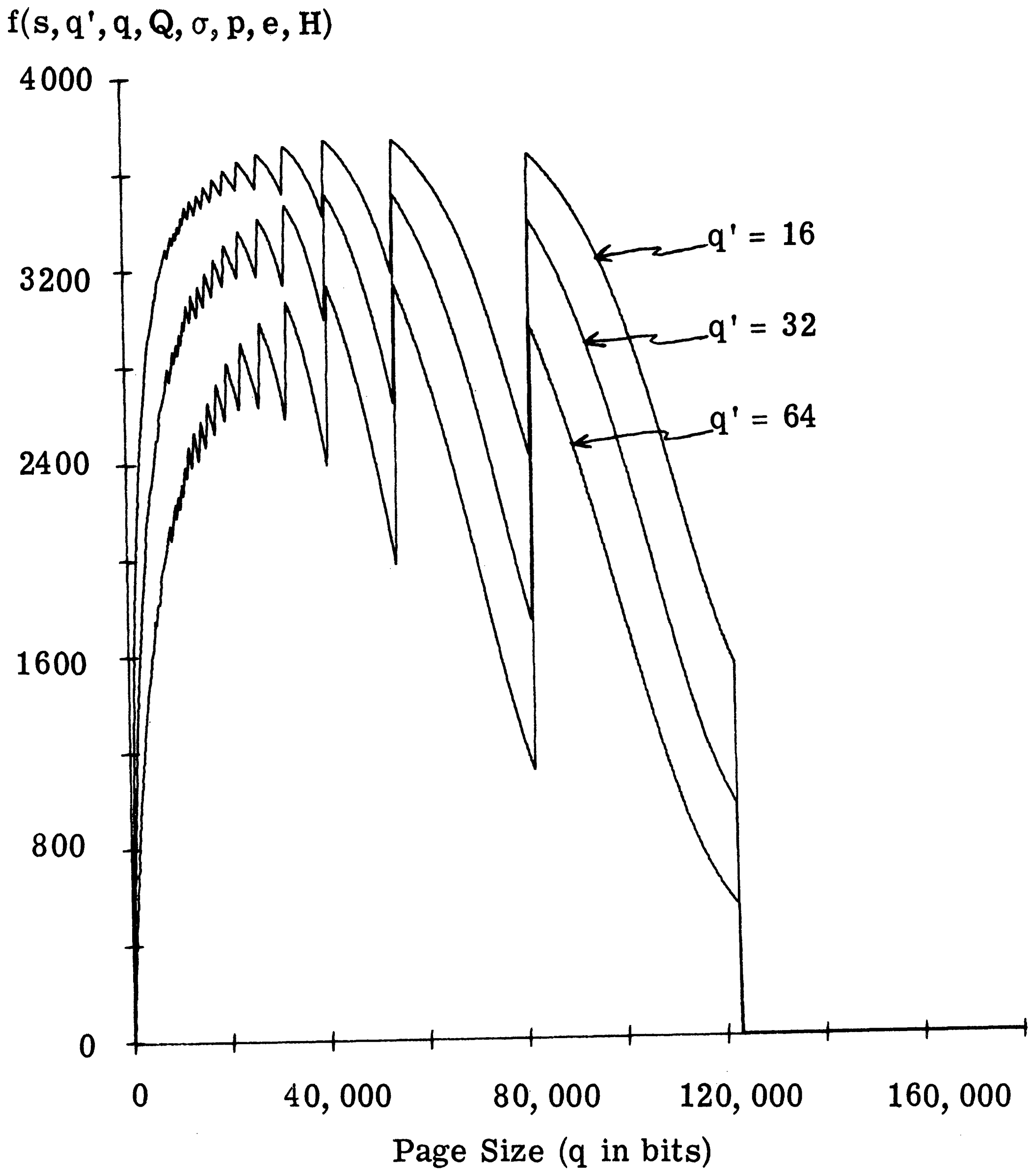
In Figure 2. 25 we show the effect of varying σ. The curves are rather simple and show an increase over all regions except the suppressed regions.

Figure 2. 26 shows the effect of varying the word size. Changes in word size cause a change in the lifecycle function which is rather uniform over all values of page size.



The Lifecycle Function Versus Page Size
With Variations in σ

Figure 2. 25



The Lifecycle Function Versus Page Size

With Variations in q'

Figure 2. 26

Chapter 3
Macroscopic Model

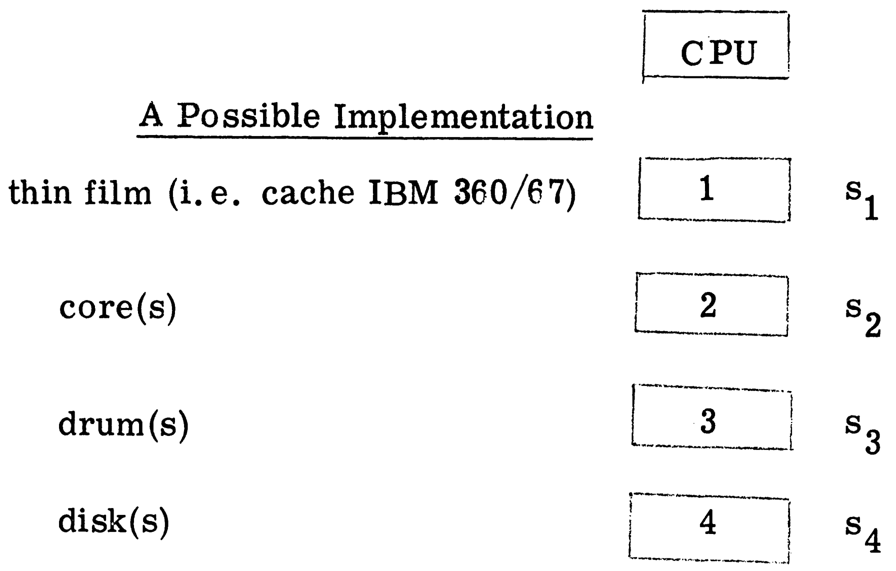
The purpose of this chapter is to discuss a macroscopic model which relates the program behavior discussed in the previous chapter to storage delays, page size, data routings, CPU execution rate and the number of programs being multiprogrammed. Taking all of these factors into consideration we will develop a solution for the average system throughput, that is, the average rate at which user induced accesses are processed.

3.1 The N Level Hierarchy

We will now turn to considering the meaning of the lifecycle function in an N level hierarchy. To begin we will consider only variations in the allocation s and thus we will write the lifecycle function as f(s). We will also ignore IØ interrupts for the time being thus we may assume $H = \infty$.

When treating an N level hierarchy we see that the case of $N = 1$ is degenerate and the case of $N = 2$ is the type of system we have assumed up to this point. Thus our focus here will be on $N > 2$.

For examples we will work with $N = 4$. The choice of $N = 4$ is attractive in that it is large enough to demonstrate all the complexities of hierarchies where $N > 4$ and yet it is small enough to be convenient for discussion. Let us consider the 4-level hierarchy shown in Figure 3.1.



A Four Level Hierarchy

Figure 3.1

Here we have shown a CPU and 4-levels of storage and on the left a feasible set of devices is given. It should be clear that program information will migrate up and down in these 4 levels in much the same manner as a 2-level system.

3.2 Storage Allocation

When considering a system with $N > 2$ certain new questions arise concerning the allocation of storage space. First we must allocate storage at several levels rather than just core and drum. We must be more precise in stating exactly what is stored where.

We may resolve the first question as it relates to the model by simply supplying the variable s with a subscript. Thus we will define s_i as the storage allocated to a process at level i. Figure 3.1 shows this vector as s_i's associated with the 4-level hierarchy.

The second question is a bit more complex. Exactly what is stored where? We will follow the following convention: If the current copy of some process information is resident at level k then a specific space must be reserved for this information at all lower levels (i.e. levels i where $i > k$). This space may or may not contain a current copy of the process information.

As a direct result of this convention we may state:

$$s_i \leq s_{i+1} \tag{3.1}$$

and

$$s_n \geq Q \tag{3.2}$$

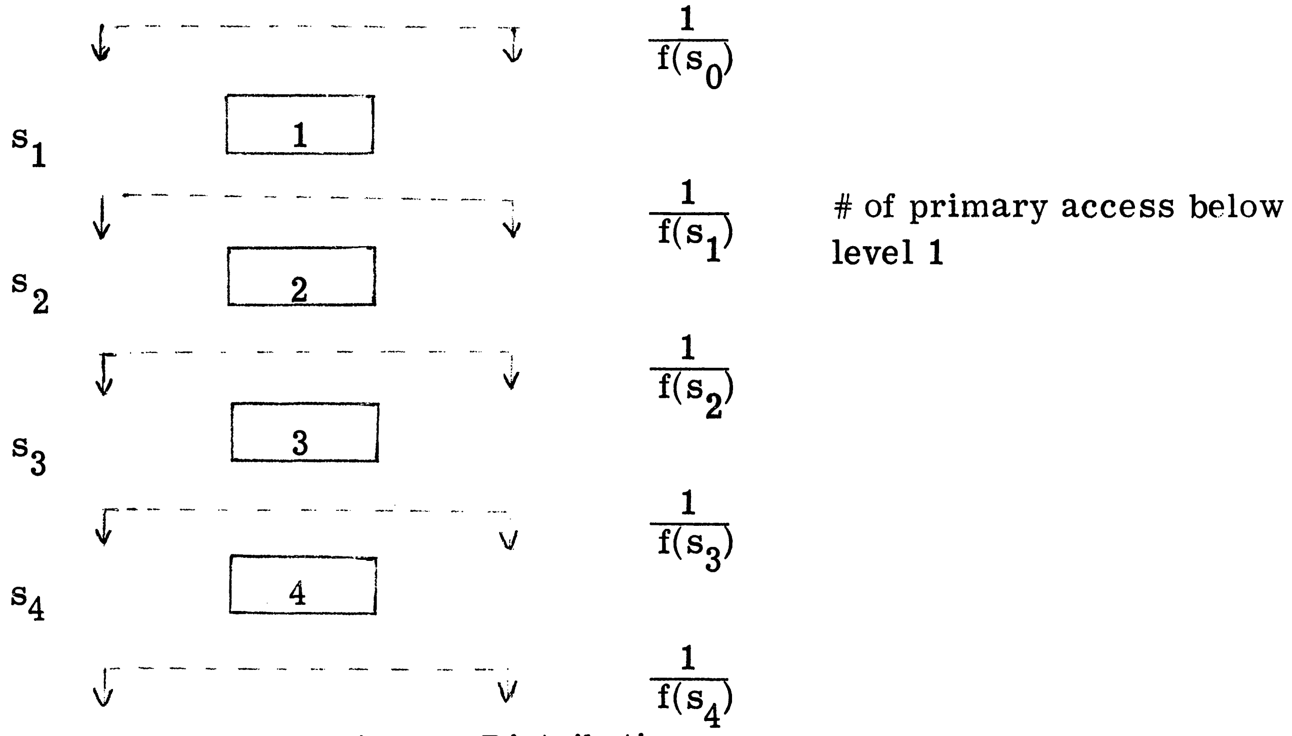
where Q = total size of a user program or process.

Notice that s_i represents not only the storage allocation at level i but also the amount of unique current information stored at and above level i. This, of course, is not the complete answer to what is stored where but it will suffice for now.

Relative Access Rates

In a 2-level hierarchy a single page fault always generates a one page transfer from the lower to the upper level. In a multi-level system a single page fault may cause many transfers to occur. We must then be more precise in defining the relationship between page faults and data transfers. We will introduce the notion of user program accesses. User program accesses are generated by storage accesses in executing user programs. The user program access will always be to the highest level of the hierarchy containing a current copy of the required information. When a user program makes an access the resultant transfer of information may be very complex involving many levels and both upward and downward transfers. A user program access as we will use it here does not imply a specific transfer but rather simply indicates that the user program requires certain information. We will discuss the transfers which result from a user program access in detail later in this chapter.

Our immediate concern here will be the fraction of user program accesses made to individual levels of the hierarchy and how this is related to the lifecycle function of the user program. Let us begin by determining what fraction of all accesses made by a process are made to level 1. Recalling that s_1 is the storage allocation at level 1 from the definition of f(s) (recalling that we are representing the entire lifecycle function as f(s)) it is clear that the average number of accesses required to produce a primary access below level 1 will be $f(s_1)$. Thus the fraction of accesses which fall below level 1 will be $\frac{1}{f(s_1)}$ and the fraction of access to level 1 is simply $1 - \frac{1}{f(s_1)}$. This general approach may be taken at all levels.



Access Distribution

Figure 3.2

Since s_i is not only the storage allowed at level i but the total unique storage allocated at level i and above, the fraction of accesses falling below level i will be $\frac{1}{f(s_i)}$. This relationship is depicted in Figure 3.2. The fraction of primary accesses, to any given level i may be expressed simply as

$$u_i = \frac{1}{f(s_{i-1})} - \frac{1}{f(s_i)} \text{ for } i=1,2,\ldots,N \qquad (3.3)$$

where

$$\frac{1}{f(s_0)} \equiv 1$$

At this point we have expressed the fraction of primary accesses to each level in the hierarchy as a function of the storage allocation. More precisely the fraction of primary accesses to any specific level is only a function of the allocation at that level and the adjacent higher level.

Paging

It is our intention to extend the basic concept of paging to an N-level hierarchy. Such an extension is not trivial and requires an explanation. What follows is a description of how paging is carried out in such an extended system and how this system is modeled.

Record Size Constraints

Up to this point we have ignored the effects of record (or page) size. Here we will consider how record sizes affect data transfers in the system.

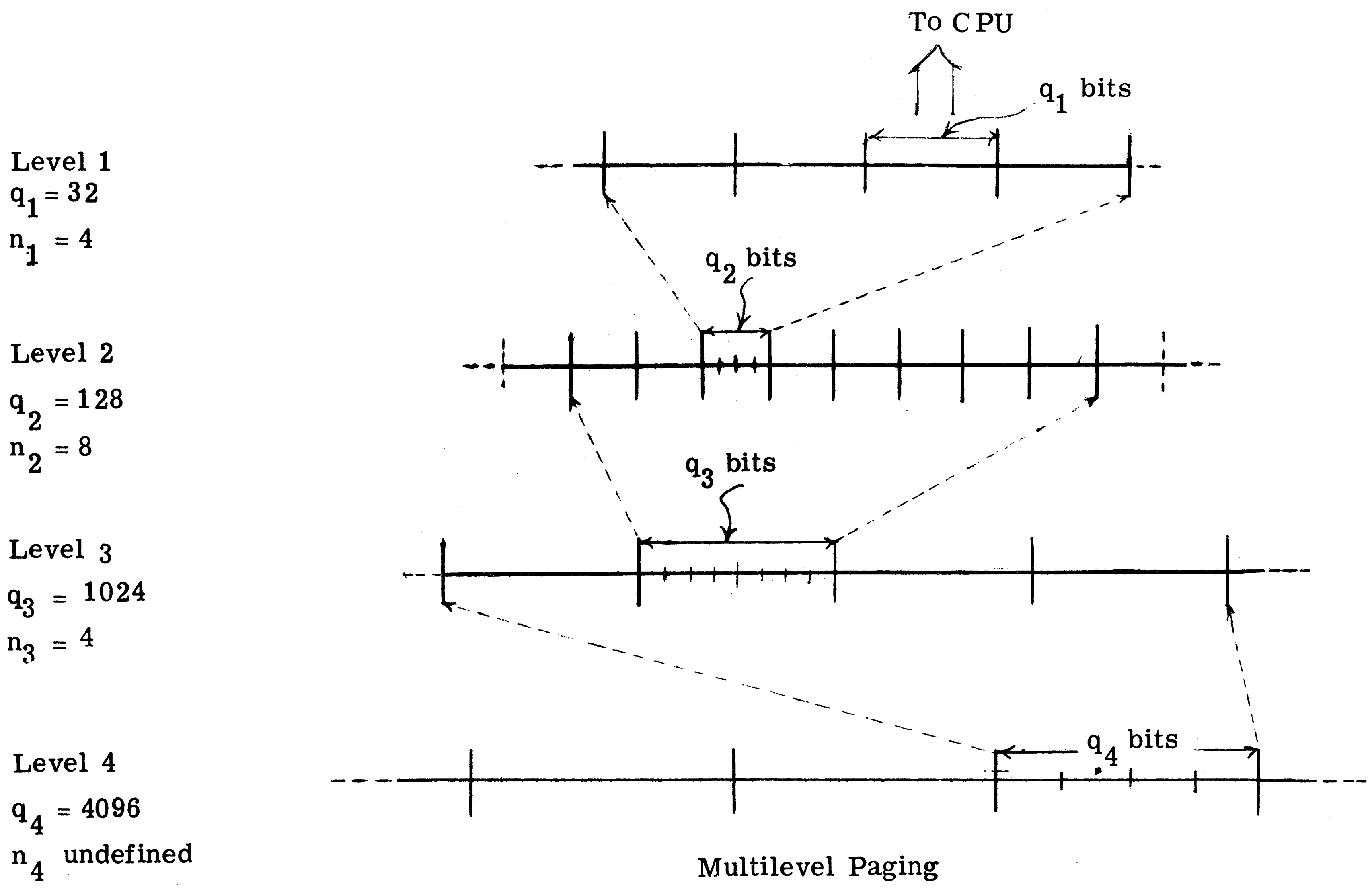
We will begin by assigning a variable q_i to each level of the hierarchy and enforcing the following constraint:

$$q_{i+1} = n_i \, q_i \text{ where } n_i = \text{positive non-zero integer.}$$

In our model we will treat the q_i's as variables and they may in turn control the paging behavior of the system.

Information Movement

We will begin to examine the paging process by considering an example. Figure 3.3 shows a 4-level hierarchy along with a specific set of q_i's and n_i's. Now suppose an executing process attempts to read a word of information from memory and the most convenient copy is at the bottom of the hierarchy, i.e. level 4. This is simply



Multilevel Paging

Figure 3.3

a user program access to level 4 of the hierarchy. It is then necessary to "page this information upward in the hierarchy. "Paging" may be carried out as follows.

1. A copy of the record containing the accessed word at level 4 of size q_4 = 4096 bits will be read from level 4 into level 3. At level 3 this will form 4 records (n_3) of 1024 bits (q_3) each.
2. A copy of the level 3 record (1024 bits) containing the accessed word is read from level 3 to level 2. At level 2 this will form 8 records (n_2) of 128 bits (q_2) each.
3. A copy of the level 2 record (128 bits) will be read from level 2 to level 1. At level one this will form 4 (n_i) records (or words) of 32 bits (q_1) each.
4. A copy of the 32 bits (q_1) accessed is read from level 1 to the CPU. (Note: the CPU will be referred to where convenient as level 0.)

User program accesses to levels above 4 may be paged in much the same manner with the exception that lower levels are not disturbed.

There are several aspects to this description of paging one of which will be taken as <u>a fixed part of the model</u> and others which may vary. As to the fixed part of the model we will assume that the final disposition of all the upward paged information will be the same as the final disposition in the example. That is each level i above the level of user program reference will acquire n_i new records of size q_i. These n_i records will contain the information contained in the single record of level i + 1 which includes the primary user program access.

Notice that we are not saying anything about how or what path the information follows in order to reach its final goal. We are only fixing the eventual distribution of the information involved in a paging operation and even this remains a function of the q_i's.

Let us consider a few examples which demonstrate different data paths. We may for instance choose to transfer q_i bits directly from the level of user program access to the CPU (level 0) eliminating the transfer from level 1 to CPU leaving other transfers the same. This would be similar to the cache core relationship of the IBM 360/85 [18]. Another possibility would be to transfer a record from the level of primary access to level 1 and then transferring the appropriate parts of that record to the other levels. This in effect uses level 1 as an interlevel buffer. This may or may not be a wise choice but it is interesting to note that the architecture of the IBM 360/67 [30] is limited to this kind of transfer pattern. It is important to note that the question of what data paths and combinations of transfers are best can be explored since this will be a variable in the model. variable portion of the model is discussed later.

We will now expand our representation of the lifecycle function from $f(s_i)$ to $f(s_i, q_{i+1})$ including the pages or record size as a variable. Applying this new form of the lifecycle function the fraction of primary accesses to any given level i (u_i) previously given in Equation 3. 3 we have

$$u_i = \frac{1}{f(s_{i-1}, q_i)} - \frac{1}{f(s_i, q_{i+1})} \text{ for } i=1, 2, \dots, N \qquad (3.4)$$

where

$$\frac{1}{f(s_o, q_1)} \equiv 1$$

and

$$\frac{1}{f(s_N, q_{N+1})} \equiv 0$$

3. 3 Multiprogramming

Here we will consider the formal representations required to represent a program in a multiprogrammed environment. We will begin by providing the lifecycle function with an index allowing each process a distinct lifecycle function

$$f_h(s_{i,h}, q_{i+1})$$

where h is an integer that satisfies $1 \leq h \leq M$ representing M distinct processes and i is an integer that satisfies $1 \leq i \leq N$ representing N levels of storage and where

$s_{i,h}$ = Total quantity of user program h information at and above level i

Notice that in addition to giving each process a distinct lifecycle function $f_h(\cdot)$ we now represent the allocation of each process at each level in the hierarchy $s_{i,h}$. This allocation can be conveniently expressed as an $N \times M$ matrix $\overline{S}$.

$$\overline{S} = \begin{bmatrix} s_{1,1} & s_{1,2} & - - - & s_{1,M} \\ s_{2,1} & s_{2,2} & - - - & s_{2,M} \\ \vdots & \vdots & & \vdots \\ s_{N,1} & s_{N,2} & - - - & s_{N,M} \end{bmatrix} \qquad (3.5)$$

The elements in the kth row of the matrix represent allocations at the k-level in the hierarchy. Elements in the kth column represent the allocation of the kth user program. Thus we may write

$$S_i = \sum_{j=1}^{M} s_{i,h} \qquad (3.6)$$

where s_i is the total storage allocation at level i.

From Equation 3.1 we may write

$$s_{i,h} \leq s_{i+1,h} \qquad (3.7)$$

and

$$s_{N,h} \geq Q_h \qquad (3.8)$$

where Q_h is the total size of the process h.

We may also redefine u_i so that individual processes may be identified. Thus:

$$u_{i,h} = \frac{1}{f(s_{i-1,h}, q_i)} - \frac{1}{f_h(s_{i,h}, q_{i+1})} \qquad (3.9)$$

where

$$\frac{1}{f_h(s_{o,h}, q_i)} \equiv 1$$

and

$$\frac{1}{f_h(s_{N,h}, q_{N+1})} \equiv 0$$

This comes directly from Equation 3.4 and represents the fraction of primary accesses to any given level i by process h.

Later in this chapter we will treat the first L levels of storage in a special manner. This will have an effect on the value of u_i and $u_{i,h}$ in the first L+1 levels of storage. We mention this here only to point out that Equations 3.4 and 3.9 will be slightly modified at a later time.

<u>IØ Interrupts</u>

Thus far we have ignored IØ interrupts in N-level hierarchies. When IØ interrupts are to be considered they will be treated as accesses to the Nth level in the hierarchy where the N level will act as a pseudo storage level. Notice that if we constrain the allocation in the (N-1)th level of storage such that

$$s_{N-1,h} \geq Q_h \qquad (3..10)$$

the fraction of "accesses" to the Nth level ($u_{i,h}$ of Equation 3.9 or u_i of Equation 3.4) will be the fraction of all interrupts (page fault and IØ) which are IØ interrupts. We will consider the role of the pseudo N^{th} level of storage as we proceed.

3.5 Interlevel Data Traffic

We will now consider a mathematical model which relates user accesses and the flow of data between levels. We will begin by defining $t_{j,k,i,\ell}$ an element of the 4-dimensional array $\overline{T}$.

$t_{j,k,i,\ell}$ = the number of records of size q_k read or written at level j as a result of a user read ($\ell=0$) or write ($\ell=1$) to level i. (3.11)

where j, k, i, and ℓ are integers satisfying

$$1 \leq j \leq N$$

$$1 \leq k \leq N$$

$$1 \leq i \leq N$$

and

$$\ell = 0 \text{ or } 1$$

Let us consider an example. Earlier we considered a specific set of transfers which would achieve the necessary disposition of information in a 4 level hierarchy. This pattern of transfers was shown in Figure 3.3 and described in detail in the associated text.

Notice that all transfers in this scheme are carried out between adjacent levels. Let's begin by considering the traffic caused by a user program access to level 4. Figure 3.4 shows the transfers which occur. To the left, with solid lines, of the hierarchy the "page up" transfers are shown as described earlier. On the right the page down transfers are shown using dashed lines. The page down transfers are dictated by the need to maintain a balance of transfers in or out of any level.

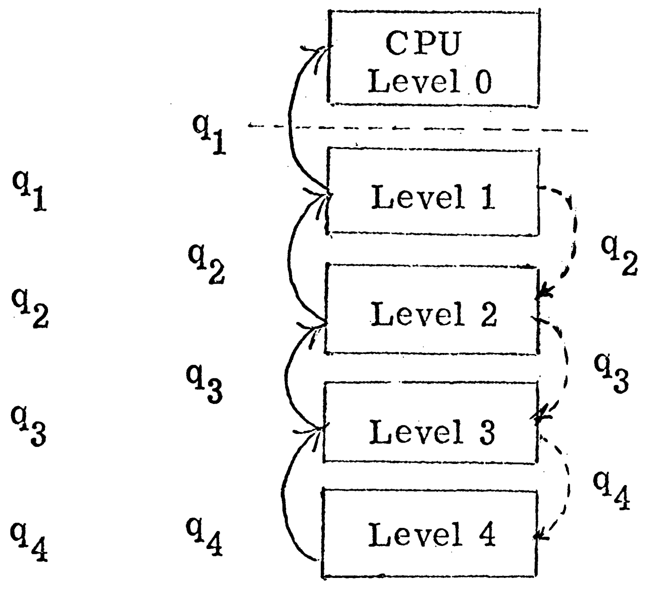


Figure 3.4 Traffic - Primary Read to Level 4

Here we are considering a user read to level 4. Thus we have fixed the i and ℓ of $t_{j,k,i,\ell}$ to 4 and 0 respectively. The nonzero values of $t_{j,k,4,0}$ are

<u>Level 1 traffic</u> (j = 1)

$t_{1,1,4,0} = 1$

$t_{1,2,4,0} = 2$

indicating 1 record of size q_1 and 2 records of size q_2

<u>Level 2 traffic</u> (j = 2)

$t_{2,2,4,0} = 2$

$t_{2,3,4,0} = 2$

Indicating 2 records of size q_2 and 2 records of size q_3.

<u>Level 3 traffic</u> (j = 3)

$t_{3,3,4,0} = 2$

$t_{3,4,4,0} = 2$

<u>Level 4 traffic</u> (j = 4)

$t_{4,4,4,0} = 2$

Noticing that fixing the values of i, the level of the user access and ℓ, indicating a read or write user access leaves us with an N by N matrix $\bar{t}_{i,\ell}$ of terms

$$t_{j,k} = t_{j,k,i,\ell} \tag{3.12}$$

There will be 2N such $\overline{t}_{i,\ell}$ matrices each of which describes the traffic in the system which results from each of the 2N possible user access types.

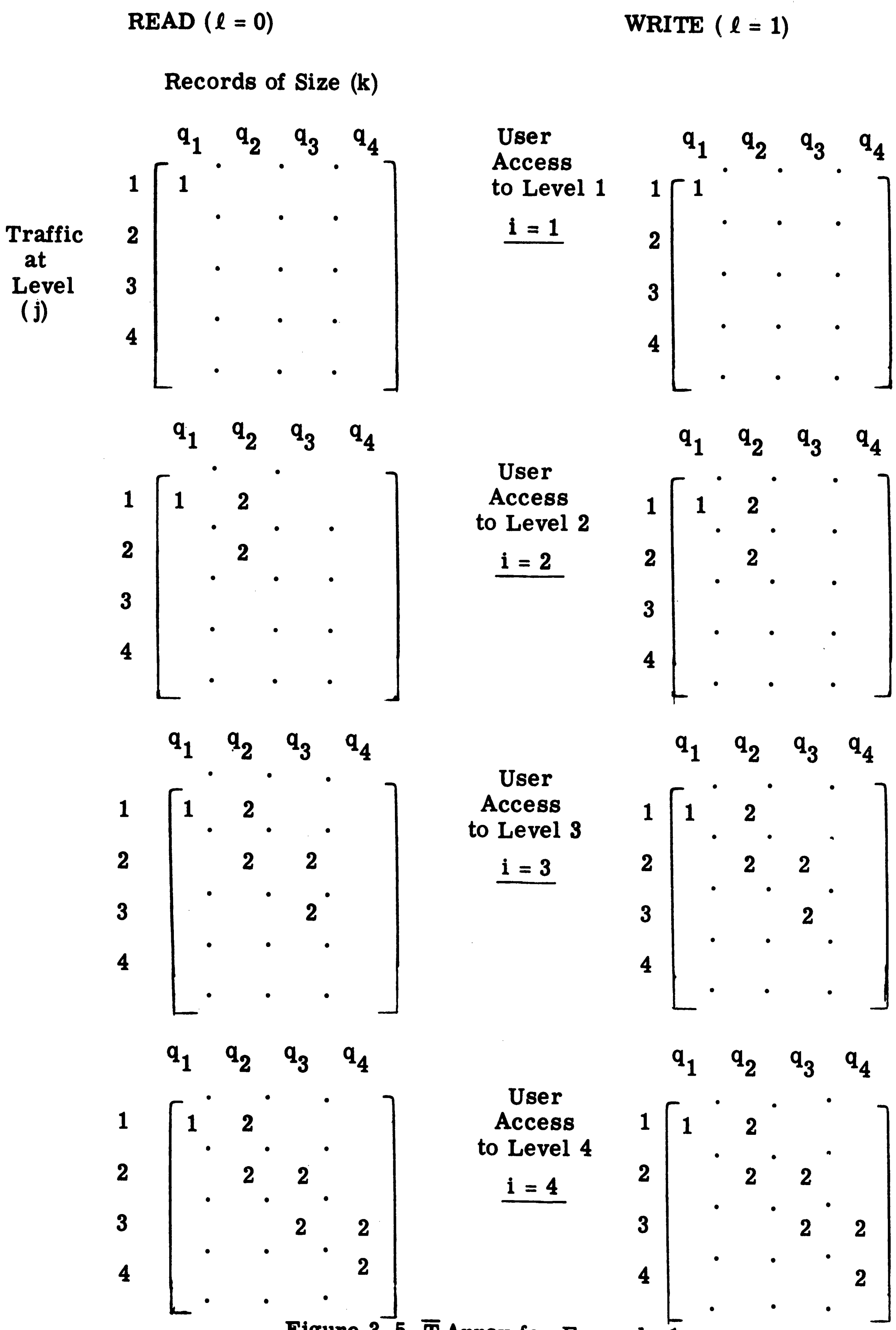


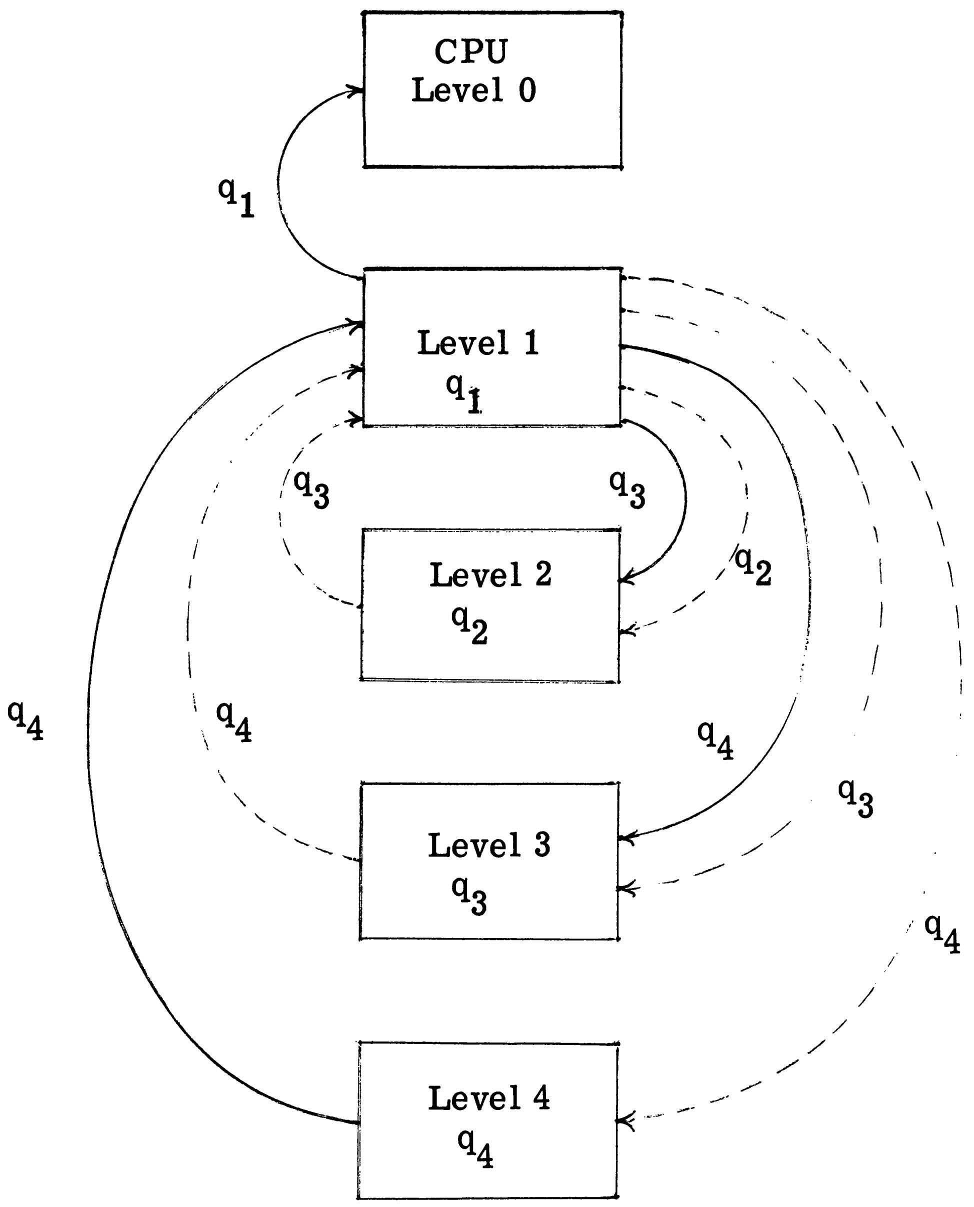
Figure 3.5 $\overline{T}$ Array for Example 1

In the case of our example we have 4 levels (N=4) and therefore 2N or 8 $\overline{t}_{i,\ell}$ matrices each of which is 4 by 4 or contains 16 $t_{j,k}$ elements. These matrices for the example in question are shown in Figure 3. 5. The elements, $t_{j,k}$, of these matrices may be determined by simply drawing the transfers which result from a specific user read or write as done in Figure 3. 4 and then summing the number of records of a given size q_k read or written at a given level j giving us the element $t_{j,k}$. The 8 $\overline{t}_{i,\ell}$ matrices for this example are shown in Figure 3. 5.

It is noteable that in this example and several to follow there is no difference between a user read or write. We will consider an example later where this is not the case.

Let us consider a second example. Here we will consider a case where all data transfers must be in or out of level 1. This situation was mentioned earlier.

Let us begin by looking at the transfers which result from a primary read to level 4. These transfers are shown in Figure 3. 6 It is important to understand clearly what is meant by page up and page down transfers when examining Figure 3. 6 since some of the page up transfers are actually moving down in the hierarchy and some page down transfers are moving up. Page up transfers are those transfers which occur directly as a response to a user access. Page down transfers occur in response to the need to maintain a balance of incoming and outgoing records at each level. In a user read to level 4 the following transfers take place:



Page Up Transfers ————————

Page Down Transfers ------------

Transfer Pattern 4-Level Hierarchy
for
User Induced Level 4 Read

Figure 3.6

Page up transfers

1. A copy of the record at level-4 containing the desired information is read from level 4 to level 1 a transfer of q_4 bits.
2. A copy of the level 1 record containing the desired information is read from level 1 to level 0 (the CPU) A transfer of q_1 bits.
3. A copy of the level 3 record containing the desired information is read from level 1 to level 2. A transfer of q_3 bits.
4. A copy of the level 4 record containing the referenced information is read from level 1 to level 3. A transfer of q_4 bits.

Note:

a. 2, 3, and 4 above may occur in any order or in parallel.

b. Only space for a level 2 record is allocated at level 1. That is after serving as a buffer for the above transfers only the level 2 record containing the primary reference is preserved.

Page down transfers

5. A level 4 records is read from level 3 to level 4 passing through level 1.

   a. A q_4 bit transfer from level 3 to level 1.

   b. A q_4 bit transfer from level 1 to level 4.

6. A level 3 record is read from level 2 to level 3 passing through level 1.

   a. A q_3 bit transfer from level 3 to level 1.

   b. a q_3 bit transfer from level 1 to level 3.

7. A level 2 record is read from level 1 to level 2.

Figure 3.7 shows the transfers for user reads to levels 1 through 3. Here as before page up transfers are indicated by solid lines and page down transfers are shown as dotted lines.

The diagrams for user writes would be the same except for a transfer from level 0 to level 1 of size q_1 rather than the q_1 transfer shown. The traffic matrices for this example are shown in Figure 3. .

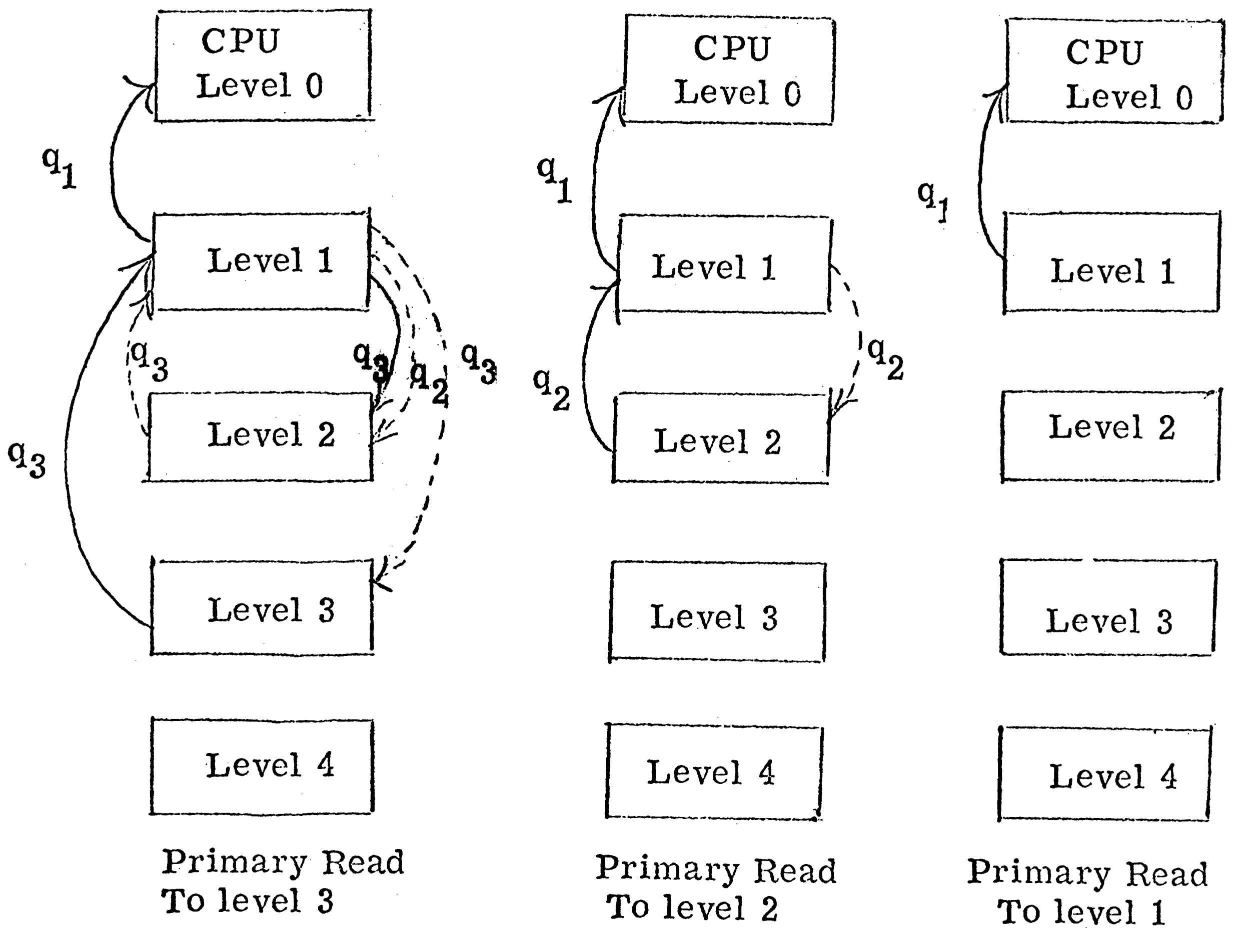
It is interesting to note that for a hierarchy limited to 2 levels both of the examples that we have discussed are identical. However for the N-level hierarchy where N is large the traffic in and out of level 1 is much greater in the second example.

3.6 Primary and Secondary Levels of Storage

There is a distinction which can be made between the levels of storage which is only meaningful in a multiprogrammed environment. This distinction is based on the question of how far down in the storage hierarchy can a user program make accesses without loss of the CPU. We will refer to the upper levels of the hierarchy as primary levels and the lower levels as secondary.

We will define

K = Number of primary levels of storage



Page Up ________

Page Down --------

Traffic Diagrams

Figure 3.7

READ ($\ell = 0$) **WRITE ($\ell = 1$)**

Records of Size (k)

Traffic at Level (j)

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & & & \\
2 & & & & \\
3 & & & & \\
4 & & & &
\end{array}
\qquad
\begin{array}{c}\text{User Access}\\\text{to Level 1}\\\underline{i = 1}\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & & & \\
2 & & & & \\
3 & & & & \\
4 & & & &
\end{array}$$

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 2 & & \\
2 & & 2 & & \\
3 & & & & \\
4 & & & &
\end{array}
\qquad
\begin{array}{c}\text{User Access}\\\text{to Level 2}\\\underline{i = 2}\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 2 & & \\
2 & & 2 & & \\
3 & & & & \\
4 & & & &
\end{array}$$

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 1 & 4 & \\
2 & & 1 & 2 & \\
3 & & & 2 & \\
4 & & & &
\end{array}
\qquad
\begin{array}{c}\text{User Access}\\\text{to Level 3}\\\underline{i = 3}\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 1 & 4 & \\
2 & & 1 & 2 & \\
3 & & & 2 & \\
4 & & & &
\end{array}$$

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 1 & 3 & 4 \\
2 & & 1 & 2 & \\
3 & & & 1 & 2 \\
4 & & & & 2
\end{array}
\qquad
\begin{array}{c}\text{User Access}\\\text{to Level 4}\\\underline{i = 4}\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
1 & 1 & 1 & 3 & 4 \\
2 & & 1 & 2 & \\
3 & & & 1 & 2 \\
4 & & & & 2
\end{array}$$

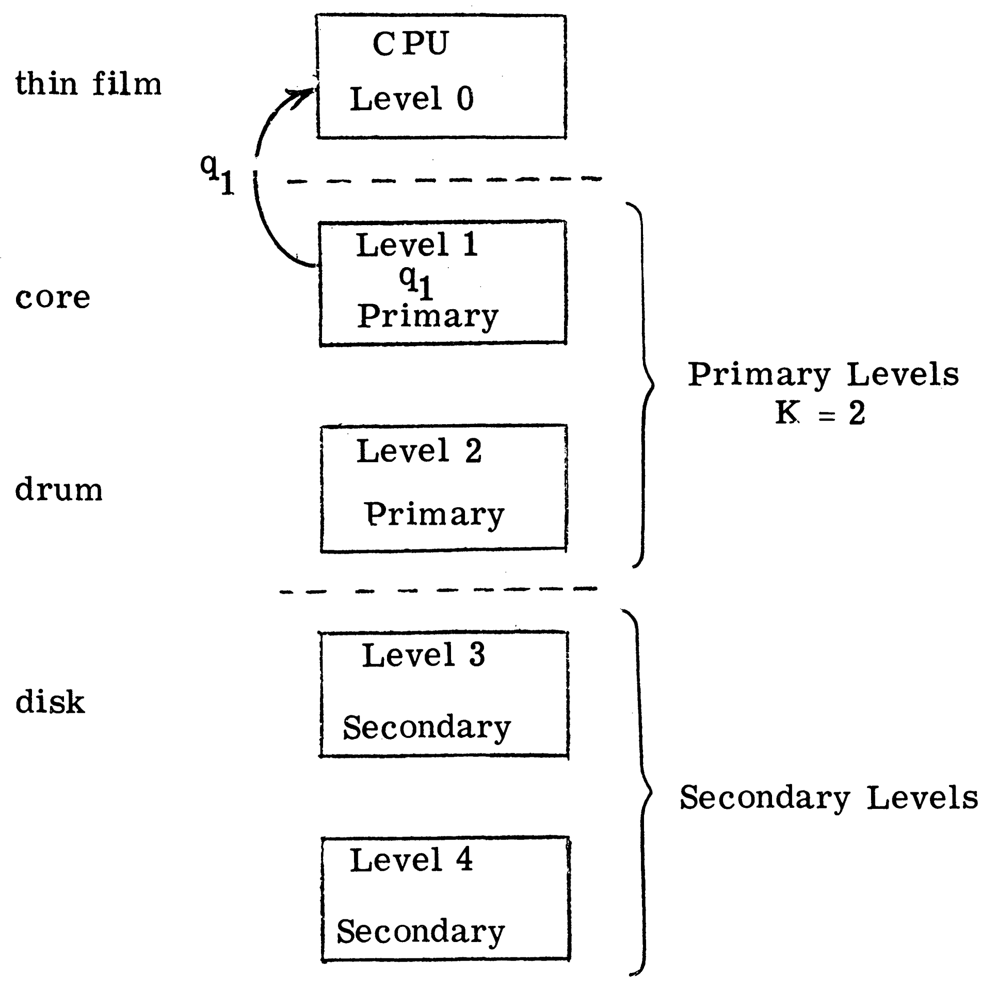
Figure 3.8 $\overline{T}$ Array for Example 2

where K is an integer satisfying $1 \leq K \leq N$ recalling that N is the total number of levels in the system.

We will now be treating a more general case where access to level K and above may cause delay in a user program's execution but will not cause the program to lose the use of the CPU. User accesses below the level K will cause program interruption and will release the use of the CPU to other programs in the system.

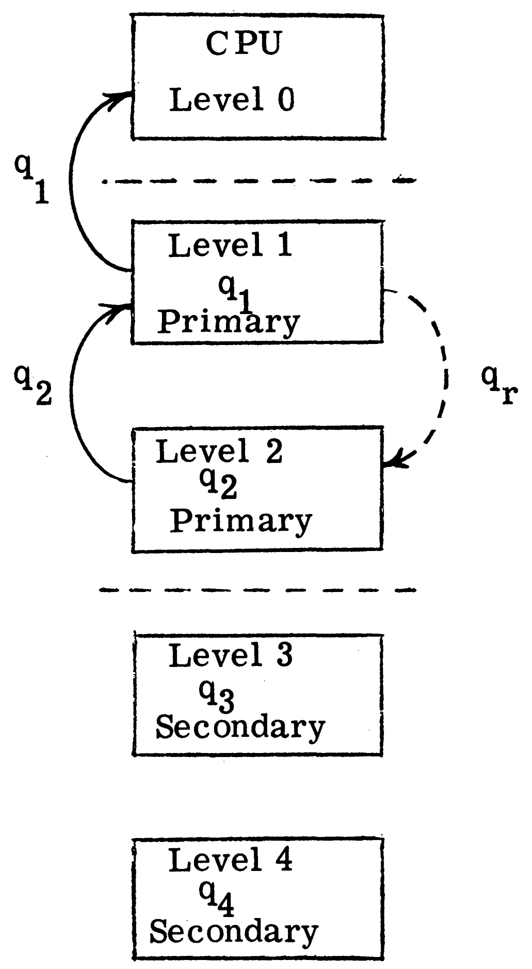
Let us consider an example where N = 4 and K = 2 as shown in Figures 3.9 through 3.13. Here we have designated the device type at each level as shown in Figure 3.9. We will transfer data in the primary levels only between adjacent levels as in an earlier example. User accesses to secondary levels will cause transfers as in the previous example but here the core at level 2 will be used as a buffer. Figures 3.9 and 3.10 show the transfers which result from user reads to the primary levels. User writes would result in a similar pattern differing only between the CPU and level 1 where we would write at level 1 rather than read.

Figures 3.11 and 3.12 show the transfer patterns for user accesses to secondary levels. Notice here that the upward movement of data stops in this example at level 2. This is because we have assumed in this example that level 1 is completely allocated to the program using the CPU. Thus in this example user programs accessing a secondary will lose all the allocations at level 1. This means that when programs are restarted they begin by accessing level 2 and have



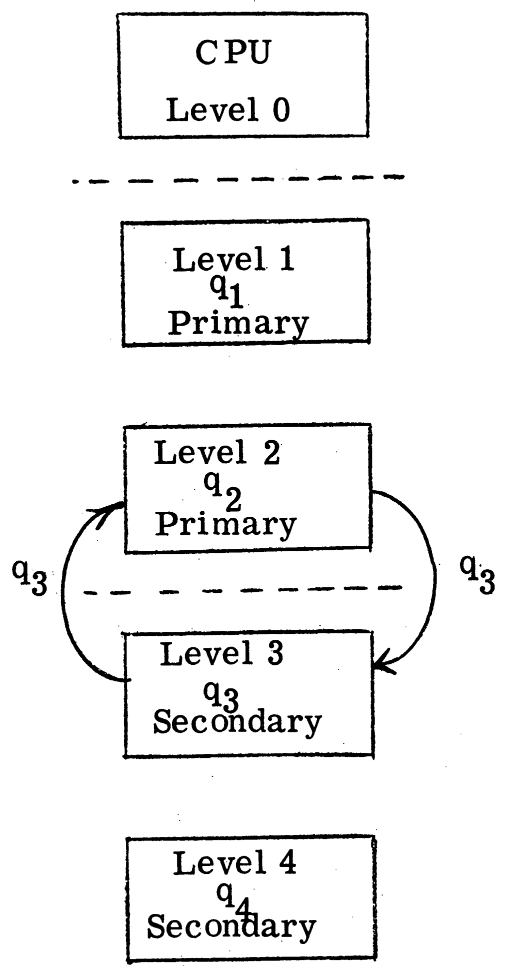
Example 3 Read to Level 1

Figure 3.9



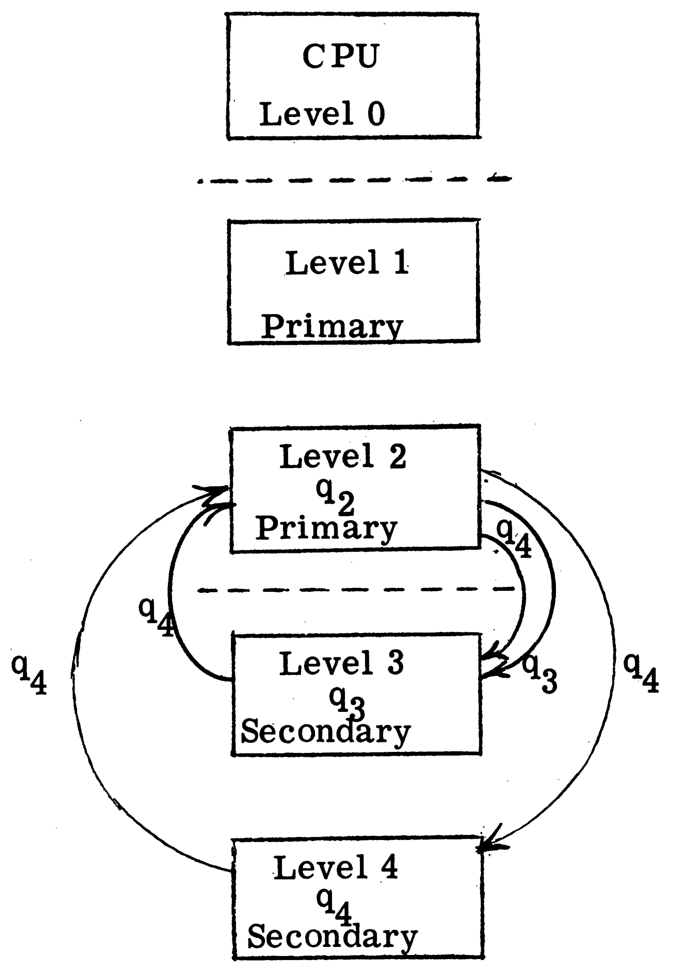
Example 3 Read to Level 2

Figure 3.10



Example 3 Read or Write to Level 3

Figure 3.11



Example 3 Read or Write to Level 4

Figure 3.12

READ ($\ell = 0$) WRITE ($\ell = 1$)

Records of Size (K)

Traffic at Level (j)

User Access to Level 1

$\underline{i = 1}$

$$\begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & 1 & & & \\ 2 & & & & \\ 3 & & & & \\ 4 & & & & \end{array} \qquad \begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & 1 & & & \\ 2 & & & & \\ 3 & & & & \\ 4 & & & & \end{array}$$

User Access to Level 2

$\underline{i = 2}$

$$\begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & 1 & 2 & & \\ 2 & & 2 & & \\ 3 & & & & \\ 4 & & & & \end{array} \qquad \begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & 1 & 2 & & \\ 2 & & 2 & & \\ 3 & & & & \\ 4 & & & & \end{array}$$

↑ Primary

- -

Secondary ↓

User Access to Level 3

$\underline{i = 3}$

$$\begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & & & & \\ 2 & & & 2 & \\ 3 & & & 2 & \\ 4 & & & & \end{array} \qquad \begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & & & & \\ 2 & & & 2 & \\ 3 & & & 2 & \\ 4 & & & & \end{array}$$

User Access to Level 4

$\underline{i = 4}$

$$\begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & & & & \\ 2 & & & 1 & 4 \\ 3 & & & 1 & 2 \\ 4 & & & & 2 \end{array} \qquad \begin{array}{c|cccc|} & q_1 & q_2 & q_3 & q_4 \\ 1 & & & & \\ 2 & & & 1 & 4 \\ 3 & & & 1 & 2 \\ 4 & & & & 2 \end{array}$$

Figure 3.13 $\overline{T}$ Array for Example 3

an initial transient period of paging in the primary levels.

The $\overline{t}_{i,\ell}$ matrices for all possible user accesses are shown in Figure 3.13.

3.7 Dedicated and Shared Levels of Storage

At this point we would like to formalize this concept of entire levels of storage being dedicated to the program using the CPU by defining

$$L = \text{Number of levels of CPU dedicated storage} \qquad (3.13)$$

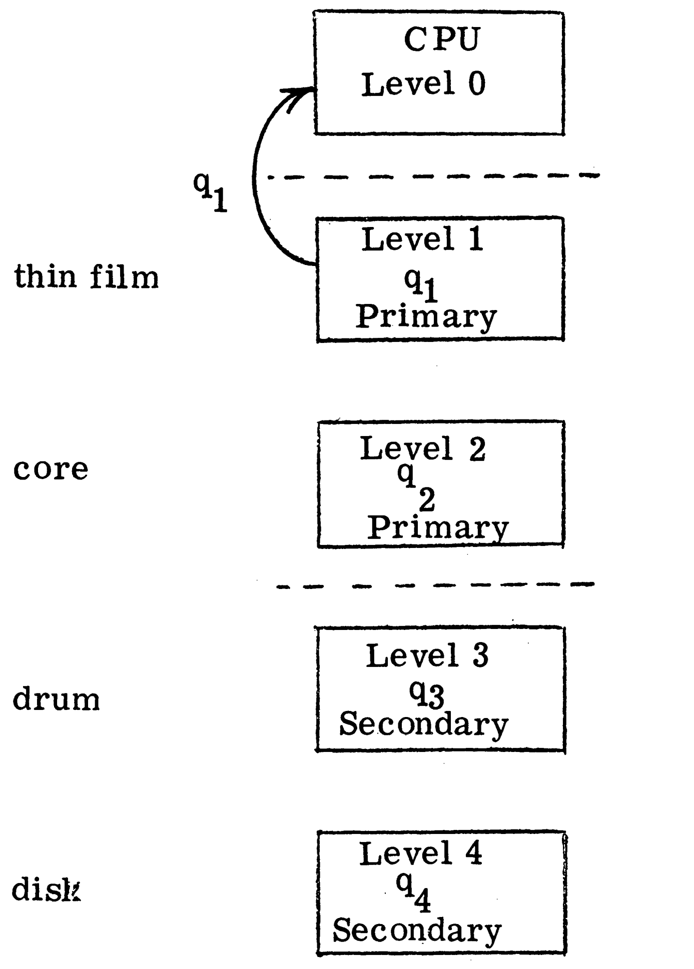
where L is an integer satisfying $0 \leq L < K$ recalling that K is the number of primary levels of storage.

The restriction that $L < K$ is required to avoid a situation where every user program being restarted would immediately make an access to secondary storage and thus be interrupted.

The Model 85

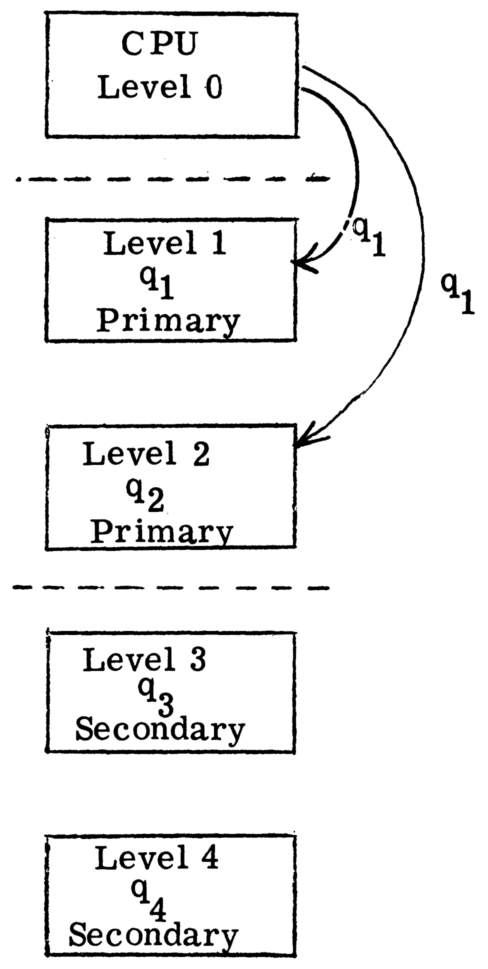
Motivation for considering systems with $K > 1$ is in part based on the apparent success of the IBM 360/85 which has two levels of primary storage (K=2). The level 1 storage of the Model 85 is thin film and is dedicated to the program using the CPU (L=1). Because of the interest generated by this particular design as well as its relevance to this work we will consider a Model 85 like situation where we have added 2 additional levels of storage, a drum and a disk, and assumed a virtual addressing feature as on the IBM 360/67.

Figure 3.14 shows the transfer path for a read to level 1. Constrasting this to the write to level 1 shown in Figure 3.15 we can



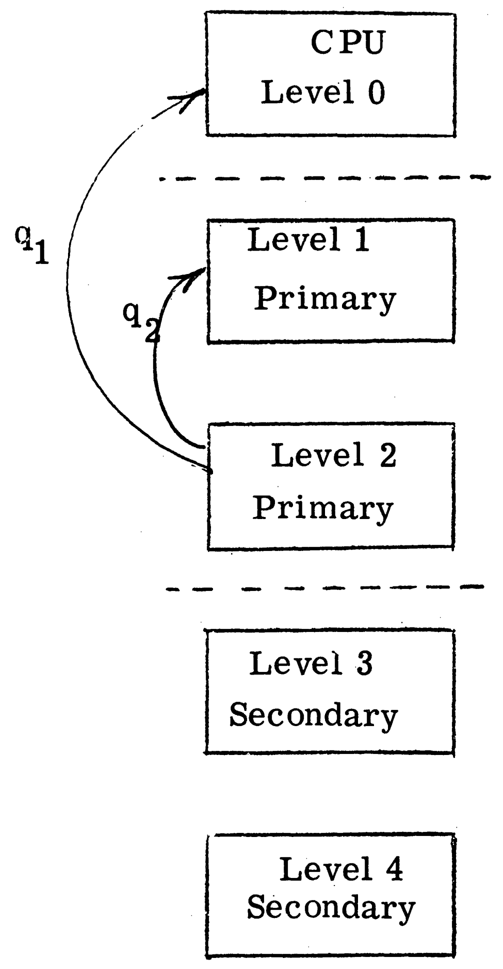
Example 4 Read to Level 1

Figure 3.14



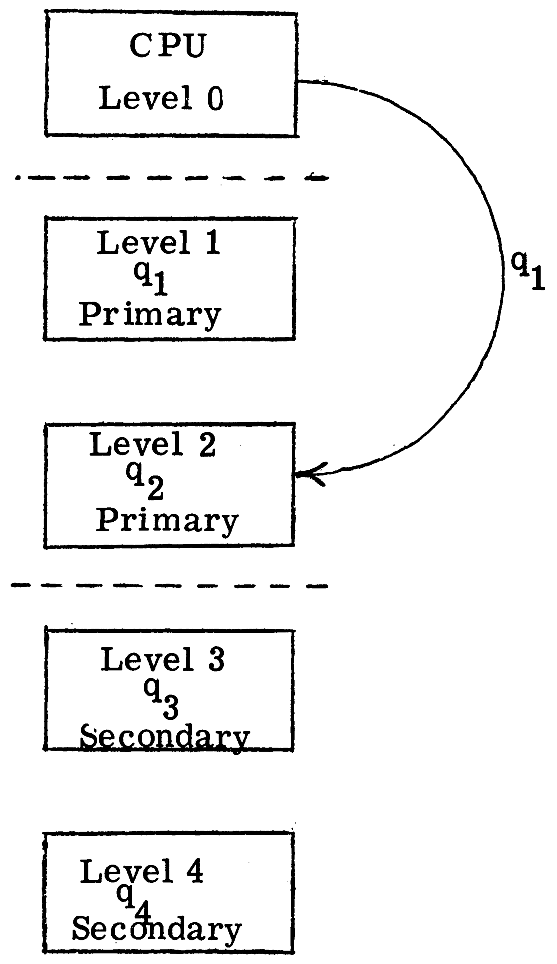
Example 4 Write to Level 1

Figure 3.15



Example 4 Read to Level 2

Figure 3.16



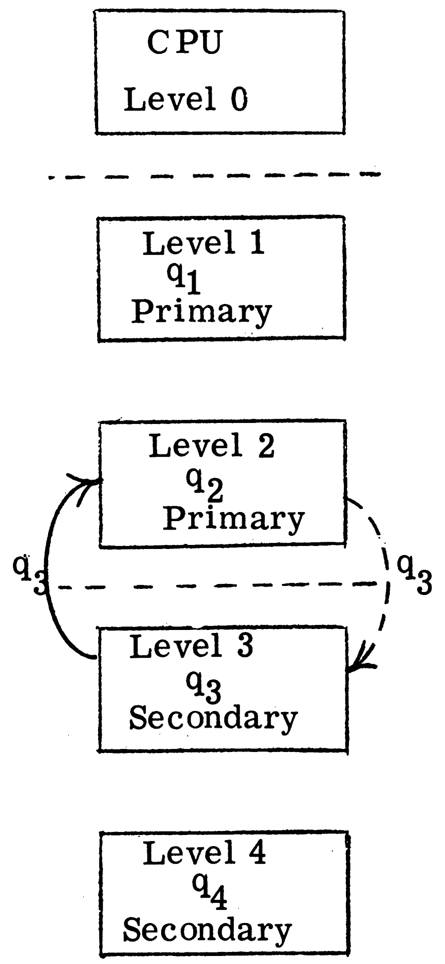
Example 4 Write to Level 2

Figure 3.17

see that there is a considerable difference between reads and writes to level 1. This difference is a result of the fact that a current copy of the contents of level 1 is always maintained at level 2. This means that there will never be an occasion necessitating the actual paging down of records from level 1 to level 2 in order to make room for records being paged up, When space at level 1 is required it is simply released to the new record since a current copy is always maintained at level 2. Figure 3.16 and 3.17 show the transfer patterns of a read and a write to level 2. The read and write differ here also in that a read to level 2 causes information to be moved up to level 1 (64 bytes in the model 85) and a write does not.

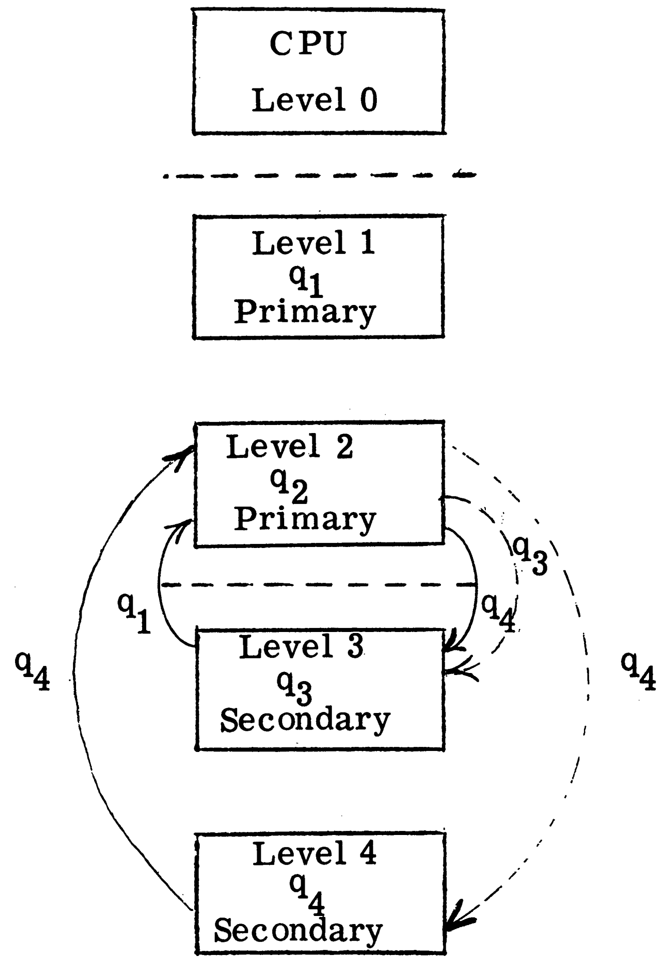
The transfer patterns resulting from user accesses to the secondary levels are shown in Figure 3.18 and 3.19. Here we see patterns similar to these discussed before and more typical of a IBM 360/67.

The traffic matrices $\bar{t}_{i,\ell}$ for all user accesses are shown in Figure 3.20. Notice here that the user read ($\ell=0$) and use write ($\ell=1$) matrices differ.



Example 4 Read or Write to Level 3

Figure 3.18



Example 4 Read or Write to Level 4

Figure 3.19

READ ($\ell = 0$) WRITE ($\ell = 1$)

Records of Size (K)

User Access to Level 1

Traffic at Level (j)

$\uparrow$ Primary

<u>i = 1</u>

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & 1 & & & \\
2 & & & & \\
3 & & & & \\
4 & & & & \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & 1 & & & \\
2 & 1 & & & \\
3 & & & & \\
4 & & & & \\
\hline
\end{array}$$

<u>i = 2</u>

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & 1 & & \\
2 & & 1 & & \\
3 & & & & \\
4 & & & & \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & & & \\
2 & 1 & & & \\
3 & & & & \\
4 & & & & \\
\hline
\end{array}$$

Secondary $\downarrow$

<u>i = 3</u>

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & & & \\
2 & & & 2 & \\
3 & & & 2 & \\
4 & & & & \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & & & \\
2 & & & 2 & \\
3 & & & 2 & \\
4 & & & & \\
\hline
\end{array}$$

<u>i = 4</u>

$$\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & & & \\
2 & & & 1 & 4 \\
3 & & & 1 & 2 \\
4 & & & & 2 \\
\hline
\end{array}
\qquad
\begin{array}{c|cccc|}
 & q_1 & q_2 & q_3 & q_4 \\
\hline
1 & & & & \\
2 & & & 1 & 4 \\
3 & & & 1 & 2 \\
4 & & & & 2 \\
\hline
\end{array}$$

Figure 3.20 $\overline{T}$ Array for Example 4

Regions of Storage

Figure 3.21 shows a general storage hierarchy relating the variables L, K, N and the terms dedicated, shared, primary, and secondary. Recalling that

Dedicated levels are completely allocated to the user program currently executing.

Primary levels are those to which an access may be made without causing an interrupt terminating the current users use of the CPU.

Secondary levels are those which if accessed cause a user program to lose the use of the CPU.

There are three basic regions of interest. They are:

1. Dedicated (levels 0 through L)
2. Shared Primary (levels L + 1 through K)
3. Secondary (levels K + 1 through N)

These three regions are of special interest because the data traffic in each is based on different factors. In the dedicated region all traffic is directly the result of the currently executing program. In the secondary levels the traffic is the result of the entire collection of programs being multiprogrammed. In the shared primary region the traffic is affected by both factors and thus is a function of the collection of programs being multiprogrammed as well as the specific program in execution.

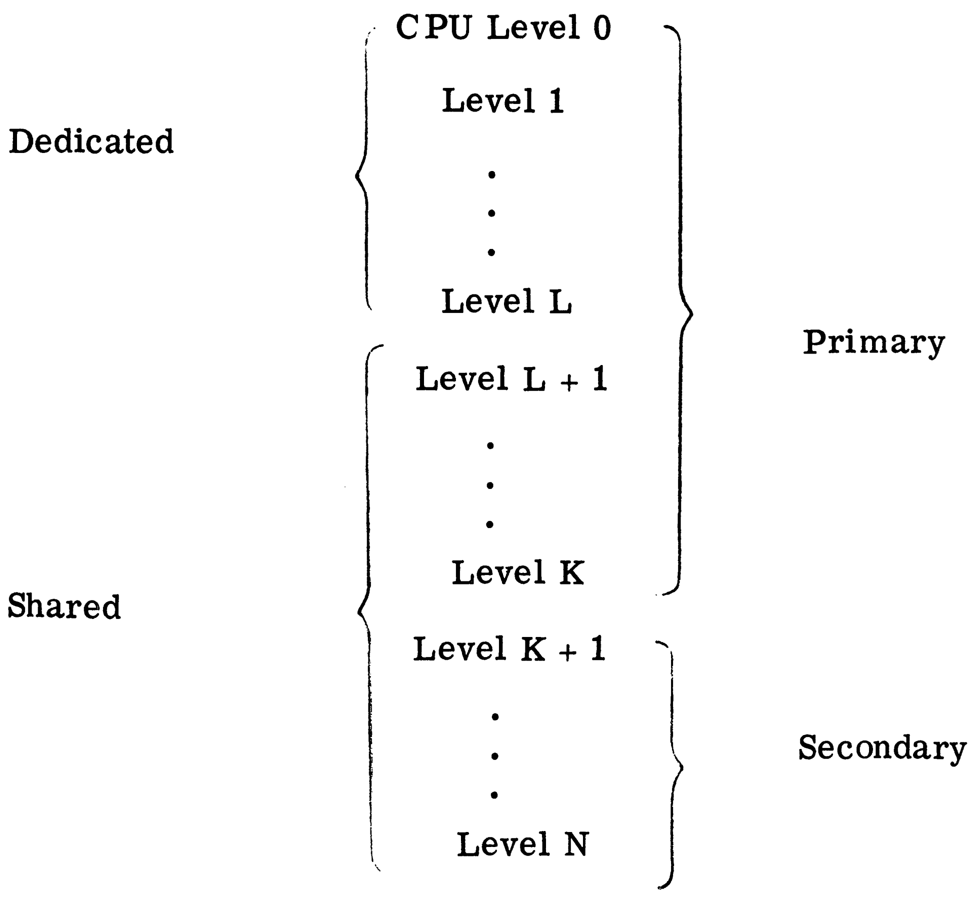


Figure 3.21 Regions of Storage

Allocation

In the dedicated levels of storage the entire capacity of each level is allocated to the program executing. In the shared levels we will allocate a fixed amount of storage at each level to each user. This fixed amount at a given level being some fraction of the total available at that level. The sum of the individual user allocations at any level will equal the total at that level.

Having a fixed allocation the question arises as to what should that allocation be. We will choose a scheme of storage allocation such that the expected execution rate (number of user storage accesses per real time sec) is equal for all of the M jobs being multiprogrammed. Thus we will have the option of running or not running a job (value of M) but if we do run it we will provide it with sufficient storage to so that it may have the same expected rate of execution as the other jobs running at the same time.

There are an infinite number of allocation strategies which will provide the desired result of equal expected execution rate. We have made a choice among these which is simple and provides certain important mathematical simplifications. <u>We will allocate the storage at each shared level such that the probability of making an access below that level is equal for all M programs being multiprogrammed.</u>

It is clear that this allocation will make all program behave identically in the system when $L = 0$. When $L > 0$ the dedicated levels of storage which always provide each program with the same allocation will cause each program to behave differently according to its lifecycle function. To avoid the complications involved in considering each program separately we will consider only cases where $L = 0$ or all programs have the same lifecycle function.

3.8 Traffic Description at Individual Levels

In writing the equations which describe the performance of the system we will view the system from two points of view. In Figure 3.22 we show the periods of user program execution interspersed with periods when the CPU is either idle or performing various system housekeeping functions. Notice that in our modeling of the system we have attributed the paging traffic in the system to the user program's behavior. Thus we will assume that the traffic introduced by the system which has not already been accounted for and associated with individual user programs is negligible. We will consider a more detailed model later but for the present we can simply think of the CPU as either executing a user program or idle as shown in Figure 3.22.

We will define

r = the average rate at which (the collection of M) user programs make access to storage (3.14)

and

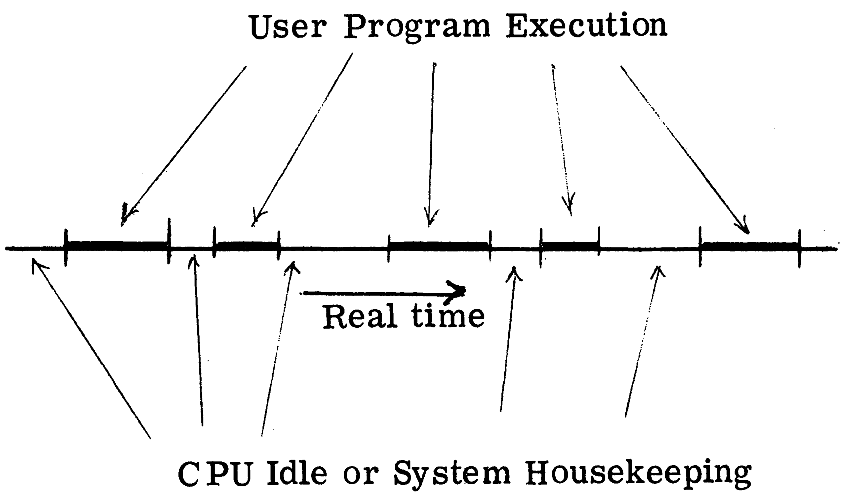


Figure 3.22 Periods of User Program Execution

r' = the average rate at which an executing user program makes access to storage when a user program is in fact executing.

(3.15)

The variables r and r' are simply related by

$$r = e'r'$$

where e' is the fraction of all time that any user program is in execution.

We will now want to express the average time required to complete a secondary access. The immediate goal is to write expressions for the traffic or rate at which records are read and written at each level. Clearly the traffic in the system is effected by executing user programs. That is the traffic in the system when a user program is executing is not the same as the traffic when the CPU is idle.

We will first consider the traffic as seen from the point of view of secondary paging. The actual reads and writes which occur as a result of a secondary access do not occur specifically during a period of user program execution or during a CPU idle period. The transfers which occur can be thought of as occurring over a period of several to many periods of both user program execution and idle CPU time. Thus we will treat the secondary paging transfers as though they occur in a traffic environment described by the average access rate taken over all time, r.

We will define

$d_{j,k}$ = the average over all time of the number of records of size q_k read or written at level j per unit real time (3.17)

where j and k are integers satisfying

$$1 \leq j \leq N$$

$$1 \leq k \leq N$$

We may write an expression for $d_{j,k}$ as follows

$$d_{j,k} = \sum_{i=1}^{N} [r(1-w) u_i t_{j,k,i,o} + rw u_i t_{j,k,i,1}] \qquad (3.18)$$

In the way of explanation we will disect the terms of the summation beginning with

$$r(1-w) u_i t_{j,k,i,o}$$

r = rate of user program accesses per unit real time.

$r(1-w)$ = rate of user program reads per unit real time.

$r(1-w) u_i$ = rate of user program reads to level i per unit real time.

$r(1-w) u_j t_{j,k,i,o}$ = rate of reads and write at level j involving records of size q_k as a result of user program reads to level i per unit real time.

The second term in the summation

$$rw u_i t_{j,k,i,1}$$

is the same as the first except here the user program writes are accounted for. Summing these two terms over all possible levels of user program access gives us $d_{j,k}$.

We will define

C_j = the average over all time of the number of records of any size read and written at level j per unit real time. (3.19)

This may be expressed simply as a sum of $d_{j,k}$ over k

$$C_j = \sum_{k=1}^{N} d_{j,k} \tag{3.20}$$

We will also define

A_j = the average over all time of the size of records being read and written at level j. (3.21)

which may be expressed simply as

$$A_j = \frac{1}{C_j} \sum_{k=1}^{N} q_k\, d_{j,k}. \tag{3.22}$$

Defining

$$Z_j = \sum_{k=1}^{N} \sum_{i=1}^{N} [(1-w)\, u_i\, t_{j,k,i,o} + w\, u_i\, t_{j,k,i,1}] \tag{3.23}$$

and

$$Y_j = \sum_{k=1}^{N} \sum_{i=1}^{N} q_k[(1-w)\, u_i\, t_{j,k,i,o} + w\, u_i\, t_{j,k,i,1}] \tag{3.24}$$

We may write

$$C_j = r\, Z_j \tag{3.25}$$

and

$$A_j = \frac{Y_j}{Z_j} \qquad (3.26)$$

where the terms Z_j and Y_j are expressed completely as functions of:

1. Allocation matrix $\overline{S}$
2. Record sizes q_k's
3. Lifecycle function $f(s_i, q_{i+1})$
4. Fraction of read accesses w
5. Traffic matrices $\overline{t}_{i,\ell}$
6. Number of programs M
7. Number of levels N
8. Number of primary levels K

For our purposes the most important consideration is that Z_j and Y_j are not functions of r or r'.

We would now like to consider the traffic environment in which user program accesses to primary levels are made. Clearly user program accesses to primary levels always occur when a user program is executing. We may also limit our concern to the traffic in the primary levels. This is simply because user program accesses to primary levels never involve data transfers in the secondary levels.

We find two sources of traffic in the primary levels. The first component comes directly from the executing user programs accesses to primary levels which occur at a rate r'. The second component is the result of secondary paging transfers which extend into the primary levels. This component is based on the access rate taken over all time r.

Consider a simple 2-level system with one primary level, a core, and one secondary level, a drum. We would find two components of traffic at the core when a user program is executing. One generated by the executing program itself and a second generated by the paging activity at the drum.

We will define

$d'_{j,k}$ = the average over user execution time of the number of records of size q_k read or written at level j per unit execution time. (3.27)

Where j and k are integers satisfying

$$1 \leq j \leq K$$

$$1 \leq k \leq N$$

We may write an expression for $d'_{j,k}$ as follows

$$d'_{j,k} = \sum_{i=1}^{K} [r'(1-w)\, u_i\, t_{j,k,i,o} + r'w\, u_i\, t_{j,k,i,1}]$$
$$+ \sum_{i=K+1}^{N} [r(1-w)\, u_i\, t_{j,k,i,o} + rw\, u_i\, t_{j,k,i,1}] \qquad (3.28)$$

The expression for $d'_{j,k}$ is very similar to the expression for $d_{j,k}$ with the exception that the summation is broken into two parts. The first from 1 to k, the primary levels, and the second k+1 to N, the secondary levels. The first summation gives us the traffic component generated directly by the executing user program. The second summation gives us the traffic component introduced by secondary paging activities.

We will define

C'_j = the average over user program execution time of the number of records of any size read and written at level j per unit execution time. (3.29)

$$= \sum_{k=1}^{N} d'_{j,k} \tag{3.30}$$

We will also define

A'_j = the average over user program execution time of the size of records being read and written at level j. (3.31)

which may be expressed

$$A'_j = \frac{1}{C'_j} \sum_{k=1}^{N} q_k \, d'_{j,k} \tag{3.32}$$

Defining

$$Z'_j = \sum_{k=1}^{N} \sum_{i=1}^{K} [(1-w) u_i \, t_{j,k,i,o} + w \, u_i \, t_{j,k,i,1}] \tag{3.33}$$

$$Z''_j = \sum_{k=1}^{N} \sum_{i=K+1}^{N} [(1-w) u_i \, t_{j,k,i,o} + w \, u_i \, t_{j,k,i,1}] \tag{3.34}$$

Also defining

$$Y'_j = \sum_{k=1}^{N} \sum_{i=1}^{K} q_k[(1-w)\,u_i\, t_{j,k,i,o} + w\, u_i\, t_{j,k,i,1}] \quad (3.35)$$

and

$$Y''_j = \sum_{k=1}^{N} \sum_{i=k+1}^{N} q_k[(1-w)\,u_i\, t_{j,k,i,o} + w\, u_i\, t_{j,k,i,1}] \quad (3.36)$$

We may write

$$C'_j = r'\, Z'_j + r\, Z''_j \quad (3.37)$$

and

$$A'_j = \frac{r'\, Y'_j + r\, Y''_j}{r'\, Z'_j + r\, Z''_j} \quad (3.38)$$

Notice that here as in the case of Z_j and Y_j the terms Z'_j, Z''_j, Y'_j and Y''_j are independent of r and r'. We may also write:

$$Z_j = Z'_j + Z''_j \quad (3.39)$$

and

$$Y_j = Y'_j + Y''_j \quad (3.40)$$

We see that if a user program is always executing making r' = r then

$$C'_j = C_j \quad (3.41)$$

and

$$A'_j = A_j \quad (3.42)$$

3.9 Storage Hardware Performance

The purpose of the β function is to provide a representation for the hardware at a single level which characterizes its performance in the overall system environment. We will define the β function as follows:

$\beta_j(C, A, q)$ = the average total time required to read or write a record at level j as a function of C, the average number of records read and written per unit time, A, the average record length, and q, the size of the specific record being read.

In general the arguments of this function will take the form $\beta_j(C_j, A_j, q_k)$ or $\beta_j(C'_j, A'_j, q_k)$ according to whether we are considering secondary or primary level paging.

The function $\beta_j(C, A, q)$ will include such delays as waits for disk seek arms to become free, disk arm seek time, wait for channel to become free, latency and read or write time. Thus $\beta_j(C, A, q)$ is itself the result of a rather detailed model of the hardware at the level j. We will devote the entire next chapter to a detailed development of these models.

It is interesting to point out that Belady [6] in modeling the 2-level hierarchy considered what we are modeling here with a function to be a constant. It was simply assumed that no queueing occurred.

Before continuing we will stop to consider some limitations on $\beta_j(C_j, A_j, q_k)$ and $\beta_j(C'_j, A'_j, q_k)$ as functions of r and r'. In the case of $\beta_j(C_j, A_j, q_k)$ we may expand C_j and A_j to write

$$\beta_j(C_j, A_j, q_k) = \beta_j(r\, Z_j, \frac{Y_i}{Z_j}, q_k) \tag{3.44}$$

Here we will only be concerned with the variable r. Clearly as r increases the average rate at which level j must process reads and writes increases. One would expect that the average time required to perform a single read or write at level j to increase with the increased traffic. We will limit the choice of functions $\beta_j(C, A, q)$ to those which are monotone non-decreasing with increasing C. Thus we can be sure that $\beta_j(C_j, A_j, q_k)$ is monotone non-decreasing with increasing r.

The case for $\beta_j(C'_j, A'_j, q_k)$ is a bit more complex. Expanding C'_j and A'_j we may write:

$$\beta_j(C'_j, A'_j, q_k) = \beta_j(r'\, Z'_j + r\, Z''_j, \frac{r'\, Y'_j + r\, Y''_j}{r'\, Z'_j + r\, Z''_j}, q_k) \tag{3.45}$$

Here we will be concerned with both r and r'. Clearly the average rate at which level j must process reads and writes increases as either r or r' increases. However here we find that the average record length A'_j is rather unpredictable as a function of r or r'.

In order to determine what we should expect of $\beta_j(C'_j, A'_j, q_k)$ we must recall the development of C'_j and A'_j. C'_j and A'_j are the result of two components of traffic one component is proportional to r and a second to r'. Thus increases in r will produce an increase in one component of the traffic and no change in the other and the same can be said for r'. Changes in A'_j which occur as a result of increases in either r or r' are clearly the result of an increase in one of the two components of traffic at level j. Thus we would expect that $\beta_j(C'_j, A'_j, q_k)$ would increase with increases in either r or r'. We will limit our choice of functions $\beta_j(C'_j, A'_j, q_k)$ to those which are non-decreasing with increasing r or r'.

3.10 Primary and Secondary Access Time

Up to this point we have written expressions which describe the traffic in the system and defined a function $\beta_j(C, A, q)$ which gives us the expected read or write time for a single record at level j under given traffic conditions. We will continue here by developing expressions for the average time required to complete a paging operation. This will of course in general involve several levels and several reads and writes.

We will be discussing two distinct access completion times. The first will be the time between a user program's access to some secondary level, thereby losing the use of the CPU, and the completion of the transfers required to make that user program eligible for execution again. Secondly we will investigate the time required for the executing program to complete accesses in the primary levels and it's effect on the execution rate of the program in question.

When describing the $\overline{T}$ array and its elements $t_{j,k,i,\ell}$ we were considering the traffic induced in the system as a result of a single user access. It is fairly clear that all the reads or writes indicated in the $\overline{T}$ array for a single user access need not be accomplished to complete the action required to make a program again eligible for execution or to continue execution if the user access is to a primary level.

Let us examine a specific case. Turning to Figure 3.6 of this chapter we see a most complex series of transfers involved in the paging operations of a user access to level 4 of a 4-level hierarchy. In this particular case $\ell = 0$ and $k = 1$. The only data transfer required to allow execution to continue is the transfer from level 4 to level 1 of a record of size q_4. The other indicated transfers may be carried out later or in some cases prior to the actual access to level 4.

Now in regard to the specific transfer from level 4 to level 1 the question arises as to the length of time required to complete the transfer. We have both a read at level 4 of a record of size q_4 and a write at level 1 of a record of size q_4. We will assume for the moment that the level 4 device is a disk, the level 1 device is a core, and the channel connecting the levels contains a buffer of size q_1. In this case we find that the read at level 4 and the write at level 1 must be carried out simultaneously except for the last word (record of size q) read into core.

In common equipment configurations the time required to complete the transfer will depend entirely on the time required to complete the level 4 read. Strictly speaking we could add the time required to write that last word from the disk in core (a level 1 write of size q_1). However this would be a rather wasted effort in this case.

In this case and similar cases we will treat the time required to complete the transfer simply as the time required to perform the read (or write) at the slower device. Thus in this particular case the time required for the read at level 4 is the time required to return the user program to a condition in which it can be eligible for execution. We will refer to the read at level 4 as a critical read.

In another case, shown in Figure 3.4 of this chapter, (which is based on a different system philosophy) an access to level 4 will require a total of three distinct record transfers to complete the necessary paging action. They are:

1. From level 4 to level 3 a record of size q_4
2. From level 3 to level 2 a record of size q_3
3. From level 2 to level 1 a record of size q_2

When these three transfers are complete the program will be eligible for execution.

Here we may encounter a situation where the channel buffer is large. For instance if we were to consider a direct transfer between a disk and a drum the size of the channel buffer would almost necessarily be as large as the record being transferred. The time required to complete the transfer would then include both the read (or write) at the disk and the write (or read) at the drum.

In general we find that there will be a series of sequential reads and/or writes which must be completed in order to continue or be eligible to continue execution. We will refer to these reads and writes as critical path reads and writes borrowing from the terminology of Critical Path Planning (CPM), PERT and other PERT like planning methods.

It should be pointed out that we are making the assumption that there will almost always be space available at each level into which records may be written.

Formalizing these ideas we will define $g_{j,k,i,\ell}$ very similarly to $t_{j,k,i,\ell}$ except here we will only include the critical reads and writes. We define:

$g_{j,k,i,\ell}$ = the number of critical reads and writes at level j involving records of size q_k in a user program read ($\ell = 0$) or write ($\ell = 1$) to level i. (3.46)

READ ($\ell = 0$) WRITE ($\ell = 1$)

Records of Size (k)

Traffic at Level (j)

User Access to Level 1

<u>i = 1</u>

$$\begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & 1 & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array} \qquad \begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & 1 & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array}$$

User Access to Level 2

<u>i = 2</u>

$$\begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & \cdot & 1 & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array} \qquad \begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & 1 & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array}$$

User Access to Level 3

<u>i = 3</u>

$$\begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & 1 & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array} \qquad \begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & 1 & \cdot \\ 4 & \cdot & \cdot & \cdot & \cdot \end{array}$$

User Access to Level 4

<u>i = 4</u>

$$\begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & 1 \end{array} \qquad \begin{array}{c|cccc} & q_1 & q_2 & q_3 & q_4 \\ \hline 1 & \cdot & \cdot & \cdot & \cdot \\ 2 & \cdot & \cdot & \cdot & \cdot \\ 3 & \cdot & \cdot & \cdot & \cdot \\ 4 & \cdot & \cdot & \cdot & 1 \end{array}$$

$\overline{G}$ array

Figure 3.23

We will refer to the collection of these elements, $g_{j,k,i,\ell}$, as the $\overline{G}$ array. We will also use $\overline{g}_{i,\ell}$ to indicate the matrix of $g_{i,k,i,\ell}$ terms where i and ℓ are fixed.

Earlier in this chapter we considered an example much like the IBM 360/85. The $\overline{T}$ array for this example is shown in Figure 3.20 and several preceding figures show transfer diagrams. The $\overline{G}$ array can be obtained from inspection of the transfer diagrams and is shown in Figure 3.23. In this case each $\overline{g}_{i,\ell}$ matrix has only one entry; note that this is not always the case.

We are now in a position to write an expression for the average time required to complete a user program access. We will begin with user program accesses to secondary levels. In the case of a user program read we may write:

$$\text{Average Completion Time} = \sum_{j=1}^{N} \sum_{k=1}^{N} g_{j,k,i,o}\, \beta_j(C_j, A_j, q_k) \qquad (3.47)$$

This is a simple summation of the critical path read and write times involved in a user program read to level i. The write completion time can be expressed as

$$\text{Average Completion Time} = \sum_{j=1}^{N} \sum_{k=1}^{N} g_{j,k,i,1}\, \beta_j(C_j, A_j, q_k) \qquad (3.48)$$

the only difference being in the last subscript of $g_{j,k,i,\ell}$. We can now express the average time required to complete an unspecified access to level i. We will define

T_i = the average time required to complete a user program access to level i

where i is an integer satisfying

$$1 \leq i \leq N \tag{3.49}$$

Expressing T_i as

$$T_i = \sum_{j=1}^{N} \sum_{k=1}^{N} [(1-w) g_{j,k,i,o}\, \beta_j(C_j, A_j, q_k) + w\, g_{j,k,i,1}\, \beta_j(C_j, A_j, q_k)]$$

$$= \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C_j, A_j, q_k) [(1-w) g_{j,k,i,o} + w\, g_{j,k,i,1}] \tag{3.50}$$

where (1-w) is the probability of a read access and w is the probability of a write access.

and i is an integer satisfying

$$K < i \leq N$$

(i.e. i indicates a secondary level)

The time required to complete a user program access to a primary level may be expressed as:

$$T_i = \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C'_j, A'_j, q_k) [(1-w) g_{j,k,i,o} + w\, g_{j,k,i,1}] \tag{3.51}$$

where i is an integer satisfying

$$1 \leq i \leq K$$

The only difference being the traffic at level j. C'_j replaces C_j and A'_j replaces A_j.

We now have expressions for the average time required to complete an access to any one of the N levels of storage. The next step will be to express the average time required to complete a secondary level access when the specific level is not specified. This is in effect the average time required to complete a secondary page fault.

We have already expressed the fraction of accesses to each level i as u_i. We will define

u''_i = the probability of accessing level i given that the access is to a secondary level (levels K+1 through N).

This may be expressed simply as

$$\begin{aligned} u''_i &= 0 \qquad \text{for } 0 \le i \le K \\ &= \frac{u_i}{\sum_{i=K+1}^{N} u_i} \qquad \text{for } K+1 \le i \le N \end{aligned} \tag{3.52}$$

where

$$u_i = \frac{1}{f(s_{i-1}, q_i)} - \frac{1}{f(s_i, q_{i+1})} \quad \text{for } K+1 \le i \le N \tag{3.53}$$

and

$$\frac{1}{f(s_N, q_{N+1})} \equiv 0$$

We will consider an expression for u_i where $1 \leq i \leq K$ later.

We can see directly from the definition of the lifecycle function or from the expression for u_i in Equation 3.53 that

$$\sum_{i=K+1}^{N} u_i = \frac{1}{f(s_K, q_{K+1})} \tag{3.54}$$

Thus we may write

$$\begin{aligned} u''_i &= 0 && \text{for } 0 \leq i \leq K \\ &= f(s_K, q_{K+1})\, u_i && \text{for } K+1 \leq i \leq N \end{aligned} \tag{3.55}$$

We will define

$\overline{\tau}_s$ = the average time required to complete a secondary page fault. (3.56)

Using u''_i this may be expressed as

$$\overline{\tau}_s = \sum_{i=K+1}^{N} u''_i\, T_i \tag{3.57}$$

or

$$= f(s_K, q_{K+1}) \sum_{i=K+1}^{N} u_i\, T_i \tag{3.58}$$

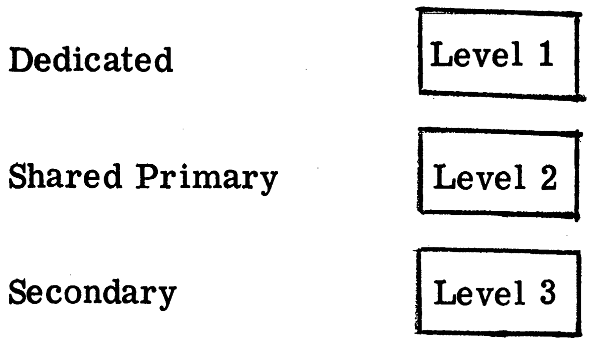
Having an expression for the average time required to make an access to a secondary level we will now turn to the average time required to make a primary access. We will define

u'_i = the probability of accessing level i given that the access is to a primary level (levels 1 through K). (3.59)

The expression for u'_i is quite involved and involves considerable digression.

When a user program begins execution it experiences an initial period in which the levels of dedicated storage go from a state of containing only the previous CPU user's (system or user programs) information to containing only the current CPU users program. This initial period should be short and in most systems of interest will have negligible or near negligible effect. However we can expect to encounter systems in which the periods of user program execution are so short that the characteristics of this initial period dominate the entire execution period of the user program. The ultimate effect is to modify the probability of accessing a given level, u_i.

We will now consider the accessing process in some detail focusing on the initial period in particular. Consider a hierarchy in which $L = 1$, $K = 2$ and $N = 3$. This will be a 3-level hierarchy with one level of dedicated storage, one level of shared primary and one level of secondary.



3-Level Hierarchy

Figure 3.24

Let us examine what happens when a user program begins execution and has no information at the dedicated level, level 1. The first access must go below level 1 because there is no information associated with the executing program at level 1. Assuming that the access is to a primary level, i.e. the user program's period of execution is not terminated, we will find q_2 bits of the executing program's information at level 1 following the first access. The user program may then continue for an average of $f(q_2, q_2)$ accesses before making another access below level 1. The second access below level 1 will bring the useful information at level 1 to $2q_2$ bits and we may expect execution to proceed for an average of $f(2q_2, q_2)$ accesses before making a third access below level 1. The quantity of useful information at level 1 will grow with each access below level 1 by q_2 until level 1 contains only information associated with the currently executing program at which point the quantity of information at level 1 will remain at s_1, the allocation at level 1 which in the case of a dedicated level corresponds to the capacity of that level. Correspondingly the average number of accesses which can be carried out with a given number of accesses below level 1 may be written as a finite series of the form:

$$f(q_2, q_2) + f(2q_2, q_2) + f(3q_2, q_2) + \dots + f(s_1, q_2) + \dots + f(s_1, q_2) \qquad (3.60)$$

where the number of accesses below level 1 corresponds to the number of terms in the series.

If we define a function

$$\begin{aligned} s_i(x) &= x \qquad \text{for } x \leq s_i \\ &= s_i \qquad \text{for } x \geq s_i \end{aligned} \qquad (3.61)$$

where s_i is the allocation at level i we may write the above series as

$$\sum_{j=1}^{R} f(s_1(j\, q_2), q_2) \qquad (3.62)$$

where R is the number of accesses below level 1.

In order to obtain an effective lifecycle function which includes the initial period of execution we will define:

$f'(s_1, q_2)$ = the average number of accesses required to produce an access below level 1 where it is assumed that initially no information associated with the executing program is stored at level 1. (3.63)

This can be written as

$$f'(s_1, q_2) = \frac{\text{\# of accesses}}{\text{\# of accesses below level 1}} \qquad (3.64)$$

We will obtain an approximation for $f'(s_1, q_2)$ by taking an average over the first R terms of the series of Equation 3.62 where R is chosen so that the summation of the series (the average number of accesses which can be carried out) corresponds to $f(s_K, q_{K+1})$ (the average number of accesses made during a single user program execution period).

Thus we will write

$$f'(s_1, q_2) \approx \frac{\sum_{j=1}^{R} f(s_1(j\, q_2), q_2)}{R} \qquad (3.65)$$

where R is the smallest integer satisfying

$$f(s_K, q_{K+1}) \leq \sum_{j=1}^{R} f(s_1(j\, q_2), q_2) \qquad (3.66)$$

Notice that as $f(s_K, q_{K+1})$ becomes large $f'(s_1, q_2) \longrightarrow f(s_1, q_2)$.

The approach we have taken here to level 1 is completely extendable to all the dedicated levels of storage. In general we may define

$f'(s_i, q_{i+1})$ = the average number of accesses required to produce an access below level i where it is assumed that initially no information associated with the executing program is stored at level i. (3.67)

And we may write

$$f'(s_i, q_{i+1}) \approx \frac{\sum_{j=1}^{R_i} f(s_i(j\, q_{i+1}), q_{i+1})}{R_i} \tag{3.68}$$

where R_i is the smallest integer satisfying

$$f(s_K, q_{K+1}) \leq \sum_{j=1}^{R_i} f(s_i(j\, q_{i+1}), q_{i+1}) \tag{3.69}$$

We may now express the overall fraction of accesses to each of the first k levels as

$$u_i = \frac{1}{f'(s_{i-1}, q_i)} - \frac{1}{f'(s_i, q_{i+1})} \quad \text{for } 1 \leq i \leq L \tag{3.70}$$

where

$$f'(s_o, q_1) \equiv 1$$

and

$$u_{L+1} = \frac{1}{f'(s_L, q_{L+1})} - \frac{1}{f\,(s_{L+1}, q_{L+1})} \tag{3.71}$$

and

$$u_i = \frac{1}{f(s_{i-1}, q_i)} - \frac{1}{f(s_i, q_{i+1})} \quad \text{for } L+1 < i \leq K \tag{3.72}$$

Returning now to the probability of an executing program accessing level i, u_i, we may write

$$u'_i = \frac{u_i}{\sum_{i=1}^{K} u_i} \qquad \text{for } 0 \leq i \leq K$$

$$= 0 \qquad \text{for } K+1 \leq i \leq N \tag{3.73}$$

Noting that

$$\sum_{i=1}^{K} u_i = 1 - \frac{1}{f(s_K, q_{K+1})}$$

$$= \frac{f(s_K, q_{K+1}) - 1}{f(s_K, q_{K+1})} \tag{3.74}$$

Thus we may write

$$u'_i = \frac{f(s_K, q_{K+1})}{f(s_K, q_{K+1}) - 1} u_i \qquad \text{for } 0 \leq i \leq K$$

$$= 0 \qquad \text{for } K+1 \leq i \leq N \tag{3.75}$$

Defining

$\overline{\tau}_p$ = the average time required to complete a primary page fault. (3.76)

This may be expressed as

$$\overline{\tau}_p = \sum_{i=1}^{K} u_i T_i$$

$$= \frac{f(s_K, q_{K+1})}{f(s_K, q_{K+1}) - 1} \sum_{i=1}^{K} u_i T_i \tag{3.77}$$

Thus completing the development of an expression for the time required to complete a primary access.

3.11 Interaction Between CPU and Primary Storage

At this point we would like to consider the average time required to complete a CPU access cycle. We will define

$\bar{\tau}_c$ = the average time between CPU accesses to storage when a user program is executing or $\frac{1}{r'}$. (3.78)

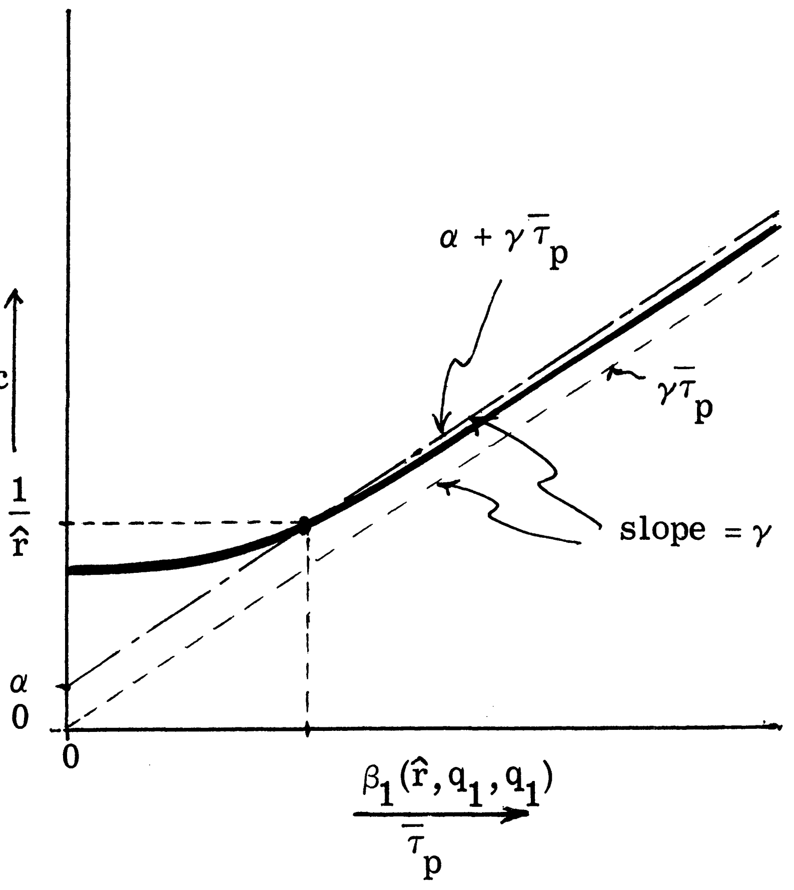
There are two factors which must be taken into consideration when determining the rate at which a CPU will make accesses. The first is the CPU itself and the second is the storage system to which the CPU makes accesses.

The solid line graph of Figure 3.25 shows the general relationship between $\bar{\tau}_c$ and $\bar{\tau}_p$. On the left where $\bar{\tau}_p$ is near 0, $\bar{\tau}_c$ is determined by the speed of the CPU and varies little with $\bar{\tau}_p$. On the right where $\bar{\tau}_p$ is much greater than the time required to actually execute a single instruction the value of $\bar{\tau}_c$ is determined almost solely by $\bar{\tau}_p$.

In a CPU which does not have a look ahead or more precisely can issue only one primary access (fetch) at a time, the slope of $\bar{\tau}_p$. as a function of $\bar{\tau}_p$ will approach 1 for large $\bar{\tau}_p$.

We will define γ as

γ = the slope of $\bar{\tau}_c$, as a function of $\bar{\tau}_p$, as $\bar{\tau}_p$ becomes large. (3.79)



$\bar{\tau}_c$ Versus $\bar{\tau}_p$

Figure 3. 25

If a CPU is capable of maintaining on the average several simultaneous primary accesses (look ahead capability) γ will have some value less than 1 but always greater than 0. We will assume that for those CPU's which have significant look ahead capability γ will be determinable either from hardware specifications or by experiment. Where the look ahead capability is not considered significant we will use $\gamma = 1$. We are ignoring the possibility that the look ahead capability itself will be directly affected by the storage system.

We will approximate $\overline{\tau}_c$ with a straight line of slope γ which passes through a known point in the $\overline{\tau}_c$ versus $\overline{\tau}_p$ curve. The approximation may be expressed as

$$\overline{\tau}_c = \alpha + \gamma \overline{\tau}_p \tag{3.80}$$

where α is a constant and is shown in Figure 3.25 with a center line.

In order to solve for α we consider the special case in which the rate at which the CPU makes accesses to the primary levels (while a program is executing) is highest. This will occur when there is only one activity in the entire system, that being the execution of a program which is contained entirely in the highest level of storage. We will define

$\hat{r}$ = the average rate at which the CPU makes primary storage accesses when executing a program which is entirely contained in the first level store and there is no other activity in the system. (3.81)

The variable $\hat{r}$ will be treated as a hardware specification and is generally equal to the maximum CPU access rate. Since in this special case $\overline{\tau}_c = \frac{1}{\hat{r}}$, we may write

$$\alpha = \frac{1}{\hat{r}} - \gamma \overline{\tau}_p \qquad (3.82)$$

Since the accesses are limited to the first level of storage

$$\overline{\tau}_p = T_1 \qquad (3.83)$$

or

$$\alpha = \frac{1}{\hat{r}} - \gamma T_1$$

$$= \frac{1}{\hat{r}} - \alpha \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C'_j, A'_j, q_k)[(1-w)q_j, k, 1, 0 + wq_j, k, 1, 1] \qquad (3.84)$$

where C'_j and A'_j may be written

$$C'_j = \hat{r} \sum_{k=1}^{N} [(i-w)t_j, k, 1, 0 + wt_j, k, 1, 1] \qquad (3.85)$$

and

$$A'_j = \frac{\sum_{k=1}^{N} q_k [(1-w)t_j, k, 1, 0 + wt_j, k, 1, 1]}{\sum_{k=1}^{N} [(1-w)t_j, k, 1, 0 + wt_j, k, 1, 1]} \qquad (3.86)$$

from Equations 3.37 and 3.38. This general solution for α appears more complex than it is in most cases. For instance, if an access to level 1 involves only the reading or writing of a record of size q_1 at level 1, as is almost always the case, α becomes simply

$$\alpha = \frac{1}{\hat{r}} - \gamma\beta_1(\hat{r}, q_1, q_1).$$

We may now write the average time required to complete a CPU access cycle as:

$$\bar{\tau}_c = \alpha + \gamma\bar{\tau}_p$$

$$= \alpha + \gamma \frac{f(s_K, q_{K+1})}{f(s_K, q_{K+1}) - 1} \sum_{i=1}^{K} u_i T_i \qquad (3.88)$$

We will now consider an expression for the average length of time that a user program executes before making a secondary access and thereby losing the use of the CPU we will define:

$\bar{\tau}_e$ = the average length of time that a user program executes before a secondary page fault. (3.89)

This may be expressed as:

$$\bar{\tau}_e = f(s_K, q_{K+1})\,\bar{\tau}_c \qquad (3.90)$$

where $f(s_K, q_{K+1})$ is the average number of accesses required to produce a secondary page fault.

and It is assumed that the time required to complete a CPU access cycle is independent of the number of accesses made.

$$\overline{\tau}_e = f(s_K, q_{K+1})[\, \alpha + \gamma \frac{f(s_K, q_{K+1})}{f(s_K, q_{K+1})-1} \sum_{i=1}^{K} u_i T_i \,]$$

(3.91)

Thus far we have developed an expression for the average time required to complete a secondary paging operation, $\overline{\tau}_s$, and the average length of time a user program spends in execution, $\overline{\tau}_e$. There is one other term that will enter into our final equations. We will define

$\overline{\tau}_h$ = the average amount of time spent performing the system housekeeping associated with a single user program passing through its cycle execution and secondary paging one time. (3.92)

This will be treated as an independent model variable.

3.12 System Performance

In analyzing the model for system performance we will assume that the system is either primary storage bound or secondary storage bound. When operating in a primary storage bound condition the CPU is assumed to be always busy and the rate at which work is done (rate of user program access) is based on the primary storage levels and the maximum CPU rate. When operating in a secondary storage bound condition it is assumed that the queue for CPU use is almost always empty and that the rate at which work is done is based primarily on the rate at which page faults to secondary levels are completed. It is fully acknowledged that when the system is operating in a near balanced condition this assumption will produce an optimistic performance figure.

Queueing models dealing with the execution queue in a more sophisticated manner have been considered. The increased complexity and additional solution time were weighed against the potential increase in model accuracy and the simpler model was chosen.

We will want to consider two performance equations - one based on primary bound operation and a second based on secondary bound operation. In analysis we will simply take the smaller of these two figures.

Primary Bound Performance

When the system is primary bound the CPU is almost always busy. If we observe the CPU we will find it serving one user program after another along with a certain amount of system activity associated with each user program's execution. We will refer to the period of time that a single user program and its associated system activity occupies the CPU during a single turn at the CPU as a CPU cycle. Figure 3.26 depicts a series of such CPU cycles. In Figure 3.26 the $\bar{\tau}_i$'s, random variables, represent the time spent servicing each user program.

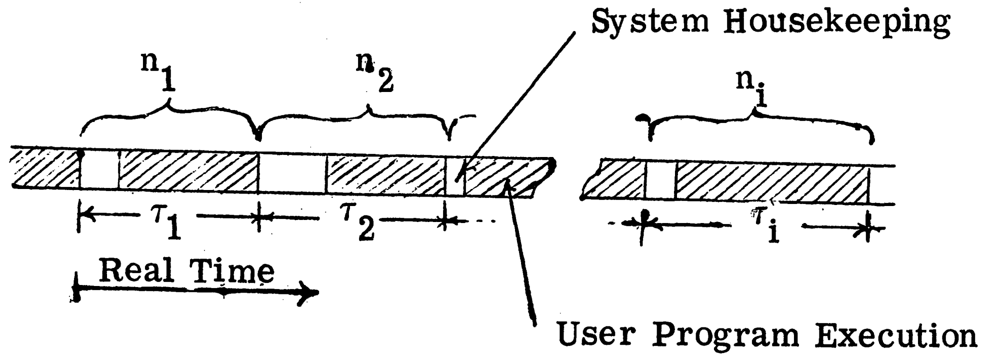


Figure 3.26 Primary Bound Operation

The η_i's, random variables, represent the number of user program accesses made during a single CPU cycle.

We will define

r_p = the average rate at which user program accesses are completed under primary bound conditions. (3.93)

We may express this as

$$r_p = \lim_{p \to \infty} \frac{\sum_{i=1}^{p} \eta_i}{\sum_{i=1}^{p} \tau_i}$$

$$= \lim_{p \to \infty} \frac{\frac{1}{p} \sum_{i=1}^{p} \eta_i}{\frac{1}{p} \sum_{i=1}^{p} \tau_i}$$

$$= \frac{\lim_{p \to \infty} \frac{1}{p} \sum_{i=1}^{p} \eta_i}{\lim_{p \to \infty} \frac{1}{p} \sum_{i=1}^{p} \tau_i} \qquad (3.94)$$

Notice that the η_i's are independent identically distributed random variables and the same can be said for the τ_i's.

We can recognize both the terms $\frac{1}{p} \sum_{i=1}^{p} \eta_i$ and $\frac{1}{p} \sum_{i=1}^{p} \tau_i$ as sample means, random variables whose variance goes to 0 as $p \to \infty$ and whose value becomes $E[\eta_i]$ and $E[\tau_i]$ respectively. Thus we may write

$$r_p = \frac{E[\eta_i]}{E[\tau_i]} \quad \text{for any i.} \tag{3.95}$$

The $E[\eta_i]$ will be simply $f(s_K, q_{K+1})$ and the $E[\tau_i]$ will be $\bar{\tau}_e + \bar{\tau}_h$ thus

$$r_p = \frac{f(s_K, q_{K+1})}{\bar{\tau}_h + \bar{\tau}_e} \tag{3.96}$$

Secondary Bound Performances

Our approach to the development of an expression for secondary bound performance will differ slightly from the approach taken with primary bound performance. Here we will examine the behavior of an individual program in the system in contrast to our focus on the CPU in the previous case.

In Figure 3.27 we show what we will call a program cycle. The program cycle consists of a period of system housekeeping, a period of user program execution and a period of secondary paging. Notice that here we are observing sequential cycles in the same user program. It should be understood that the system housekeeping will not all necessarily occur just prior to user program execution but may be spread throughout the program cycle.

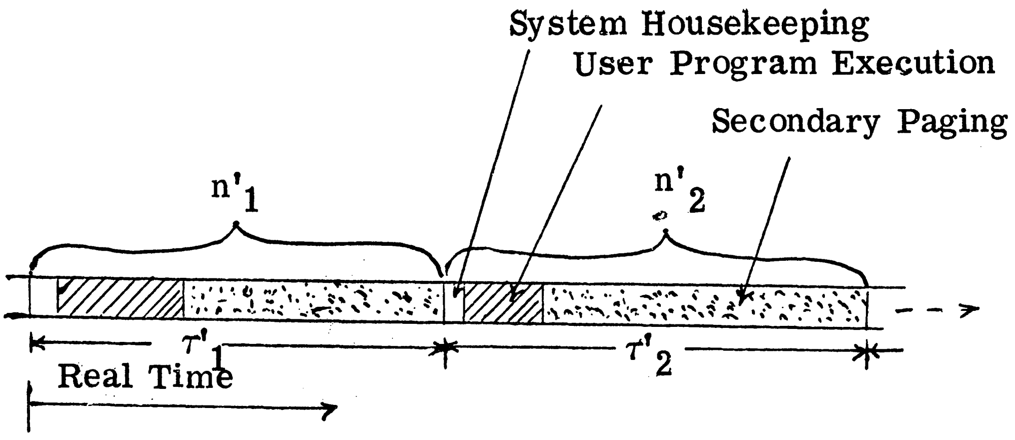


Figure 3.27 Secondary Bound Operation

Here the η'_i's, random variables, represent the number of accesses completed in the ith program cycle and the τ'_i's, random variables, represent the time required to complete the ith program cycle. Notice that there is no queueing delay for the CPU in secondary bound operation.

We will define

r_s = the average rate at which user program accesses are completed under secondary bound conditions. (3.97)

This may be written as

$$r_s = M \lim_{p\to\infty} \frac{\sum_{i=1}^{p} \eta'_i}{\sum_{i=1}^{p} \tau'_i} \qquad (3.98)$$

where M is the number of programs being multiprogrammed.

Following the same arguments used in the primary bound case we may write

$$r_s = M \frac{E[\eta'_i]}{E[\tau'_i]} \qquad \text{for any } i \qquad (3.99)$$

The $E[\eta'_i]$ is simply $f(s_K, q_{K+1})$ and $E[\tau'_i] = \overline{\tau}_h + \overline{\tau}_e + \overline{\tau}_s$ thus

$$r_s = \frac{M\, f(s_K, q_{K+1})}{\overline{\tau}_h + \overline{\tau}_e + \overline{\tau}_s} \qquad (3.100)$$

We now have an expression for r_p and r_s the average rate at which user program accesses are completed under primary bound and secondary bound conditions respectively. Both r_p and r_s are expressed in terms of r, the average rate at which user program accesses are completed, and r', the average rate at which user program accesses are completed during periods of user program execution. The first step in the solution of this system is to determine r' given r. To do this we must express r' as

$$r' = \lim_{p\to\infty} \frac{\sum_{i=1}^{p} N''_i}{\sum_{i=1}^{p} \tau''_i} \qquad (3.101)$$

where η''_i, a random variable, is the number of user program primary accesses made during the ith period of user program execution and τ''_i, a random variable, is the time required to complete the ith period of user program execution.

This becomes as before

$$r' = \frac{E[\eta''_i]}{E[\tau''_i]} \tag{3.102}$$

and continuing

$$r' = \frac{f(s_K, q_{K+1})}{\tau_e}$$

$$= \frac{1}{\alpha + \gamma \dfrac{f(s_K, q_{K+1})}{f(s_K, q_{K+1}) - 1} \displaystyle\sum_{i=1}^{K} u_i T_i} \tag{3.103}$$

where T_i for $1 \leq i \leq K$ is

$$T_i \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C'_j, A'_j, q_k) \; [(1-w) g_{j,k,i,o} + w \, g_{j,k,i,1}]$$

and

$$C'_j = r' Z'_j + r Z''_j$$

and

$$A'_j = \frac{r' Y'_j + r Y''_j}{r' Z'_j + r Z''_j}$$

We will now develop two additional expressions for r', one based on primary bound conditions, and a second on secondary bound conditions. We will begin with the primary bound case.

We will solve for r' given r and assuming that the system is primary bound. We will first solve for $\bar{\tau}_e$ directly from Equation 3.96

$$\bar{\tau}_e = \frac{f(s_K, q_{K+1}) - r_p \bar{\tau}_h}{r_p} \tag{3.104}$$

Substituting this into Equation 3.103 and dropping the p subscript for r_p we have

$$r_p' = \frac{rf(s_K, q_{K+1})}{f(s_K, q_{K+1}) - r\bar{\tau}_h} \tag{3.105}$$

where r_p' is r' assuming primary bound conditions.

Proceeding in the same fashion we may use Equations 3.100 and 3.103 to write:

$$r_s' = \frac{rf(s_K, q_{K+1})}{Mf(s_K, q_{K+1}) - r(\bar{\tau}_h + \bar{\tau}_s)} \tag{3.106}$$

where r_s' is r' assuming secondary bound conditions.

3.13 Solution

We have at this point generated a sufficient number of relations and we can now preceed in a reasonably orderly manner with the actual determination of r and r'. The solution for the 2 unknowns r and r' will involve either Equations 3.103 and 3.105 or Equations

3.103 and 3.106. The equations used depend on whether the system is primary or secondary bound.

We will now examine Equations 3.105 and 3.106 to determine their behavior as a function of r.

Since the term $\bar{\tau}_s$ in Equation 3.106 is a function of r we will begin by considering the behavior of $\bar{\tau}_s$ as a function of r. Repeating the expression for $\bar{\tau}_s$:

$$\bar{\tau}_s = f(s_K, q_{K+1}) \sum_{i=K+1}^{N} u_i T_i \qquad (3.107)$$

where T_i for $K+1 \leq i \leq N$ is

$$T_i = \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C_j, A_j, q_k) \left[(1-w)g_{j,k,i,o} + w g_{j,k,i,1}\right]$$

and

$$C_j = r Z_j$$

and

$$A_j = \frac{Y_j}{Z_j} \quad .$$

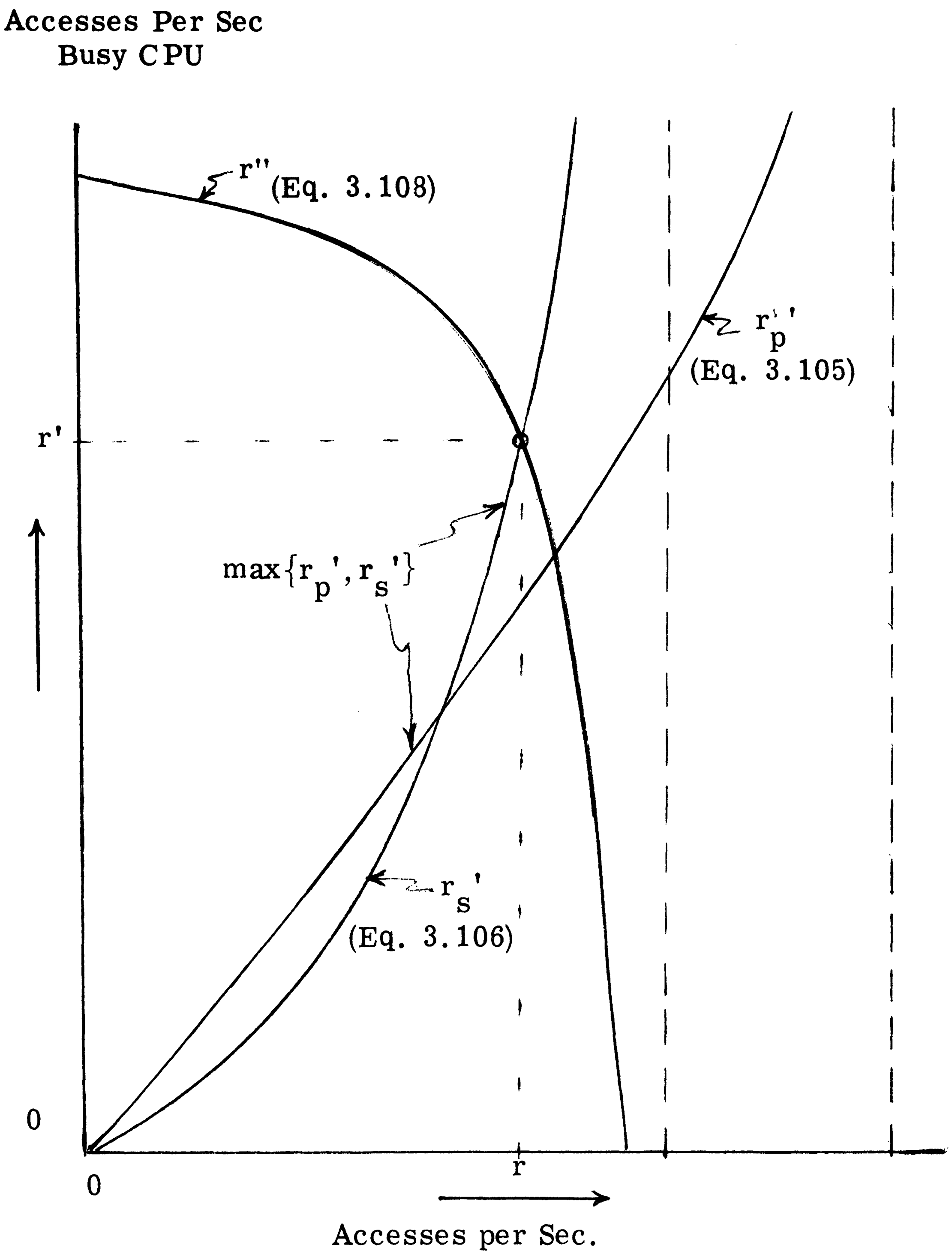
Since the $\beta_j(C_j, A_j, q_k)$'s are monotone non-decreasing with increasing r, $\bar{\tau}_s$ is also monotone non-decreasing with increasing r.

Returning to Equations 3.105 and 3.106 we see that in both cases the numerator is 0 for r = 0 and becomes larger with increasing r. The denominator has a positive and a negative term. The positive term is constant with respect to r and the negative term is 0 for r = 0 and increases with increasing r. Thus the denominator begins at some positive value for r = 0 and progressively becomes smaller for increasing r passing through 0 and continuing negatively. Thus both the expressions for r_p' and r_s' increase with increasing r, growing without bound for some value of r and then go negative.

Let us now see what this means in terms of the physical system we are modeling. Under either a primary bound assumption or a secondary bound assumption we see that as we increase the mean user program access rate, r, the rate at which the CPU must carry out these accesses during its busy periods increases. As r is increased we reach some point at which there is simply no time remaining for the CPU to perform user program accesses. As we near this point we see r_p' and r_s' grow without bound and become undefined. Under primary bound conditions this occurs when the CPU is totally consumed with making user program interchanges (i.e. $f(s_K, q_{K+1}) - r\bar{\tau}_h = 0$). Further increasing r we obtain negative results for r_p' and r_s'.

In Figure 3.28 we show typical curves for r_p' and r_s'. We display here only the positive behavior of r_p' and r_s' since these will be the only values of interest. The 2 vertical lines shown are assymtotic to r_p' and r_s' and indicate the point at which they are undefined.

In order to satisfy Equation 3.103 we will introduce a slight modification of Equation 3.103. We will replace r' on the left with r'' and in the right hand member by $\max\{r_p', r_s'\}$. This is written



Solution for r and r'

Figure 3. 28

$$r'' = \frac{1}{\alpha + \gamma \dfrac{f(s_K, q_{K+1})}{f(s_K, q_{K+1})-1} \sum_{i=1}^{k} u_i T_i} \qquad (3.108)$$

where

R_i for $1 \leq i \leq K$ is

$$T_i \sum_{j=1}^{N} \sum_{k=1}^{N} \beta_j(C_j', A_j', q_k) [(1-w)g_{j,k,i,o} + wg_{j,k,i,1}]$$

and

$$C_j' = \max \{r_p', r_s'\} Z_j' + rZ_j''$$

and

$$A_j' = \frac{\max\{r_p', r_s'\} Y_j' + rY_j''}{\max\{r_p', r_s'\}Z_j' = rZ_j''}$$

This equation is plotted in Figure 3.28 and labled r". The solution we are seeking will be at the intersection of r" and r_p' or the intersection of r" and r_s', whichever produces a smaller r. In the case shown, the desired solution is the intersection of γ'' with r_s' and it is marked with a small circle. The solution we are seeking is in fact the intersection of r" and $\max\{r_p', r_s'\}$. Notice first that this solution satisfies either Equation 3.105 or Equation 3.106 and Equation 3.103.

Second, we can show that there is at most one solution. Since r_p' and r_s' increase with increasing r, $\max\{r_p', r_s'\}$ must increase with increasing r. To show there is at most one solution,we will show that r" does not increase with increasing r. Note that the right hand member of Equation 3.103 is of the form:

$$\frac{1}{a + \sum_{j=1}^{N} \sum_{k=1}^{N} b_{j,k}\, \beta_j(C_j', A_j', q_k)} \tag{3.109}$$

where a and $b_{j,k}$ are constants which are independent of r'

and $\beta_j(C_j', A_j', q_k)$ is a monotone non-decreasing function of r and r'.

Equation 3.108 is identical in form except that r' has been replaced by $\max\{r_p', r_s'\}$ a monotone increasing function of r. Thus the denominator of Equation 3.108 is a monotone non-decreasing function of r and r" is a monotone non-increasing function of r .

Last we can see that the solution is based on the proper bounding condition. Recalling that we want to employ the most restrictive bounding condition, and with the described behavior of r", r_p', and r_s', we see that the solution of r" and $\max\{r_p', r_s'\}$ results in a minimum r.

The actual solution algorithm is based on a binary search over the variable r. The process of determining if a given trial r is too large or too small is carried out as follows:

1. Evaluate r_p' and r_s';if either is undefined or negative the trial r is too large.
2. Compare r" of Equation 3.108 with max $\{r_p', r_s'\}$: if $r'' > \max\{r_p', r_s'\}$, the trial r is too small

if $r'' < \max\{r_p', r_s'\}$ the trial r is too large

if $r'' = \max\{r_p', r_s'\}$ the trial r is the correct solution,

and $r'' = \max\{r_p', r_s'\} = r'$

We have at this point shown that given the system variables:

1. Number of levels N
2. Number of dedicated levels L
3. Number of primary levels K
4. Traffic matrices $\bar{t}_{i,\ell}$'s
5. Critical path matrices $\bar{g}_{i,\ell}$'s
6. Record size at each level q_k's
7. Maximum CPU rate $\hat{r}$
8. CPU look ahead factor γ
9. Mean system overhead time $\bar{\tau}_h$
10. Hardware behavior for each level $\beta_j(C, A, q)$'s
11. Storage capacity at each level Q_i's
12. Program behavior description f(s, a)
13. Fraction of write accesses w
14. Number of programs being multiprogrammed M

we may obtain a solution for the average rate at which user program accesses are processed.

Chapter 4

Hardware Model for Individual Levels of Storage

Here we will consider the modeling of the performance of the hardware at an individual level. Our primary concern is the average total time required for a read or write request to be processed at a given level.

There are many possible models which span a rather wide range between simplicity and reality. At one end of the spectrum we find the simple assumption that the average read/write time is some constant T. This is in fact the assumption made by Belady when applying his lifetime function [3]. In contrast to this simple assumption we find much more complex models at the opposite end of the spectrum. These models may involve a detailed treatment of the read/write request arrival distribution, channel queueing, service storage device queueing and service, and the relationship which exists between these operations. The most realistic of these require simulation or numerical solutions.

Here we are developing a storage hardware model to represent a single level of storage in an overall system model. In choosing a position in the spectrum between simplicity and reality three basic guidelines were specified at the outset:

1. We would like to take into consideration the effects of congestion at individual devices (such as core boxes and disk drives) as well as the congestion at the channels serving these devices.
2. The model should have an analytical solution.
3. We would like to be able to model all levels using the same basic model, the model being sufficiently general to adapt to different device types such as core, drum, and disk by relatively simple parameter variation.

As development proceeded, it was necessary to use some numerical techniques. It was also necessary to compromise 3. above to some degree.

4.1 Basic Model

Here we will consider the form of a basic model which will be used to model each level in the system. We will first describe the model and then discuss the effects of various assumptions in the model.

The basic model will consist of a simple queue and server as shown in Figure 4.1 along with its state transition diagram in Figure 4.2. The arrivals will be assumed Poisson and the service exponential. The number of requests in the queue will

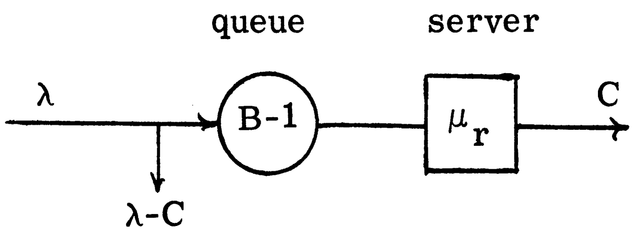


Figure 4.1 State Diagram for Basic Model

be limited to B-1 after which additional arrivals are lost. This will limit the total number in the system, queue and server, to B. The service rate μ_r will be treated as a function of the number of requests in the system at the storage level in question.

We will define

μ_r = service rate when there are r requests in the system (queue and server).

The value of the μ_r's will be determined by the hardware configuration at a given level. The determination of the value of the μ_r's will include such factors as the number of channels, number of devices, seek time, latency time and actual read/write times.

The value of B, the maximum number in the system, will be generated from the $\overline{T}$ array discussed in Chapter 3. This will be done simply by solving for the system conditions which produce the maximum number of read/write requests at the level in question.

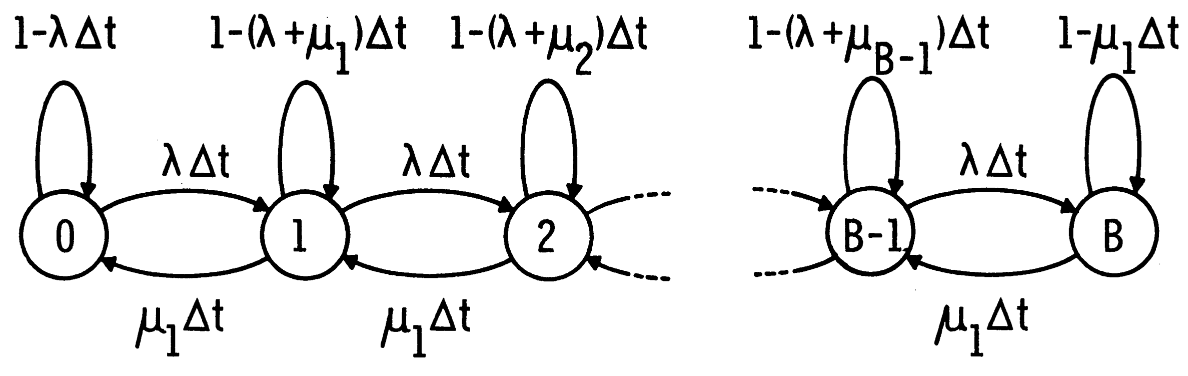


Figure 4.2 State Diagram for Basic Model

When solving for the mean total waiting time (queueing and service) we will know the value of C, the mean service rate. Hence it will be necessary to solve for λ given C, B and the μ_r's. The solution for λ will be numeric. Having λ the mean total waiting time may be obtained directly. Before considering a solution in detail and developing parameters for specific hardware configurations we will consider the assumptions of the basic model more closely.

We will begin by considering the effect of assuming that the server of Figure 4.1 is exponential. It is clear that the actual storage hardware which we may find at any given level may produce anything from exponential to deterministic service times. We will concern ourselves here with the effects of using an exponential service distribution to model a deterministic server such as a single core box. We must remember that the result which we seek from this model is the mean total waiting time. Thus we must consider how the actual mean total waiting time compares with the mean total waiting time given by the model. First we see that for low utilization or small λ, the model is accurate. Second, for high utilization, large λ, the results of the model are again accurate and agree with the real system. In this case we will almost always find the queue full (i.e. B requests in the system). Thus the mean total waiting time will be B/μ_B regardless of the service distribution (μ_B = service rate with B requests in the system). It is comforting to note that the agreement at both low and high utilizations is essentially independent of the distribution of service times produced by the hardware, and the distribution of interarrival times of service requests.

There will be some discrepancy between the model and the actual system in the region of moderate utilization. The model, if in error, will tend to produce a pessimistic result, i.e. a larger mean total waiting time.

A second source of possible error occurs in the assumption that requests for service have Poisson arrivals. In the case of arrivals we find that the distribution of interarrival times is dependent on essentially everything in the system except the level in question. The arrivals are in general the result of a combination of random events, the basic of driving event being the user program access. Data taken during operation of the Michigan Terminal System (MTS) a multiprogrammed page on demand system with virtual addressing, supports the assumption of Poisson arrivals. Specifically the data (pp. 113-120 of [45]) demonstrates that in this system (MTS) the intervals of CPU execution between interruptions (page faults and IØ interrruptions) have a characteristically exponential distribution. This indicates that access to levels below level 1 will tend to be Poisson but says nothing about about level 1 in the hierachy. The distribution of interarrival time at the first level of storage will most likely be dominated by times directly related to the CPU's cycle.

It is in the first level of storage that we will most likely incur the largest error. This is simply because here we are most likely to see deterministic or near deterministic service and the least random of arrivals. In terms of the macroscopic model of the previous chapter this may have the effect of introducing an error in the rate at which instructions are executed when a program is in execution. Consider however the development of the equation for $\overline{\tau_c}$ (the average time between CPU accesses to storage when a user program is executing) in Section 3.11. Here we see that for small values of $\bar{\tau}_c$ any errors introduced by the first level are completely canceled by the choice of the constant α. This will cause these errors if any to reappear for large values of $\bar{\tau}_c$ where their effect is greatly reduced.

There is one additional restriction which we will want to add to the model of Figure 4.1. That is we will limit the values of μ_r to those which satisfy:

$$\mu_r \leq \mu_{r+1} \tag{4.1}$$

This simply means that an additional request in the system (single level of storage) does not reduce the rate at which the system is servicing requests. In general this will always be the case when modeling realistic hardware configurations. It is difficult to imagine a system in which having an additional request in queue causes a reduction in the rate at which requests are serviced.

Consider a level composed of several disk drives. The more requests in the system at any given time the less likely it will be to find idle disk drives. The same is true for any level in which we can carry out service in parallel. In addition to the effects of parallel service, large queues can improve the utilization of a single device in the system. For instance the motion required of a disk arm can be reduced when the queue of requests is large and proper scheduling is used. In the worst case we would expect that $\mu_r = \mu_{r+1}$.

An example of this would be a level of storage composed of a single core box. The scheduling of requests here is irrelevant.

We will continue at this point with the solution to the basic model and specific determination of the μ_r's for various device configurations.

Our first step in the solution will be to solve for B using the $\overline{T}$ array discussed in Chapter 3. Recalling that the elements of the $\overline{T}$ array $\overline{t}_{j,k,i,\ell}$ are defined

$t_{j,k,i,\ell}$ = the number of records of size q_k read or written at level j as a result of a user read (ℓ=0) or write (ℓ=1) to level i.

The number of records read and written at level j as a result of a user read (ℓ=0) or write (ℓ=1) to level i will be simply $\sum_{k=1}^{N} t_{j,k,i,\ell}$. The maximum number of read and writes which a single program can generate at level j will be

$$\max_{\substack{i\epsilon[1,N] \\ \ell\epsilon[0,1]}} \{ \sum_{k=1}^{N} t_{j,k,i,\ell} \} \tag{4.2}$$

where i and ℓ are integers and N, an integer, is the number of levels in the hierarchy.

When M programs are being multiprogrammed the maximum number of read and write requests at level j, B_j, may be taken as

$$B_j = \max_{\substack{i\epsilon[1,N]\\ \ell\epsilon[0,1]}} \left\{ \sum_{k=1}^{N} t_{j,k,i,\ell} \right\}$$

for M = 1

$$= \max_{\substack{i\epsilon[1,N]\\ \ell\epsilon[0,1]}} \left\{ \sum_{k=1}^{N} t_{j,k,i,\ell} \right\}$$

$$+ (M-1) \max_{\substack{i\epsilon[K+1,N]\\ \ell\epsilon[0,1]}} \left\{ \sum_{k=1}^{N} t_{j,k,i,\ell} \right\}$$

for $M > 1$ (4.3)

where i and ℓ are integers and N, an integer, is the number of levels in the hierarchy.

In the case for $M > 0$ we see two terms, the first with a range of i from 1 to N and a second with a range of i from K+1 to N. This is necessary because we can have only one user program making a primary access, i.e. an access to the first K levels.

As we proceed we will be focusing on the performance of a single level. Thus for convenience of notation we will not use a subscript on the variable B.

We will now turn our attention to the determination of the steady state probabilities of the Markov model shown in Figure 4.2. We will consider a slightly more general model so that the results will be useful for other purposes later in this chapter.

General Solution

The state diagram for the general model is shown in Figure 4.3. The only difference between this state diagram and the one of Figure 4.2 is that the arrival rate λ is taken to be a function of state and thus is expressed as $\lambda_o, \lambda_1, \ldots, \lambda_N$. We have also used the variable N to indicate the maximum number of requests in the system.

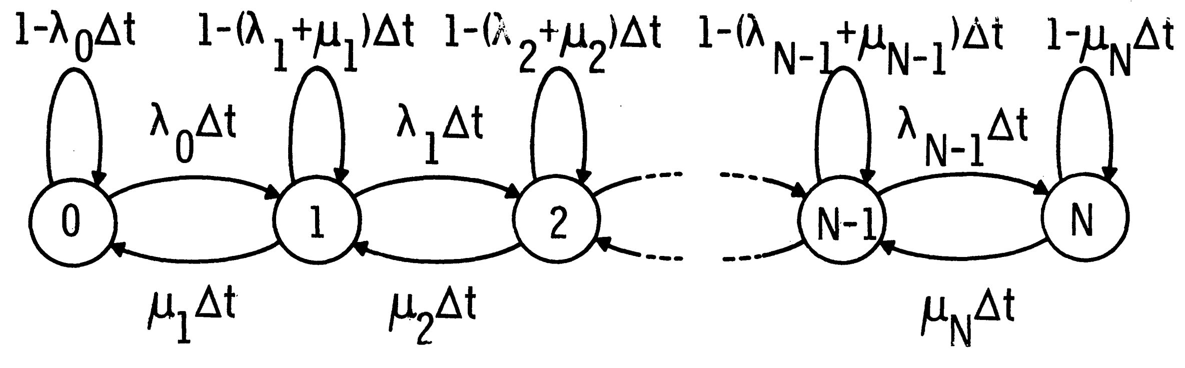


Figure 4.3 State Diagram for General Model

The stationary probabilities are (p. 44 of [17]):

$$P_o = \frac{1}{1 + \frac{\lambda_o}{\mu_1} + \frac{\lambda_o \lambda_1}{\mu_1 \mu_2} + \ldots + \frac{\lambda_o \lambda_1 \cdots \lambda_{N-1}}{\mu_1 \mu_2 \cdots \mu_N}}$$

and

$$P_i = \frac{\frac{\lambda_o \lambda_1 \cdots \lambda_{i-1}}{\mu_1 \mu_2 \cdots \mu_i}}{1 + \frac{\lambda_o}{\mu_1} + \frac{\lambda_o \lambda_1}{\mu_1 \mu_2} + \ldots + \frac{\lambda_o \lambda_1 \cdots \lambda_{N-1}}{\mu_1 \mu_2 \cdots \mu_N}} \qquad (4.4)$$

for $1 \leq i \leq N$

4.2 Solution of Basic Model

Returning to the specific we may rewrite Equation 4.3 and 4.4 substituting B for N and setting $\lambda_i = \lambda$ for $0 \leq i \leq B-1$.

$$P_o = \frac{1}{1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \qquad (4.5)$$

and

$$P_i = \frac{\frac{\lambda^i}{\mu_1 \mu_2 \cdots \mu_i}}{1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \qquad (4.6)$$

For the moment at least we will assume that we have the μ_i's and concentrate on solving for the mean waiting time as a function of the traffic at the level.

In terms of the system external to the specific level being discussed we do not know the arrival rate λ but rather the average rate at which request are serviced. We will define C as the average rate at which requests are serviced (mean traffic), ignoring in this context the complexities of specifying the level involved. If we add appropriate subscripts to designate levels, C_j will appear as defined in earlier chapters.

In order to relate C and λ we may write

$$C = \sum_{i=1}^{B} \mu_i P_i \tag{4.7}$$

$$= \frac{\lambda\left(1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^{B-1}}{\mu_1 \mu_2 \cdots \mu_{B-1}}\right)}{\left(1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^{B-1}}{\mu_1 \mu_2 \cdots \mu_{B-1}}\right) + \frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \tag{4.8}$$

defining

$$P(\lambda) = 1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^{B-1}}{\mu_1 \mu_2 \cdots \mu_{B-1}} \tag{4.9}$$

we may write

$$C = \frac{\lambda P(\lambda)}{P(\lambda) + \frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \tag{4.10}$$

Before we consider a solution for λ we will examine the behavior of C as a function of λ. The general relationship between C and λ is shown in Figure 4.4 and can be verified either from Equation 4.10 or from the basic queueing diagram of Figure 4.1.

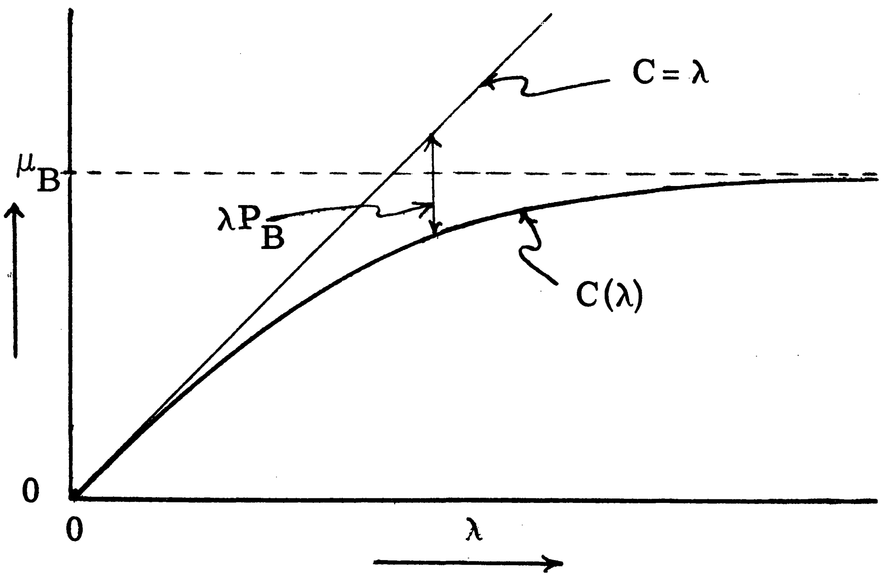


Figure 4.4 Service Rate Versus Arrival Rate

Since the mean loss rate is $\lambda\ P_B$ we can see that the $C(\lambda)$ curve falls below the λ curve by that amount. As λ becomes large $P_B \longrightarrow 1$ and C becomes asymptotic to μ_B.

We will now turn to the problem of finding λ given C. We may rewrite Equation 4.7 in the form of a polynomial in λ:

$$0 = C + \left(\frac{C}{\mu_1} - 1\right)\lambda + \frac{1}{\mu_1}\left(\frac{C}{\mu_2} - 1\right)\lambda^2 + \ldots + \frac{1}{\mu_1 \cdots \mu_{i-1}}\left(\frac{C}{\mu_i} - 1\right)\lambda^i +$$

$$\ldots + \frac{1}{\mu_1 \cdots \mu_{B-1}}\left(\frac{C}{\mu_B} - 1\right)\lambda^B \quad (4.11)$$

We may show that there is only one solution to this equation using the constraint of Equation 4.1 and Descartes' rule of signs (p. 28 [10]) which may be stated

(p. 28 of [11]) no polynominal can have more real positive zero's tnan its coefficients have changes of sign from + to - , and from - to +.

Examining Equation 4.11 with the constraint of 4.1 in mind we see that we have at most a single change in sign and thus at most a single real positive root. The solution for λ although numerical is straightforward.

Having obtained λ we may compute the P_i's followed immediately by the mean number in the system $\sum_{i=1}^{B} i P_i$. Knowing the mean number in the system and the mean rate at which requests arrive and the mean service rate C we may write [33]

$$\beta' = \frac{\sum_{i=1}^{B} i\, P_i}{C} \tag{4.12}$$

where β' is the mean time which an individual request spends in the system.

Expanded

$$\beta' = \frac{1}{C} \left(\frac{\frac{\lambda}{\mu_1} + 2\frac{\lambda^2}{\mu_1 \mu_2} + \ldots + B\frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}}{1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \right) \tag{4.13}$$

4.3 Solution Algorithm

Successive approximations will be obtained using Newton's method.

We write

$$\lambda' = \lambda - \frac{f(\lambda)}{f'(\lambda)} \tag{4.14}$$

where

$$f(\lambda) = C - \frac{\lambda\, P(\lambda)}{P(\lambda) + \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \tag{4.15}$$

Taking the derivative

$$f'(\lambda) = - \frac{P(\lambda)^2 + P(\lambda)\dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B} + \lambda\, P'(\lambda)\dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B} - P(\lambda)\, B\, \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}}{\left(P(\lambda) + \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}\right)^2} \tag{4.16}$$

Solving for λ' using the above equations

$$\lambda' = \lambda + \frac{\left(P(\lambda) + \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}\right)\left[C\left(P(\lambda) + \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}\right) - \lambda\, P(\lambda)\right]}{P(\lambda)\left[P(\lambda) - \dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}(B-1)\right] + \lambda P'(\lambda)\dfrac{\lambda^B}{\mu_1 \mu_2 \cdots \mu_B}} \tag{4.17}$$

recalling that

$$P(\lambda) = 1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1 \mu_2} + \ldots + \frac{\lambda^{B-1}}{\mu_1 \mu_2 \cdots \mu_{B-1}} \tag{4.18}$$

we may write

$$P'(\lambda) = \frac{1}{\mu_1} + 2\frac{\lambda}{\mu_1 \mu_2} + 3\frac{\lambda^2}{\mu_1 \mu_2 \mu_3} + \ldots + B-1\,\frac{\lambda^{B-2}}{\mu_1 \mu_2 \cdots \mu_{B-1}} \tag{4.19}$$

and

$$\lambda\ P'(\lambda) = \frac{\lambda}{\mu_1} + 2\frac{\lambda^2}{\mu_1\mu_2} + 3\frac{\lambda^3}{\mu_1\mu_2\mu_3} + \ldots + (B-1)\frac{\lambda^{B-1}}{\mu_1\mu_2\cdots\mu_{B-1}} \tag{4.20}$$

In computing $P(\lambda)$ and $\lambda\ P'(\lambda)$ we will make use of the common factors $\frac{\lambda^i}{\mu_1\mu_2\cdots\mu_i}$.

A FORTRAN program follows which performs the search for λ and computes β'. The following program variables correspond to the terms of Equation 4.15 and other associated variables.

P	$P(\lambda)$
DP	$\lambda\ P'(\lambda)$
F	$\frac{\lambda^i}{\mu_1\mu_2\cdots\mu_i}$
R	$\frac{\lambda^B}{\mu_1\mu_2\cdots\mu_B}$
S	$(P(\lambda) + \frac{\lambda^B}{\mu_1\mu_2\cdots\mu_B})$
FL	λ
FLP	λ'
IB	B
C	C
MU(I)	μ_i
BETA	β

```
     FUNCTION  BETAρ (C, IB, MU)
     REAL  MU (10)
     FL = C
     K = IB - 1
1    F = 1.
     DP = 0.
     P = 1.
     DO  2  I = 1,  K
     F = F * (FL/MU(I))
     P = P + F
2    DP = DP + F * I
     R = F* (FL/MU(IB))
     S = R + P
     FLP = FL - (S * (C * S - FL * P))/ (P * (P - R * (IB - 1)) + DP * R)
     IF (ABS (FLP - FL)/FLP. GT. 0.01) GO TO 1
     BETA = (DP + IB * R)/ (C * S)
     RETURN
     END
```

This rather short FORTRAN function tends to summarize the results of this section rather well. The function BETA provides the mean total service time for a request given the mean service rate C, the maximum number in the system B and the service rates μ_r as a function of the number in the system r. The remaining task is to find the μ_r's for different device types.

4.4 Disk Model

We will begin by considering the disk because the results can be simplified to obtain other results and disks are reasonably common and well understood devices.

First we must consider how a disk access is carried out. Consider the following steps:

1. Wait for the required arm to become free. As each read or write task enters the system for service it must acquire the service of a particular disk arm for its entire period of service.
2. Wait for free channel. When an arm becomes free we must acquire a channel so that the seek operation can be initiated.
3. Initiate seek operation. As soon as the use of a channel is obtained the seek operation is started. It is important to understand that the use of a channel is required only to initiate the seek operation and that the time involved is negligible. The channel is not held during the actual seek operation.
4. Wait for seek to complete. This involves the mechanical positioning of the disk arm over the desired track. Typical disks (IBM 2314) require a little less than .1 sec. for this operation. Note again that the channel is not tied up during this operation.

5. Wait for channel. At this point the arm is positioned and the task enters into contention for the use of a channel.

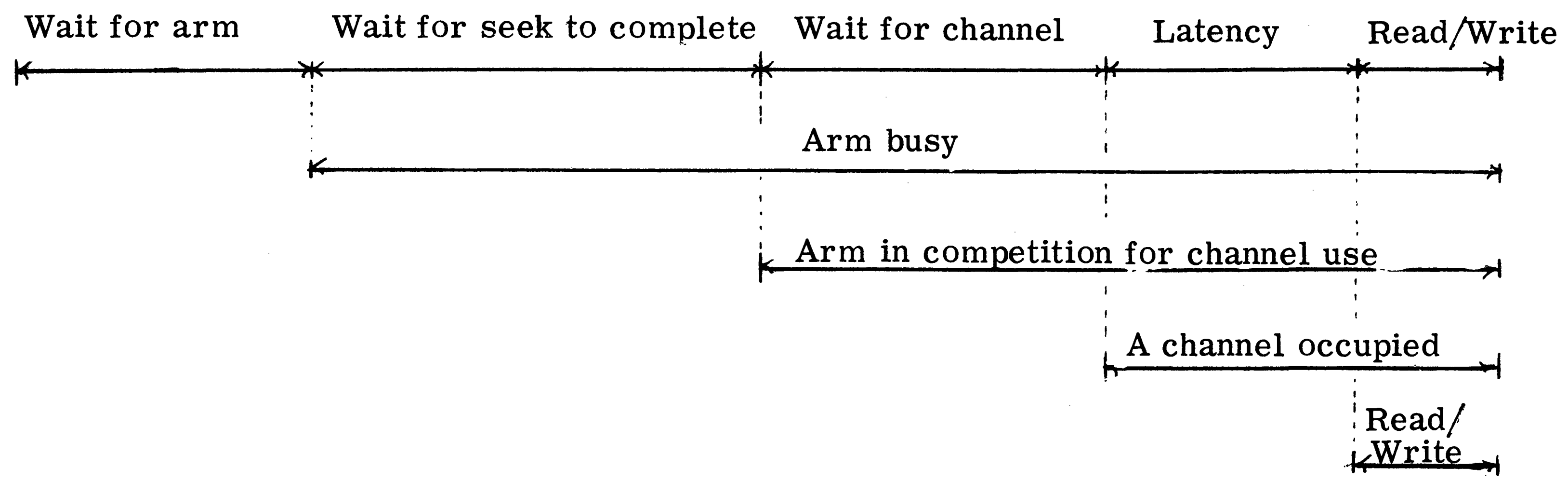
6. Latency. As soon as a channel is acquired we begin to wait for the rotation of the disk to bring the first of the desired data under the read/write heads. This is generally about 1/2 a rotation or 8 milliseconds for an IBM 2314.

7. Actual read or write with data transfer. The actual time required for the read or write operation may be only a few milliseconds depending on the length of the record.

We will assume at the outset that the time spent performing Steps 2 and 3 is negligible. Step 3 is obviously negligible since it requires only a few micro-seconds to complete in an environment where 10's of milliseconds are common. Step 2 is a bit more complex.

This wait for a channel is in order to initiate a seek operation. It is only reasonable to place these requests for use of the channel at the head of the channel use queue and complete this operation as soon as possible since it requires a negligible amount of channel time.

Assuming that requests for channel service are handled in this way we can consider the effect of this delay. First, when the system is lightly loaded a channel will almost always be free and



Disk Service Timing Diagram

Figure 4.5

no delay is incurred. Second, under heavy load we may find the disk arms almost always busy. Notice that at the end of a complete read or write operation both an arm and a channel become free at the same time. If a task is waiting in queue for that arm it may use the channel freed with the arm to initiate a new seek. Therefore in a heavily loaded system, i. e. busy arms, there will almost never be a wait for a channel to initiate a seek operation. This delay only occurs when a request arrives for an idle arm and all the channels are busy.

Lastly, there is the case where the delay does occur but is of little consequence. This is again in the heavily loaded system but here we are concerned with heavy channel loading rather than heavy arm loading. When the channels are almost always busy the delay in getting the seek started has only a small effect if any on the total waiting time.

Figure 4.5 illustrates the delays which occur along with the periods during which the arm and channel facilities are tied up.

The basic goal of this section is to express the mean service completion rate of a system of disk drives when there is some fixed number of requests in the system. We will define

$$r = \text{number of request in the system}$$

Thus in the specific case of r requests we are seeking μ_r.

One of the basic factors in solving for μ_r is the manner in which the requests for disk arm use distribute themselves among the available arms.

If we are in state r of the basic model then we know that there are r requests in the system. In the case of disks these requests queue for the use of individual disk arms. We may find that all r requests are in the queue for a single disk arm. In this case we would expect that the bottleneck would be the disk arms and that there would be no congestion at the channel or channels. At the other extreme we may find the r requests spread uniformly over all the arms in the system. Here we would expect to find the bottleneck at the channel(s). This is all dependent on the number of channels, the number of disk arms, r and other hardware specifications.

Our basic goal here is to find the mean completion rate μ_r when there are r requests in the system. To do this we will consider the steady state behavior of the system when m arms are busy. Where m can be any integer value from 0 to N_A where N_A is the number of disk arms. More specifically we will solve for the mean service completion rate when m arms are busy. Defining

$\mu(m) \equiv$ mean completion rate when m arm busy

we may write

$$\mu_r = \sum_{m=1}^{N_A} p_m(r, N_A)\ \mu(m) . \tag{4.21}$$

where

$p_m(r, N_A)$ = the probability that m arms are busy when N_A arms are servicing r requests.

$$= \frac{\binom{N_A}{N_A - m}\binom{r-1}{m-1}}{\binom{N_A + r-1}{r}} \qquad (4.22)$$

The development of $p_m(r, N_A)$ is considered in detail in Appendix B. We will continue here with the development of and expression for $\mu(m)$.

Mean Service Rate when M Arms Busy

At this point we are not concerned with queueing for disk arms. We are concerned with the very specific case where m arms are busy. The steps through which a busy arm proceeds in accomplishing a transfer are as follows:

1. Seek
2. Wait for channel
3. Latency + read/write

The seek operation is independent of other operations and does not occupy a channel. When the seek operation is complete the request entersa queue for channel service. Upon acquiring a channel the latency and read or write operations proceed. Figure 4.6 shows a queueing model which models these operations. We will use Poisson servers in this model to represent both the seek operation and the latency-read/write operation.

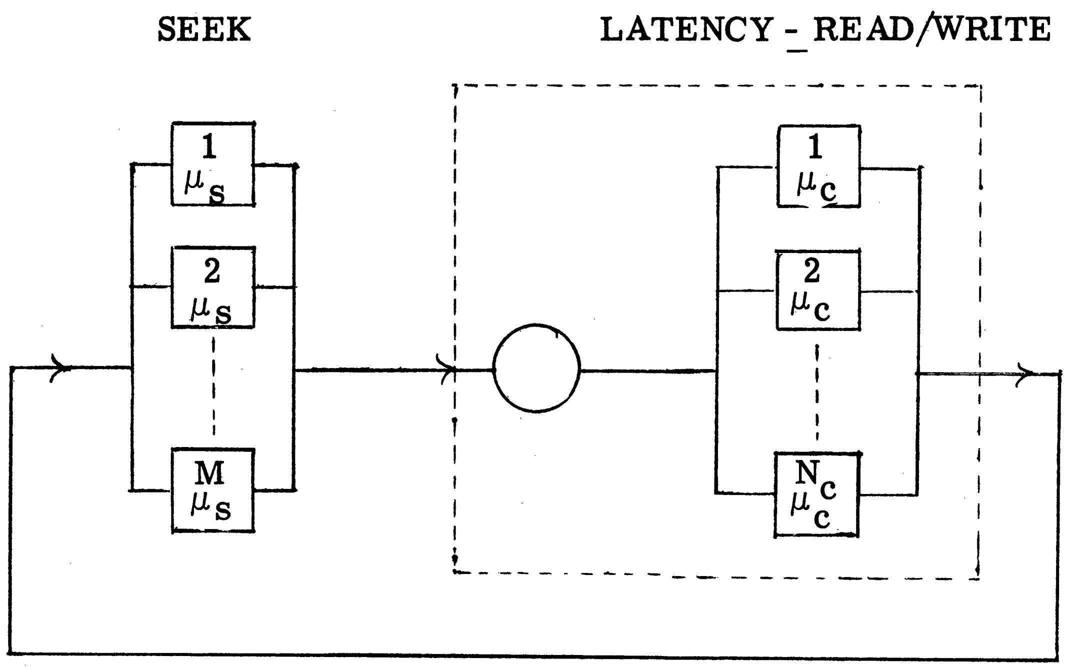
We will define

$$\mu_s = \text{mean seek service rate} \tag{4.23}$$

and

$$\mu_c = \text{mean (latency + read/write) service rate} \tag{4.24}$$

As a request enters the system (from the left) it enters one of the m servers of service rate μ_s. Since the system will have



μ_s = Mean Seek Time μ_c = Mean Channel Holding Time

Figure 4.6 Channel Queueing Model

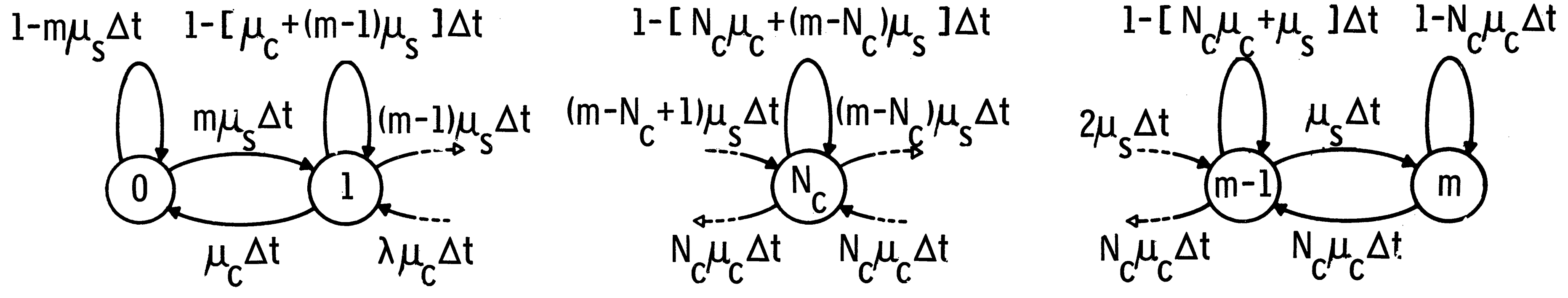
exactly m requests circulating, there will never be any queueing or blocking at the servers representing the seek time. Remember we are considering the case of m busy arms. When the requests complete the seek operation it enters the queue for channel service. Here the request waits for one of the N_c servers representing channel service. Channel service is required for the duration of the latency and read/write operations. After completing channel service the request reenters the system on the left, maintaining m arms busy.

The use of Poisson servers in this model permits the use a Markov model. We may define the states of a Markov model to correspond to the number of requests in that part of the system of Figure 4.6 that is contained in the dashed box. A state diagram with state transition probabilities is shown in Figure 4.7. Notice that this is analogous to the general model solved earlier in this chapter if we define

$$\lambda_i = (m-i)\,\mu_s \tag{4.25}$$

and

$$\begin{aligned} \mu_i &= i\,\mu_c \text{ for } i \leq N_c \\ &= N_c\,\mu_c \text{ for } i > N_c \end{aligned} \tag{4.26}$$



State Diagram for Channel Queueing Model

Figure 4.7

The stationary state probabilities may be obtained directly from Equation 4.4

$$P_o = \frac{1}{1 + \frac{\lambda_o}{\mu_1} + \frac{\lambda_o \lambda_1}{\mu_o \mu_1} + \ldots + \frac{\lambda_o \lambda_1 \ldots \lambda_{m-1}}{\mu_1 \mu_2 \ldots \mu_m}} \tag{4.27}$$

and

$$P_i = \frac{\frac{\lambda_o \lambda_1 \ldots \lambda_{i-1}}{\mu_1 \mu_2 \ldots \mu_i}}{1 + \frac{\lambda_o}{\mu_1} + \frac{\lambda_o \lambda_1}{\mu_o \mu_1} + \ldots + \frac{\lambda_o \lambda_1 \ldots \lambda_{m-1}}{\mu_1 \mu_2 \ldots \mu_m}} \tag{4.28}$$

We may express the mean completion rate $\mu(m)$ when there are m arms busy in terms of μ_s.

$$\mu(m) = \sum_{i=0}^{m} P_i (m-i) \mu_s \tag{4.29}$$

expanding

$$\mu(m) = \mu_s \frac{m + (m-1)\frac{\lambda_o}{\mu_1} + (m-2)\frac{\lambda_o \lambda_1}{\mu_1 \mu_2} + \ldots + \frac{\lambda_o \lambda_1 \ldots \lambda_{m-2}}{\mu_1 \mu_2 \ldots \mu_{m-1}}}{1 + \frac{\lambda_o}{\mu_1} + \frac{\lambda_o \lambda_1}{\mu_1 \mu_2} + \ldots + \frac{\lambda_o \lambda_1 \ldots \lambda_{m-1}}{\mu_1 \mu_2 \ldots \mu_m}} \tag{4.30}$$

where

$$\lambda_i = (m-i) \mu_s \tag{4.31}$$

$$\mu_i = i\, \mu_c \quad \text{for } i \leq N_c \tag{4.32}$$

$$= N_c \mu_c \quad \text{for } i \geq N_c \tag{4.33}$$

To obtain μ_r we simply write

$$\mu_r = \sum_{m=1}^{N_A} P_m(r, N_A)\, \mu(m) \tag{4.34}$$

Turning our attention to μ_c and μ_s, we will express

$$\mu_s = \frac{1}{\text{mean seek time}} \tag{4.35}$$

and

$$\mu_c = \frac{1}{\text{latency + read/write}}$$

$$= \frac{1}{\frac{1}{2} \times \frac{60}{\text{RPM}} + \frac{60}{\text{RPM}} \times \frac{A}{\text{BPT}}}$$

$$= \frac{\text{RPM}}{60\ (\frac{A}{\text{BPT}} + \frac{1}{2})} \tag{4.36}$$

where RPM = disk speed in revolutions per minute

BPT = bits per track

A = mean record size.

4.5 Data Cell Model

The data cell is an IBM device in particular the IBM 2321. The data cell records and retrieves information on magnetic strips. The IBM 2321 uses a total of 2000 strips each containing 100 tracks of data. The strips are stored vertically in a circular file which rotates to position the desired strip at the reading station. The desired strip is then separated from neighboring strips using a tab indexing technique. The desired strip is removed from the circular file and wrapped on a drum. The drum is read by a movable head which has five positions and twenty read/write heads providing access to the 100 tracks on the strip. After the above operations are complete a channel is acquired for the latency-read/write operation.

The data cell is analogous to the disk in that all the operations prior to the latency-read/write operation are carried out without the active use of a channel. Correspondingly the term seek is used for those operations which precede the latency- read/write operation.

We can use the same model for the data cell that was developed for disk by simply redefining the following terms:

N_A	=	# of access mechanisms = # data cells
m	=	# of busy access mechanisms
μ_s	=	data cell seek service rate
μ_c	=	data cell latency + read/write service rate.

Other variables are identical.

Since at this writing there is only one data cell (IBM 2321) to be considered we will specify μ_s and μ_c for this device.

$$\mu_s = \frac{1}{\text{mean seek time}} \tag{4.37}$$

for the IBM 2321

$$\mu_s = \frac{1}{.550} \tag{4.38}$$

and

$$\mu_c = \frac{RPM}{60\,(\frac{A}{BPT} + \frac{1}{2})}$$

for the IBM 2321

$$\mu_c = \frac{1200}{60\,(\frac{A}{16000} + \frac{1}{2})} \qquad (4.39)$$

These numbers assume random strip access and include strip replacement time.

Additional information on the IBM 2321 data cell may be obtained from IBM [50].

4.6 Core Model

The modeling of the congestion in core is much simpler than either the disk or data cell. First channel congestion is either non-existent or negligible. This removes the complexities involving channel independent and channel dependent operations.

The congestion in a system of cores occurs at the core boxes. In effect the queueing is for core drivers and sense windings. We are seeking the mean service rate for a system of N_A core boxes with r requests in the system μ_r. Assuming that core boxes are accessed randomly we may then express the mean service rate μ_r as

$$\mu_r = \sum_{m=1}^{N_A} p_m(r, N_A)\,\mu(m) \qquad (4.40)$$

where $\mu(m)$ = mean service rate when m core boxes are busy.

$$= \frac{m}{\text{core cycle time}} \tag{4.41}$$

4.7 Drum Model

Here as before we are seeking μ_r but here we are concerned with a drum. The drum poses a rather different problem when compared to those that we have treated previously. The access time and mean transfer rate for a drum can be improved considerably by ordering the read/write requests in the order which the information passes under the read/write heads. This improvement is sufficient to motivate most users of drums to go to the trouble to implement the ordering of requests in the drum management software. Correspondingly it would be unrealistic to ignore this improved operation when modeling a drum.

To begin let us consider what we mean by sectors and fields. The drum may have on the order of 1000 read/write heads (IBM 2301 has 800). These heads are divided into groups (possibly groups of one head) which work together in reading and writing information on the drum. The individual tracks covered by a single group of heads is called a field. The number of sectors is simply the number of records in a single field. Figure 4.8 shows the fields and sectors of a drum.

Drums exhibit certain characteristics which can complicate their modeling. One of these is that the read/write heads cannot be switched from writing to reading mode instantaneously. Generally speaking if a record is currently being written in a given field the

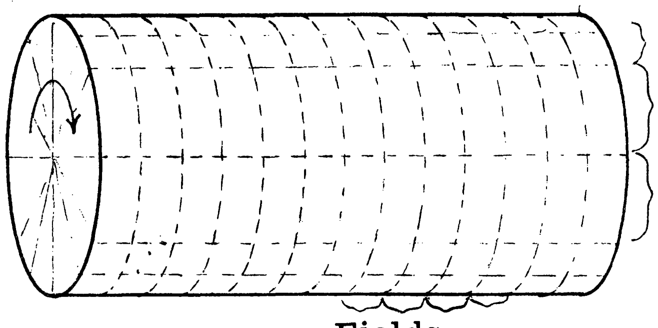


Figure 4.8 Drum Sectors and Fields

record in the same field of the next sector cannot be read. This problem can be circumvented in several ways:

1. Intersector gaps sufficiently wide to provide for head switching.
2. Request ordering to avoid unnecessary head switching.

We will ignore the possible delays caused by head switching, assuming either that some measure has been taken to completely avoid the problem or that the delays incurred are negligible.

We will make the assumption that the distribution of requests for sectors is the same as used previously for the disk data cell and cores. The difference here is that the service is occurring in a cyclic fashion rather than at random. Thus we may write

$$p_m(r, N_A) = \text{probability that m sectors have a request in queue when there are r requests and } N_A \text{ sectors.} \tag{4.42}$$

We will assume that the mean distance between the active sectors (sectors which have request in queue) is $\frac{N_A}{m}$ sectors. The mean distance to the starting point of the first of these m active sectors is $\frac{1}{2}\,\frac{N_A}{m}$ sectors. The mean distance between the starting points of each of the succeeding (m-1) active sectors is $\frac{N_A}{m}$ sectors or a total distance of $(m-1)\,\frac{N_A}{m}$ sectors. We must then continue to traverse the last of these sector a distance of one sector. Combining the total distance required to access m active sectors may be written:

$$\text{Total distance m active sectors} = \frac{1}{2}\,\frac{N_A}{m} + (m-1)\,\frac{N_A}{m} + 1$$

$$= N_A\left(1 - \frac{1}{2m}\right) + 1 \tag{4.43}$$

The time required to traverse a single sector is $\frac{60}{N_A \times \text{RPM}}$ seconds where RPM is the revolutions per minute of the drum.

Thus we may write the mean time required to access m active sectors

$$\text{Time to access m sectors} = \frac{60}{\text{RPM}}\left(1 - \frac{1}{2m} + \frac{1}{N_A}\right) \tag{4.44}$$

The mean rate of accesses will be

$$\mu(m) = \frac{m \times \text{RPM}}{60\left(1 - \frac{1}{2m} + \frac{1}{N_A}\right)} \tag{4.45}$$

and

$$\mu_r = \sum_{m=1}^{N_A} p_m(r, N_A) \frac{m \times \text{RPM}}{60\left(1 - \frac{1}{2m} + \frac{1}{N_A}\right)} \tag{4.46}$$

4.8 Effect of Specific Record Sizes

Thus far we have developed a model giving us the mean total service time for core, drum, disk and data cell. Each of these models is a function of the specific characteristics of the devices at the level in question as well as the service rate C and the mean record size A. Focusing on the mean total service time as a function of C and A we may write:

$\beta_j'(C, A)$ = mean total service time for a read or write request at level j when serving an average of C requests per second which are of an average record length A in bits. (4.47)

We would now like to consider the mean total service time when a record of specific size q is read or written at level j. We will define:

$\beta_j(C, A, q)$ = mean total service time for a read or write request involving a record of size q at level j when serving an average of C requests per second which are of an average size A. (4.48)

We will assume that the requests are generated independently. This means that given the size, q, of a specific record being serviced at a given level gives us no information about the size of the other records being serviced at that level. We may write

$$\beta_j(C, A, q) = \beta'_j(C, A) + \chi_j(q-A) \qquad (4.49)$$

where

$$\chi_j = \frac{1}{\text{Max transfer rate at level j (bits/sec)}} \qquad (4.50)$$

Notice that in all the models developed the size of a record has no effect on its total service time until the actual read or write operation takes place. The length of time required to complete this read/write operation is simply χ_j q where χ_j is the transfer rate. The term $\chi_j(q-A)$ gives us the variations about $\beta'_j(C, A)$ caused by differing record sizes.

Notice that if we take the expected value of $\beta_j(C, A, q)$ for a distribution of record sizes, q, we may write

$$E[\beta_j(C, A, q)] = E[\beta_j'(C, A)] + E[\chi_j q] - E[\chi_j A]$$

$$= \beta_j'(C, A) + \chi_j A - \chi_j A$$

$$= \beta_j'(C, A) \qquad (4.51)$$

At this point we have completed the development of $\beta_j(C, A, q)$ the mean total service time at level j, as a function of the mean service rate C, mean record size A, and the specific record size q.

Chapter 5
A Case Study in Analysis

Up to this point we have been concerned with the development of a general mathematical model of a computing system. In this chapter and the next we will be concerned with implementation and application of this model.

In the first section we will discuss briefly the computer program which was designed to implement the mathematical model developed in the preceding chapters. In the sections following (5.2 through 5.6) we will discuss a case study in analysis. Here we will describe a single system in some detail, study its performance, and then proceed to examine the changes in performance which occur when various model parameters are changed.

5.1 Program Description

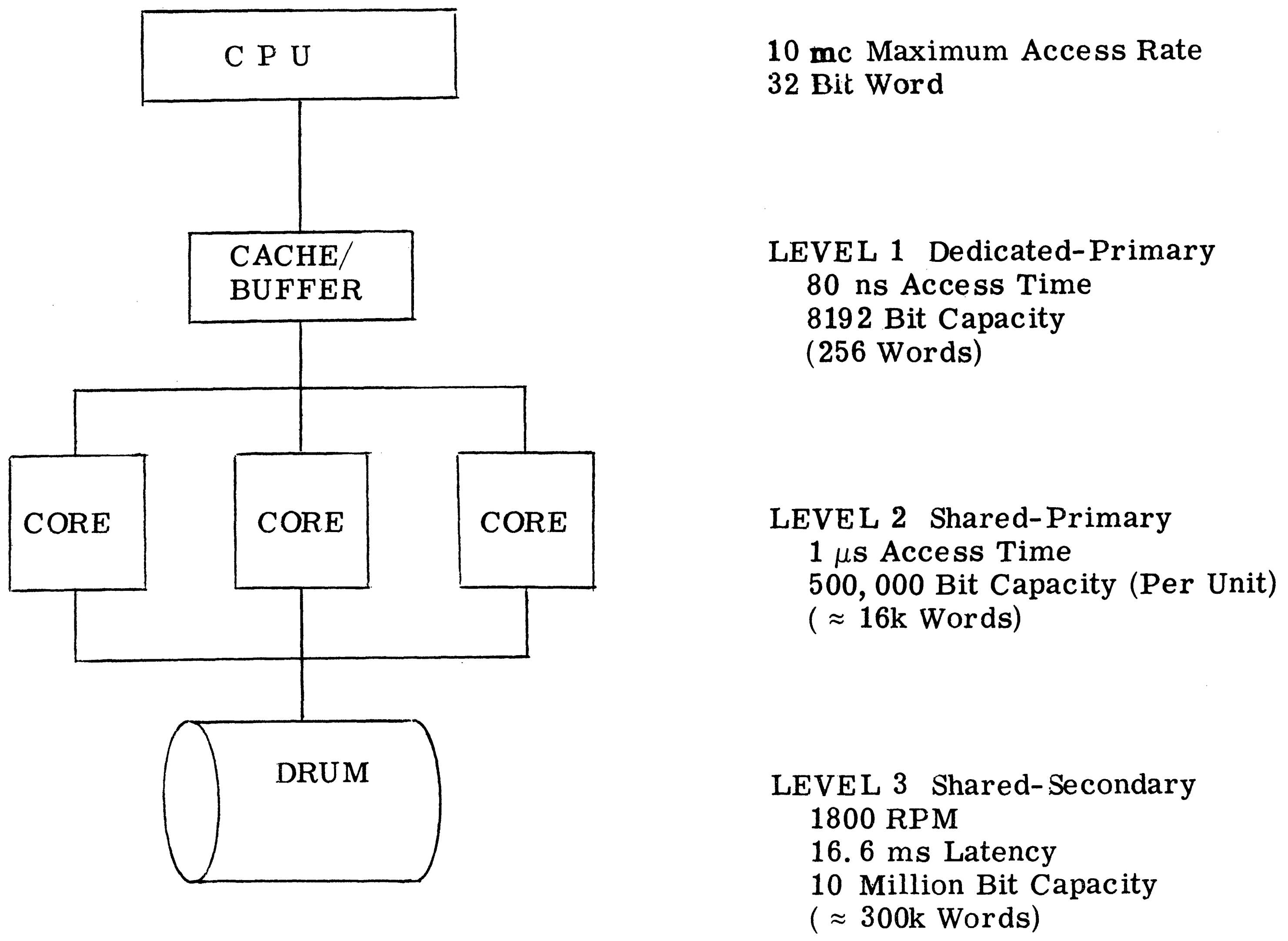
The design of the computer program implementing the system model closely follows the development of the system model described in Chapters 2, 3, and 4. The lifecycle function of Chapter 2 is implemented in a single 8-variable FORTRAN function. Each of the storage models described in Chapter 4 is implemented in a separate subroutine or group of subroutines. These models are all interfaced through a FORTRAN function BETA which corresponds to the β function discussed in Chapters 3 and 4. This FORTRAN function calls on the appropriate model based on the level of storage in question and returns the desired mean total service time as a function of the traffic at that level. The storage device model in effect for a given level is controlled by a simple index. The number of different storage device models is not limited and additional models may be added rather simply.

The macroscopic model described in Chapter 3 is implemented in several subroutines which compute the values of the u_i's, C_j's, C'_j's, A_j's, A'_j's etc. The method of solution is as described in Chapter 3.

In addition to the basic analysis program there are a number of subroutines which perform the functions of input, output, and program control. The program is designed to run in a highly interactive manner allowing the user to modify and display system parameters by using simple commands.

There is one important restriction which appears in the computer program and is not necessarily part of the mathematical model itself. The program assumes that all user programs are identical and correspondingly shared levels are allocated equally among the programs being multiprogrammed.

We will discuss several other aspects of the program as we consider the examples that follow. The optimization capability of the program will be discussed in the next chapter where we treat the optimization in detail.



SYSTEM DIAGRAM

Figure 5.1

5.2 System Description

In this section we will describe a multiprogrammed, demand paged computing system using a 3-level storage hierarchy. We will consider the system itself and its performance in considerable detail. The need for careful description here is motivated by the fact that this system will remain at the center of attention throughout our study of analysis and in the next chapter treating optimization.

The computing system in question is first shown in Figure 5.1. Here we see a CPU and 3 levels of storage. The CPU is capable of a maximum access rate of 10 mc. More precisely $\hat{r}$ as defined in Equation (3.81) is 10 mc. The CPU word size is 32 bits as noted.

The level 1 store is contrived to resemble the cache on the IBM 360/85 or the so-called buffer on the IBM 370 series. This cache or buffer store will be allocated entirely to the executing program and thus will be a dedicated level of storage. This level is also clearly a primary level since access to it will not cause loss of the CPU to another program. The level 1 store here is a random access device with an 80 ns access time and a 8192 bit or 256 word capacity.

The level 2 store consists of 3 core boxes. The space at this level will be allocated equally among the programs being multiprogrammed, making this a shared level. This level too is a primary level and access will not cause loss of the CPU. The access time at level 2 is 1 and each core unit has a capacity of 500,000 bits or approximately 16k words.

The level 3 store is shared-secondary and is implemented in the form of a drum. Shared indicates that each multiprogrammed program is allocated equally at this level and secondary indicates that user program access to the drum will cause loss of the CPU to another program. The drum at level 3 rotates at 1800 RPM and has a corresponding latency of 16.6 ms. The drum's capacity is 10 million bits or approximately 300 k words.

Figure 5.2 is the first of a number of figures which show the actual computer output generated by the modeling program. This and future figures containing this style of type were generated by issuing one or more display commands to the program.

The first 3 entries of Figure 5.2 give us the number of levels, the number of dedicated levels and the number of primary levels. These of course correspond to the system of Figure 5.1. Notice that where possible the variable names used earlier in model development are used here.

Continuing, we see the number of programs to be multiprogrammed has been set to 13. The next 2 entries establish limits of the values which M (the number of programs) may take on. The primary function of a minimum and maximum number of programs is in optimization where we are searching over a range of values for an optimum. We will discuss this in more detail in the next chapter, which treats optimization.

```
NUMBER OF LEVELS                                N=  3

NUMBER OF DEDICATED LEVELS                      L=  1

NUMBER OF PRIMARY LEVELS                        K=  2

NUMBER OF PROGRAMS MULTIPROGRAMED               M=   13

MIN # OF PROGRAMS TO BE MULTIPROGRAMED          MINM=    2

MAX # OF PROGRAMS TO BE MULTIPROGRAMED          MAXM=   25

MAXIMUM CPU RATE (ACCESS PER SEC.)              RHAT=  Ø.1ØØØØØØE Ø8

CPU OVERLAP FACTOR                              GAM=Ø.49999982

SYSTEM OVERHEAD TIME                            TAUH=  Ø.1499997E-Ø3


 TABLE OF RECORD SIZES

LEVEL  RECORD SIZE
  1          32
  2        1Ø24
  3       16384
```

SYSTEM VARIABLES

Figure 5.2

Next we see the maximum CPU access rate ($\hat{r}$ here represented as RHAT). This is followed by the CPU overlap factor. The value of the CPU overlap factor indicates that the CPU is roughly capable of keeping 2 access requests pending at once. This variable (represented here as GAM) corresponds to γ defined in Equation (3.79).

The next of these independent variables is the system overhead time $\bar{\tau}_h$ (represented here as TAUH). This is the CPU time required for program interchange. $\bar{\tau}_h$ is defined in Equation (3.92).

Finally at the bottom of Figure 5.2 we find a table of record sizes. This is the logical record size and corresponds to the q_i's where the i corresponds to the level.

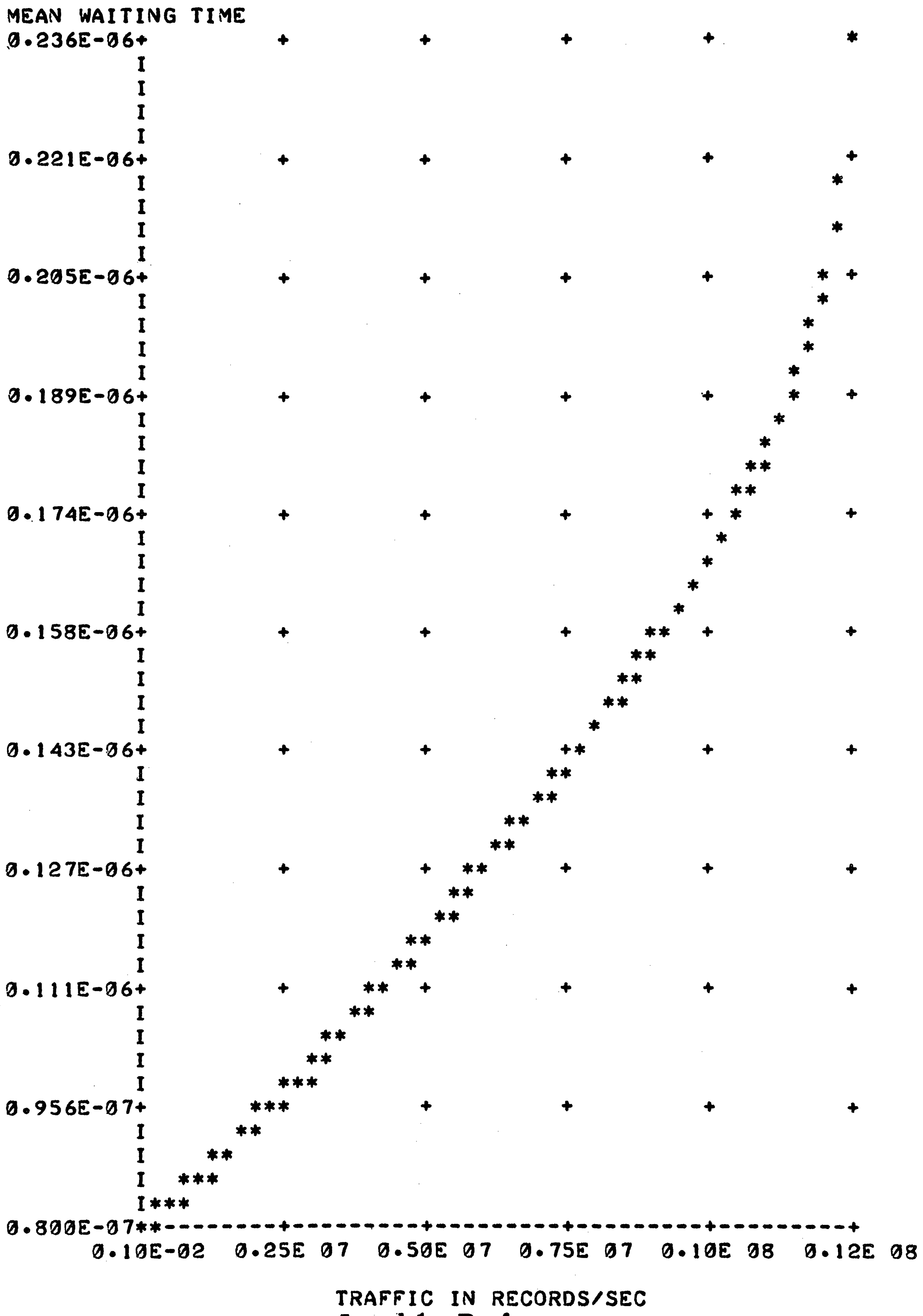
We will now turn our attention to the individual levels of storage themselves beginning with level 1. In Figure 5.3 we see a description of the hardware at level 1. Here we see that we are using model type 2 at level 1. This model corresponds to the core model developed in Section 4.6. The average access time for a single unit is 80 ns as indicated earlier. Note that all times are in seconds unless otherwise noted.

The basic device record size is 32 bits. The basic device record size here happens to be the same as the logical record size but this need not be the case. The device record size is the number of bits which are read or written in a single access time. The logical record size specifies the quantity of data "paged up". The unit capacity

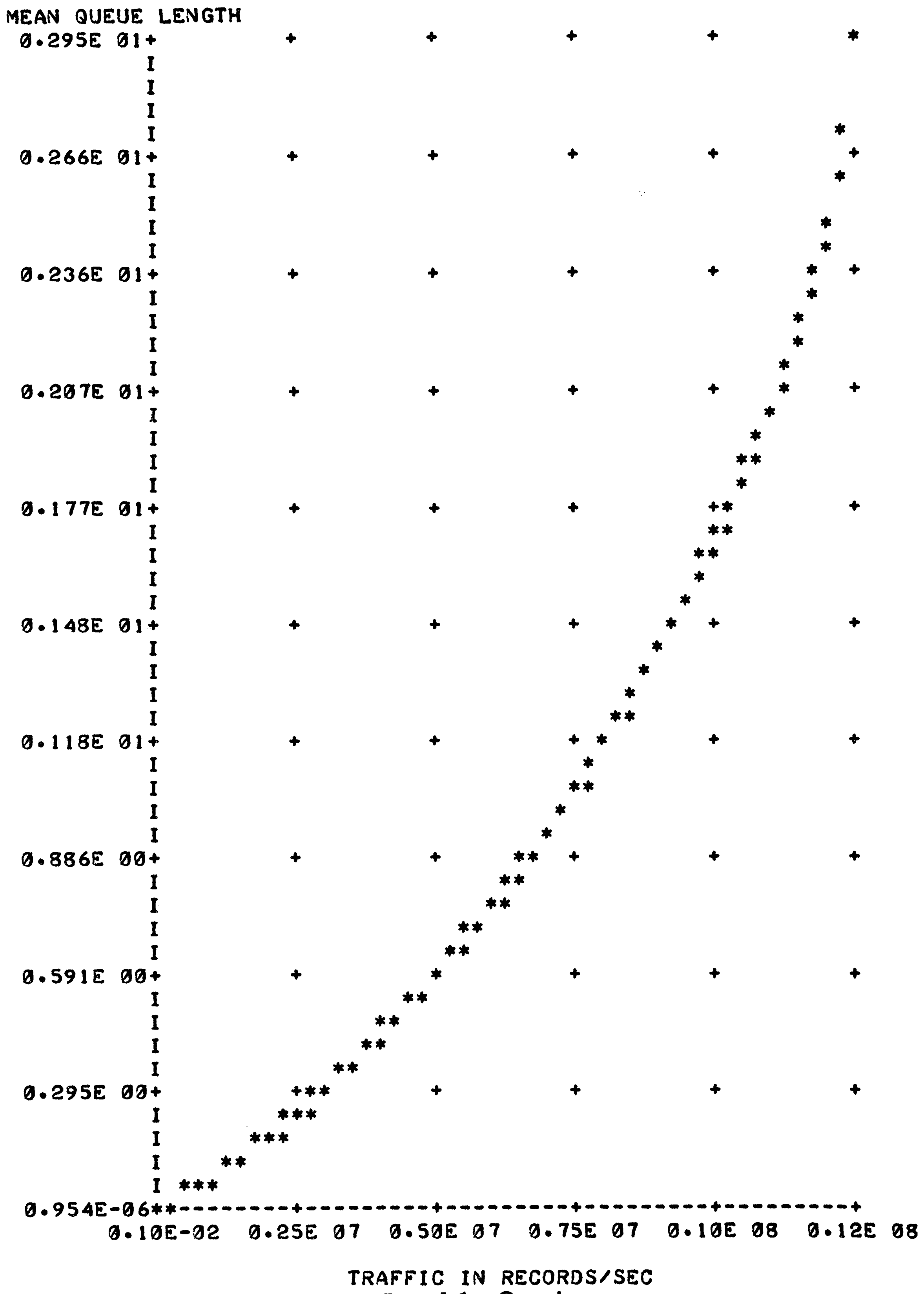
```
*** LEVEL 1 ***

MODEL TYPE  2

MODEL 2  (RANDOM ACCESS OR CORE UNITS)

AVERAGE ACCESS TIME FOR A SINGLE UNIT   0.7999967E-07

BASIC DEVICE RECORD SIZE                          32.

UNIT CAPACITY                           0.8192000E 04

NUMBER OF UNITS                           1

MIN # OF UNITS                            1

MAX # OF UNITS                            4

COST PER UNIT ($)                            50000.00
```

Level 1 Hardware Description

Figure 5.3



Level 1 Performance
Figure 5.4



Level 1 Queuing
Figure 5.5

is given as 8192 bits. Note that all capacities and record sizes are in bits. We see the number of units, the minimum number of units and the maximum number of units. Clearly the number of units is set to 1 which corresponds to our example. The minimum and maximum are bounds used in optimization when the number of units is a decision variable. Last we find the cost per unit at \$50,000. This value is used in optimization and, as you will see, for simplicity we have chosen a cost of \$50,000 per unit at each level of storage. This is a bit unrealistic but simplifies our discussion of optimization.

Turning to Figure 5.4 we have a plot generated at the terminal which displays the performance characteristics of the level 1 configuration. The plot gives us the mean waiting time for a single service request as a function of the rate at which requests arrive. The request arrival rate or traffic records per second is given on the horizontal axis and runs from near 0 to just less than 12.5 million records/sec. In generating this plot it was assumed that a maximum of 3 requests for service could be in queue and service (i.e. $B = 3$, see Section 4.1). It is also assumed that all the records being read or written are 32 bit records. In terms of the β function defined in Equation (4.48) we are seeing a plot of $\beta_1(C,32,32)$ versus C, the traffic in records per second.

In the lower left-hand corner of the plot we see that under low traffic conditions the access time for level 1 corresponds to access time of the device 80 ns. This is of course a consequence of virtually no queueing. As we move to the right increasing traffic is seen as an increase in the mean waiting time due to congestion. At the far right near 12.5 million record/sec. the mean waiting time increases to 236 ns. It is important to note that 12.5 million records/sec. is the maximum possible service rate for an 80 ns server under these conditions. Also note that if we always have 3 requests for service in the system the mean waiting time will be 3×80 ns or 240 ns. The maximum mean waiting time shown here is 236 ns, which is appropriate for a traffic figure slightly under the maximum possible. It is useful to note that in generating these plots the program modeling the β function is called with increasing values of C or traffic, until it responds with a special return which indicates that the device being modeled cannot operate at such a high traffic.

Figure 5.5 gives us the mean queue length as a function of the traffic at level 1 under the same circumstances. Here we see the mean queue length (which includes the request in service) go from near 0 to 2.95 or just under 3, as would be anticipated. The values of service rate or traffic using the relation [33]:

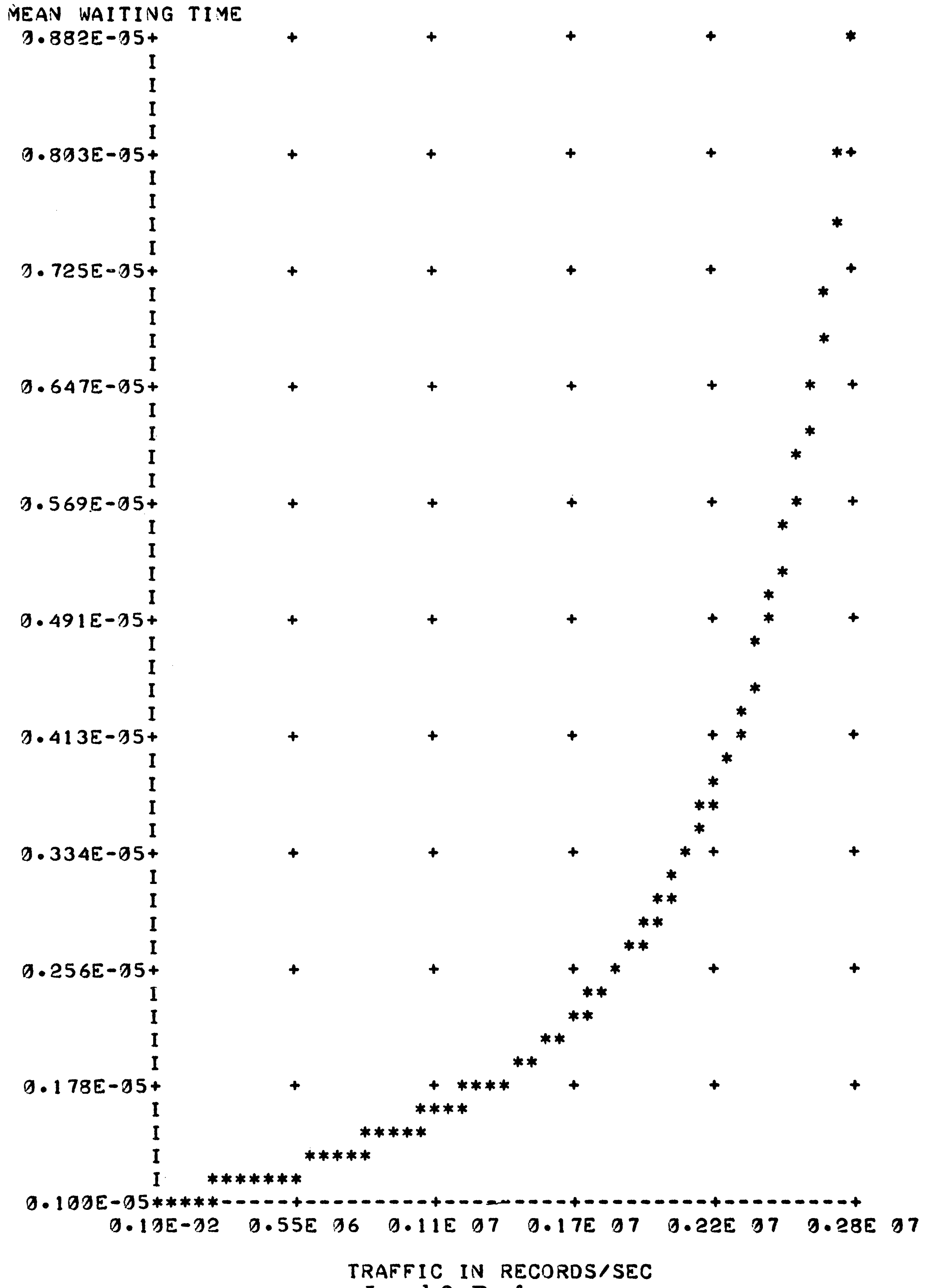
Mean number in queue and service = mean service rate $\times$ mean waiting time.

We will now turn to the next level of storage, level 2. The description of the level 2 configuration is in Figure 5.6. At level 2 we are using the same model that was used at level 1. Here, however, the

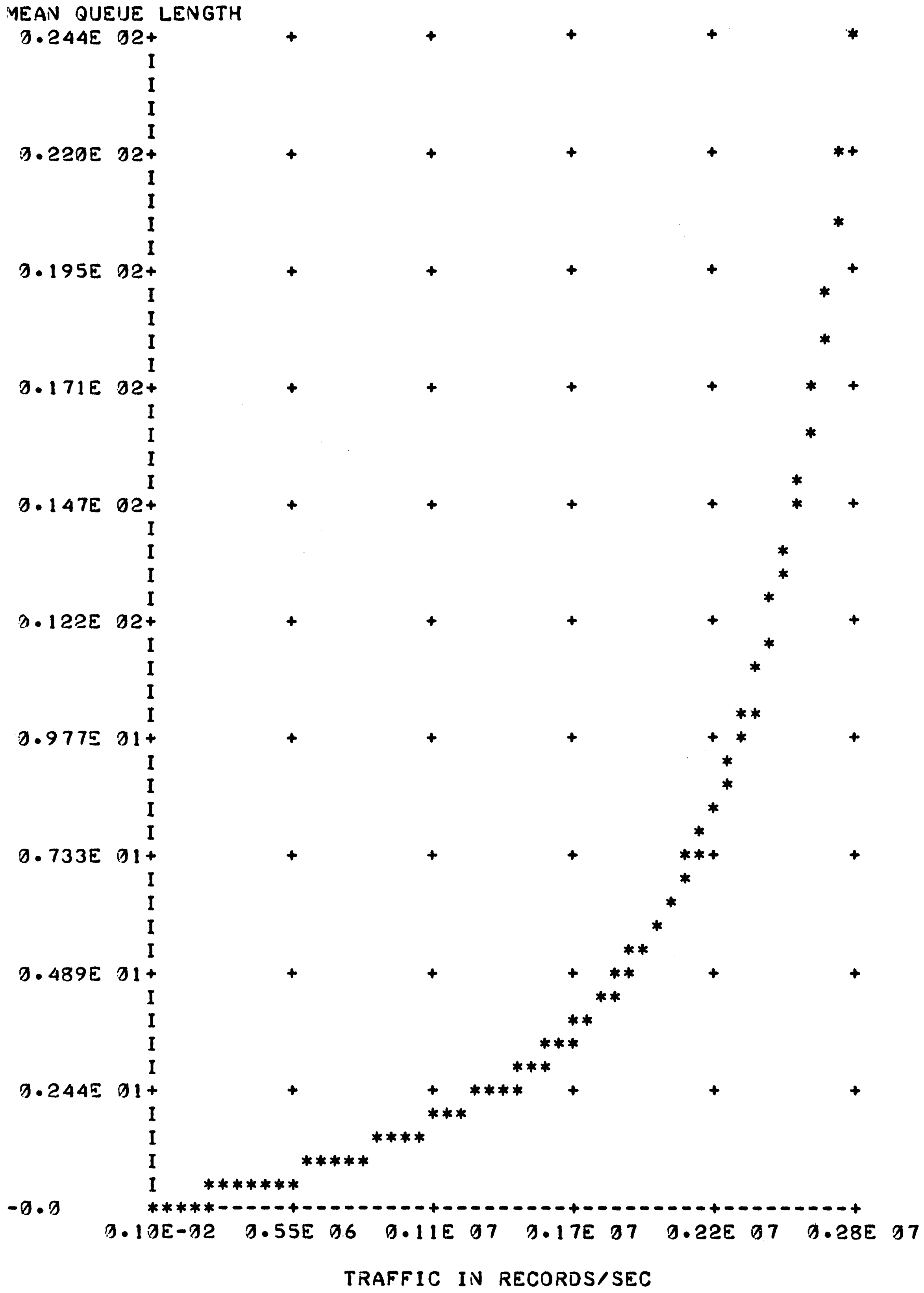
```
*** LEVEL 2 ***

MODEL TYPE  2

MODEL 2  (RANDOM ACCESS OR CORE UNITS)

AVERAGE ACCESS TIME FOR A SINGLE UNIT    0.9999967E-06

BASIC DEVICE RECORD SIZE                          128.

UNIT CAPACITY                            0.5000000E 06

NUMBER OF UNITS                            3

MIN # OF UNITS                             2

MAX # OF UNITS                             8

COST PER UNIT ($)                             50000.00
```

Level 2 Hardware Description

Figure 5.6



Level 2 Performance
Figure 5.7



Level 2 Queuing
Figure 5.8

parameters are different. There is very little new here except possibly the range of allowable number of core units and the basic device record size. Thus we have 3 core units, each with a capacity of 1/2 million bits and each capable of accessing a 128 bit word in 1 μs.

The performance of level 2 under load is shown in Figure 5.7. Here as for level 1 we have plotted the mean waiting time as a function of traffic in records per second. The record size used here was the basic device record size of 128 bits and the maximum number in queue and service was set to 26. (The motivation for the choice of the values for the maximum number in queue and service will become obvious later.) The maximum service rate would be 3 million records/sec. if all 3 core units were always busy. Of course we cannot expect all 3 core units to remain busy at every instant. But, with a maximum of 26 requests for service in the system we can expect the maximum feasible traffic to be near 3 million records per second as indicated in Figure 5.7. To get a rough estimate of the maximum mean waiting time we can assume that the requests for service divide evenly among the 3 core units, thus giving us a maximum of $1/3 \times 26 = 8.666$ in each queue. If there were in fact 8.666 in each queue the mean waiting time would be 8.666 μs. Notice that the maximum given in Figure 5.7 is a little larger than 8.666 μs. This should be expected since an occasionally idle core unit and the resulting long queues at the other units would be expected to increase the mean waiting time.

Figure 5.8 shows us the mean queue length as a function of traffic, and this varies as expected from near 0 to just under the maximum of 26.

Turning now to level 3 described in Figure 5.9 we see that the situation has changed somewhat. At level 3 we use model type 3 which is the drum model described in Section 4.7. The device parameters indicate a 1900 RPM drum with 100 K bits per track and a capacity of 10 million bits. There is currently one drum and we may attach from 1 to 4. Again the cost per unit is $50,000.

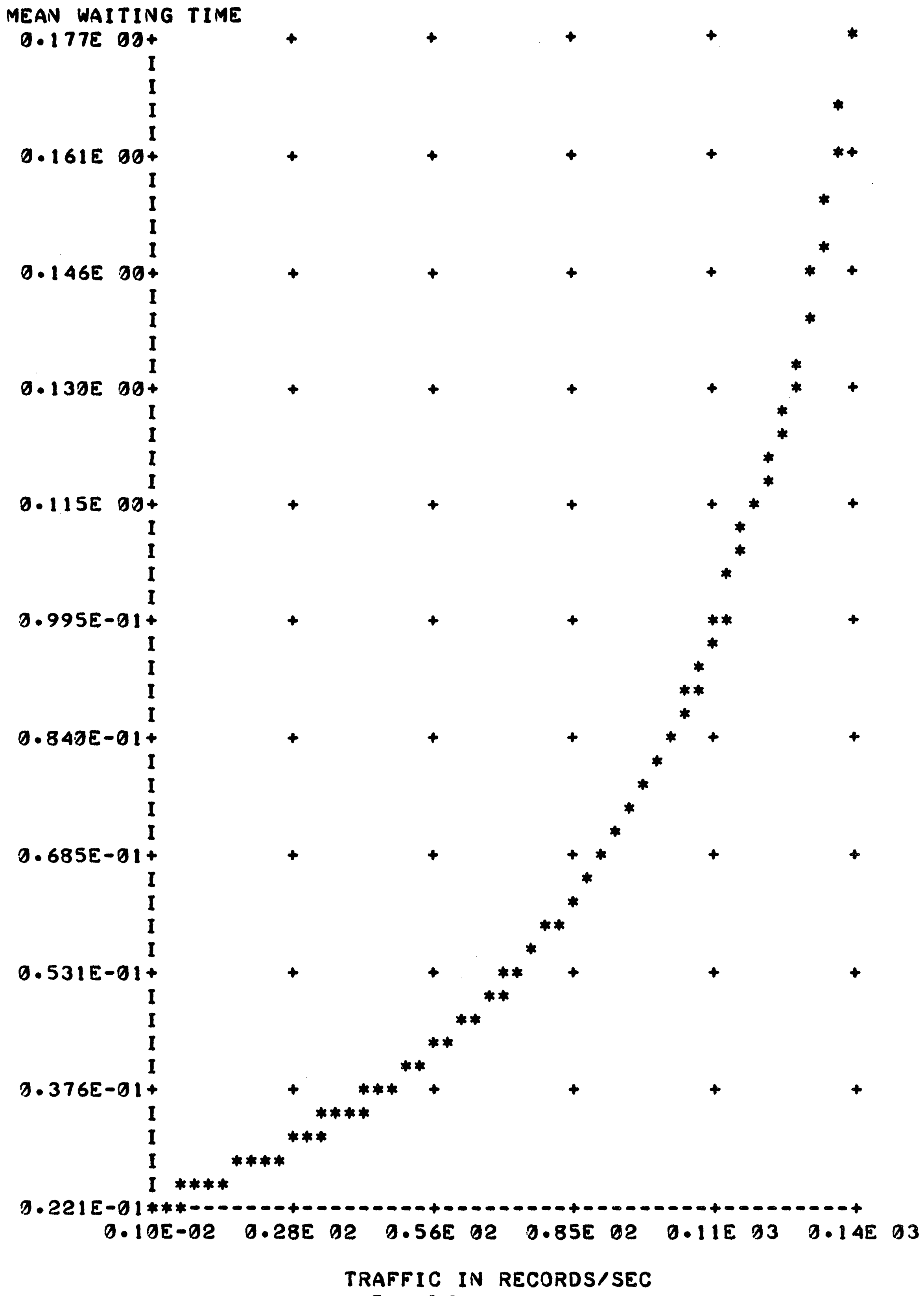
The performance is shown in Figure 5.10. Here we have assumed a maximum of 26 requests in the system and a record size of 16384. Notice that at the low traffic end we have a mean waiting time of 22.1 ms. Using a record size of 16384 bits we have 6 sectors on the 100 K bit per track drum. Rotating at 1800 RPM we have 16.6 ms of latency (1/2 a rotation) and $1/3 \times 16.6 = 5.5$ ms for data transfer. Thus latency and transfer time is 22.1 and agrees with the low traffic waiting time shown in Figure 5.10.

The maximum traffic is 6 records per revolution. Since a complete revolution requires 33.3 ms the maximum traffic will be 180 records/sec. Since we have a maximum of 26 requests for service in the system at level 3 and 6 sectors on the drum we see that this is only roughly 4.3 requests per sector. This means that it is quite likely that a sector may pass under the read/write heads with no request in queue

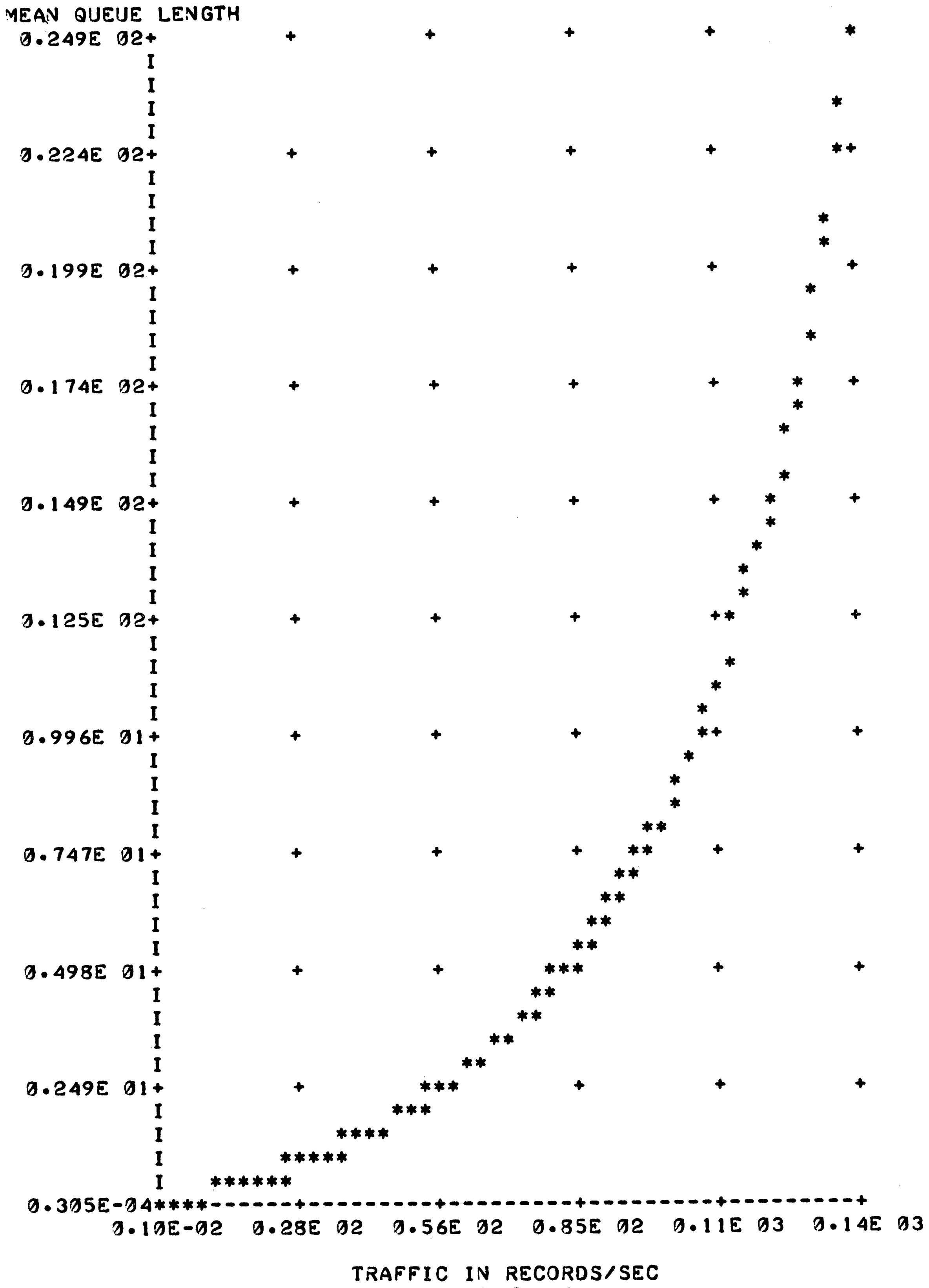
```
*** LEVEL 3 ***

MODEL TYPE  3

MODEL 3  (DRUM OR FIXED HEAD DISK)

DEVICE RPM                                       0.1800000E 04

BITS PER TRACK                                   0.1000000E 06

UNIT CAPACITY                                    0.1000000E 08

NUMBER OF UNITS                                    1

MIN # OF UNITS                                     1

MAX # OF UNITS                                     4

COST PER UNIT ($)                                       50000.00
```

Level 3 Hardware Description

Figure 5.9



Level 3 Performance
Figure 5.10



Level 3 Queuing
Figure 5.11

to be serviced. Thus we see that the maximum indicated here of approximately 140 records per second is expected. A very rough estimate on the maximum mean waiting time can be calculated by simply assuming 4.3 requests in service at each sector and a full rotation or 33.3 ms required for each read. This gives us an average mean waiting time of 143 ms. The maximum shown in Figure 5.10 is 177 ms, clearly larger than our estimate due to the random queueing and idle periods.

Figure 5.11 shows the mean queue length, which as expected runs from near 0 to just under 26, the maximum.

Traffic

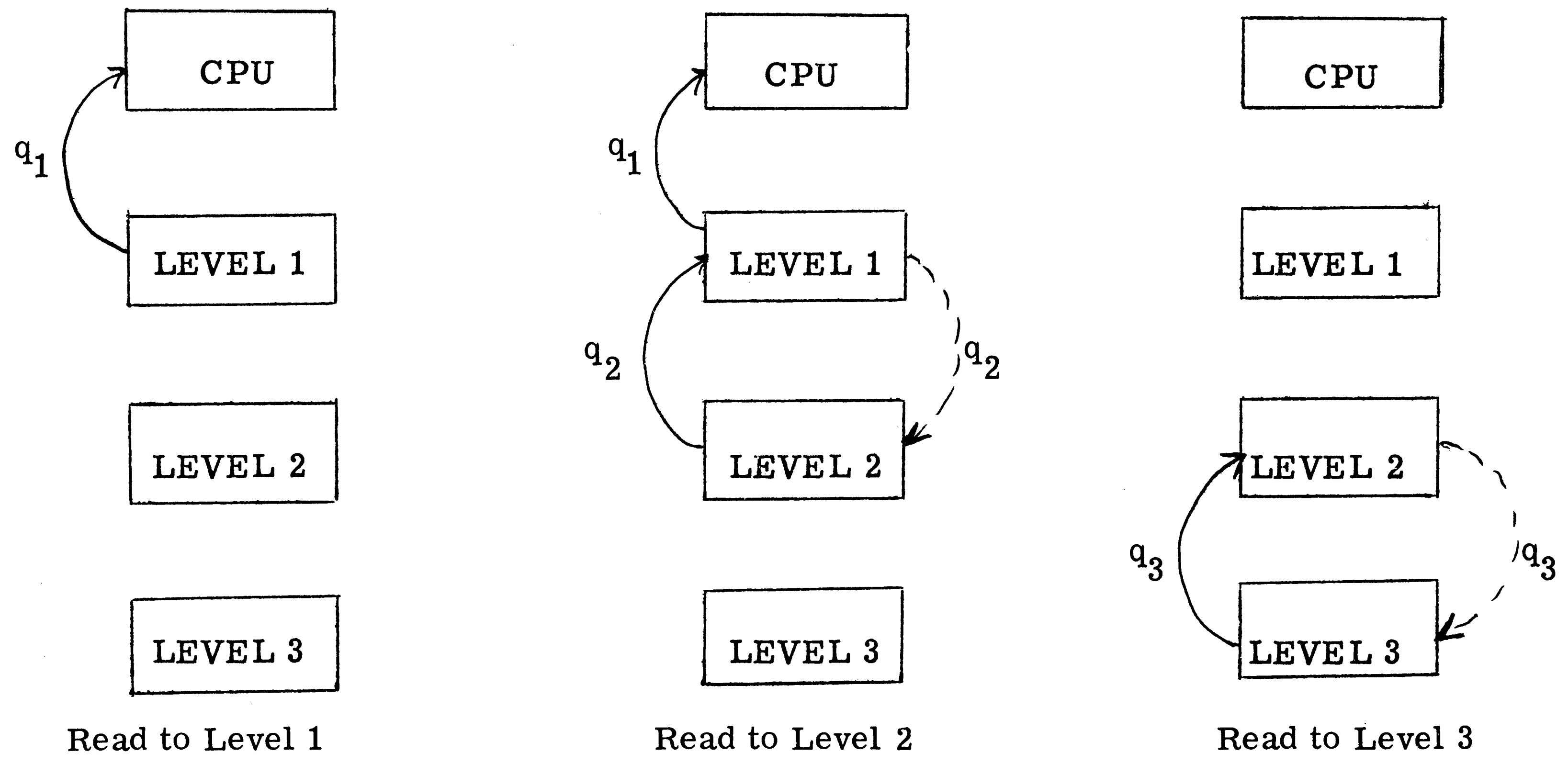
We will now examine the traffic patterns in the system and the critical reads and writes. We will begin with the diagrammed transfers in Figure 5.12.

In Figure 5.12 the data transfers which result from a user program read to each of the 3 levels of storage are shown. The solid lines indicate the transfers moving data up in the hierarchy. The dashed lines indicate the transfers required to page out or those transfers required to make room for the data being paged up.

The q_i indicates the size of the transfer in terms of the record sizes at the three levels, in this case

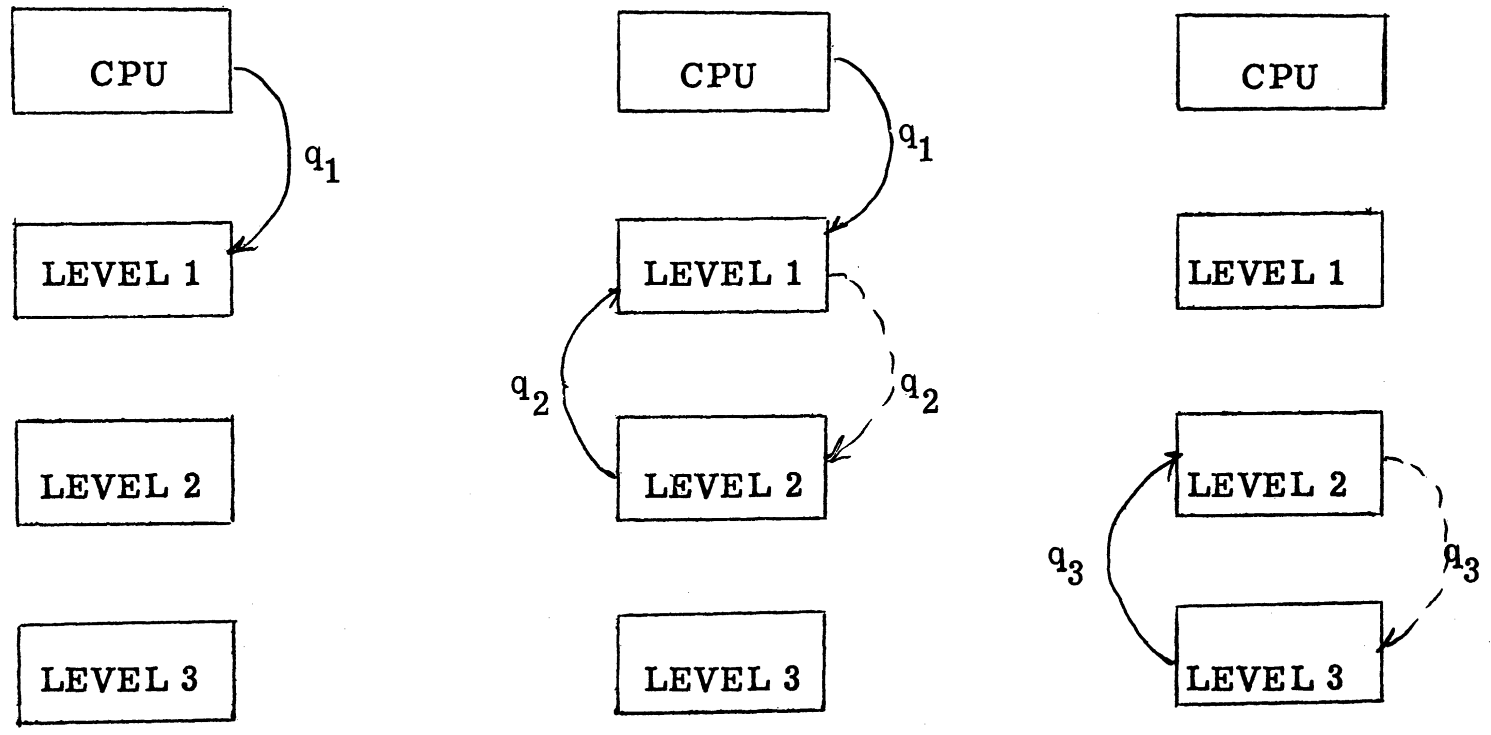
$$q_1 = 32 \text{ bits},$$

$$q_2 = 1024 \text{ bits}$$



Traffic Patterns Resulting From User Program Reads

Figure 5.12



Write to Level 1 Write to Level 2 Write to Level 3

Traffic Patterns Resulting From User Program Writes

Figure 5.13

and $q_3 = 16384$ bits

as indicated in the table of record sizes given in Figure 5. 2.

On the left of Figure 5.12 w see that a user program read to level 1 causes a record of size q_1 or 32 bits to be read from level 1 into the CPU. In the center we see the effect of level 2 or core access. In this case 1024 bits are brought from core (level 2) into the cache (level 1) followed by a transfer of 32 bits from the cache to the CPU. The downward transfer from the cache to the core of 1024 bits is necessary to maintain space in the cache for the upward paged information.

The effect of a user program read to level 3, a secondary access,is shown on the right. Here we see an upward transfer of 1024 from drum (level 3) to core (level 2) and a corresponding downward transfer of the same size.

Figure 5.13 shows the effect of user program writes. The transfers performed here are almost identical to those for reads, differing only in the direction of transfer between core and cache.

The transfers shown in Figure 5.12 and 5.13 are represented in the model in the form of the "T" array shown in Figure 5.14 and 5.15. Let us examine carefully the center matrix in Figure 5.14. Here we have a description of the traffic which occurs as a result of a user program read to level 2. The elements of this matrix are derived from the traffic pattern shown in the center figure of Figure 5.12.

```
      ***  THE "T" ARRAY  ***

     USER PROGRAM READ TO LEVEL 1

  TRAFFIC       RECORD SIZE
  AT        Q(1)   Q(2)   Q(3)
  LEVEL
   1        1.00   0.0    0.0


   2        0.0    0.0    0.0


   3        0.0    0.0    0.0




     USER PROGRAM READ TO LEVEL 2

  TRAFFIC       RECORD SIZE
  AT        Q(1)   Q(2)   Q(3)
  LEVEL
   1        1.00   2.00   0.0


   2        0.0    2.00   0.0


   3        0.0    0.0    0.0




     USER PROGRAM READ TO LEVEL 3

  TRAFFIC       RECORD SIZE
  AT        Q(1)   Q(2)   Q(3)
  LEVEL
   1        0.0    0.0    0.0


   2        0.0    0.0    2.00


   3        0.0    0.0    2.00
```

The Traffic Array for User Program Reads

Figure 5.14

```
   USER PROGRAM WRITE TO LEVEL 1

TRAFFIC      RECORD SIZE
AT        Q(1)  Q(2)  Q(3)
LEVEL
 1        1.ØØ  Ø.Ø   Ø.Ø


 2        Ø.Ø   Ø.Ø   Ø.Ø


 3        Ø.Ø   Ø.Ø   Ø.Ø
```

```
   USER PROGRAM WRITE TO LEVEL 2

TRAFFIC      RECORD SIZE
AT        Q(1)  Q(2)  Q(3)
LEVEL
 1        1.ØØ  2.ØØ  Ø.Ø


 2        Ø.Ø   2.ØØ  Ø.Ø


 3        Ø.Ø   Ø.Ø   Ø.Ø
```

```
   USER PROGRAM WRITE TO LEVEL 3

TRAFFIC      RECORD SIZE
AT        Q(1)  Q(2)  Q(3)
LEVEL
 1        Ø.Ø   Ø.Ø   Ø.Ø


 2        Ø.Ø   Ø.Ø   2.ØØ


 3        Ø.Ø   Ø.Ø   2.ØØ
```

The Traffic Array for User Program Writes

Figure 5.15

Notice that 1 record of size q_1 is read, 1 record of size q_2 is read and 1 record of q_2 is written at level 1. Summing up, we have 1 record of size q_1 and 2 records of size q_2 read or written at level 1 as a result of a user program read to level 2. In the center matrix of Figure 5.14 this traffic at level 1 is indicated in row 1 of the matrix. The rows correspond to levels and the columns correspond to record sizes.

Following the same lines we see that 2 records of size q_2 are read or written at level 2. This results in an entry of 2 at row 2, column 2 of the matrix. There is no traffic generated at level 3 and correspondingly zero entries in row 3 of the matrix.

Each of the 6 matrices shown in Figure 5.14 and 5.15 can be derived from its corresponding traffic pattern diagram in a similar manner. This will be left up to the reader's interest and energy.

We will now turn to the critical reads and writes shown in the "G" array in Figure 5.16 and 5.17. Here we specify exactly which of the reads and writes generated by a user program access must be completed in order to complete that access. We will again focus on the user program read to level 2, which corresponds to the center matrix of Figure 5.16. This matrix specifies that the time required to complete a user program access to level 2 is the time required to read a q_1 or 32 bit record at level 2 plus the time required to read a q_1 or 32 bit record at level 1. This means that we are assuming that in the transfer from core to cache the write time at the cache is negligible or

```
          ***  THE "G" ARRAY  ***

        USER PROGRAM READ TO LEVEL 1

      TRAFFIC      RECORD SIZE
      AT       Q(1)  Q(2)  Q(3)
      LEVEL
       1       1.00  0.0   0.0


       2       0.0   0.0   0.0


       3       0.0   0.0   0.0




        USER PROGRAM READ TO LEVEL 2

      TRAFFIC      RECORD SIZE
      AT       Q(1)  Q(2)  Q(3)
      LEVEL
       1       1.00  0.0   0.0


       2       1.00  0.0   0.0


       3       0.0   0.0   0.0




        USER PROGRAM READ TO LEVEL 3

      TRAFFIC      RECORD SIZE
      AT       Q(1)  Q(2)  Q(3)
      LEVEL
       1       0.0   0.0   0.0


       2       0.0   0.0   0.0


       3       0.0   0.0   1.00
```

The Critical Read/Write Array for User Program Reads

Figure 5.16

```
   USER PROGRAM WRITE TO LEVEL 1

TRAFFIC       RECORD SIZE
AT        Q(1)   Q(2)   Q(3)
LEVEL
 1        1.00   0.0    0.0


 2        0.0    0.0    0.0


 3        0.0    0.0    0.0




   USER PROGRAM WRITE TO LEVEL 2

TRAFFIC       RECORD SIZE
AT        Q(1)   Q(2)   Q(3)
LEVEL
 1        1.00   0.0    0.0


 2        1.00   0.0    0.0


 3        0.0    0.0    0.0




   USER PROGRAM WRITE TO LEVEL 3

TRAFFIC       RECORD SIZE
AT        Q(1)   Q(2)   Q(3)
LEVEL
 1        0.0    0.0    0.0


 2        0.0    0.0    0.0


 3        0.0    0.0    1.00
```

The Critical Read/Write Array for User Program Writes

Figure 5.17

concurrent and that we may access the required 32 bit word from the cache as soon as the first 32 bits reaches the cache from core. We have also assumed that the page down operation from the cache is carried out with sufficient anticipation to avoid delays while down transfers are made.

The critical reads and writes and their associated assumptions for level 2 are far more complex than either of the remaining 2 levels. In the case of level 1 read, it is simply the time required to read a q_1 or 32 bit record from level 1. In the case of a level 3 read, it is simply the time required to read a q_3 or 16384 bit record from the drum. It is again assumed here that neither writing in core not paging out at the core will cause delay.

The User Program

Five of the parameters which specify user program behavior are given in Figure 5.18. These parameters are independent variables in the program model developed in Chapter 2.

Notice that the values given here are identical to the values used in the "standard" case discussed in Section 2.7. The CPU word size here, 32 bits, also agrees with the "standard" case in Chapter 2. Continuing, the logical record or page size in the "standard" case is 16384 which is the same as the record size at the drum or level 3 in this example. Thus when we examine the relationship between primary and secondary levels in this system as a function of core allocation, we

```
***  USER PROGRAM SPECIFICATIONS  ***

PROGRAM SIZE IN BITS                                 163840

CYCLE REPETITION NUMBER                            20.00000

MODE OF THE CYCLE LENGTH                            0.50000

DELTA DISTRIBUTION PARAMETER                       20.00000

ACCESSES BETWEEN NON-STORAGE INTERRUPTS                4000
```

User Program Specifications

Figure 5.18

```
TOTAL # OF ACCESSES/
     ACCESSES BELOW LEVEL 2
 0.351E 04+         +         +         +         +        **
          I                                             ***
          I                                           ***
          I                                         ***
          I                                      ***
 0.316E 04+         +         +         +      ***+         +
          I                                  **
          I                                 **
          I                                *
          I                               *
 0.281E 04+         +         +         +*        +         +
          I                              *
          I                             *
          I                             *
          I                            *
 0.246E 04+         +         +        *+         +         +
          I
          I                           *
          I
          I                           *
 0.211E 04+         +         +         +         +         +
          I                          *
          I                          *
          I
          I                         *
 0.176E 04+         +         +         +         +         +
          I                         *
          I
          I                        *
          I
 0.141E 04+         +         +    *    +         +         +
          I                       *
          I                       *
          I
          I                      *
 0.105E 04+         +         +  *      +         +         +
          I                     *
          I                    *
          I                   **
          I                  **
 0.704E 03+         +       **+         +         +         +
          I                **
          I              **
          I           ***
          I      ******
 0.352E 03+     *   +         +         +         +         +
          I
          I
          I
          I
 0.100E 01******----+---------+---------+---------+---------+
      -0.0       0.31E 05  0.62E 05  0.93E 05  0.12E 06  0.16E 06

              ALLOCATION AT LEVEL 2 IN BITS
```
User Program Behavior
Figure 5.19

will get the same results as given in Section 2.7 for the standard case.

The plot shown in Figure 5.19 shows the relation between the total number of accesses and the accesses below level 2 as a function of allocation at level 2. This is the same as the curve shown in Figure 2.11 with obvious differences in scaling and plotting method.

There is only 1 remaining independent variable to be discussed. This is the mean time required to complete an interrupt external to the storage system, such as a terminal user. The value used in this case will be 60 ms. Thus on the average a user program will require some kind of interaction with devices outside the CPU and storage system after each 4000 accesses are completed, and that interaction will cause loss of the CPU and an average wait on the external divide of 60 ms.

System Performance

We have at this point succeeded in wading through the independent variables in the system and we are now in a position to examine the performance of the system.

The next 3 figures, 5.20, 5.21, and 5.22, contain the dependent variables which are generated by analysis. These 3 figures are important because we will see them repeated a number of times in the near future. Each repetition will of course contain different numbers resulting from some system modification, but the format and meaning of the numbers will remain fixed for ease of comparison. Because of the repeated appearance of these dependent variables we will examine them very carefully here to avoid the necessity for repeated explanation.

Beginning at the top of Figure 5.20, Part 1 of the dependent variables we see the mean CPU access rate. This is the rate at which the collection of M user programs makes access or reference to individual words of storage. This is the measure of work done or throughput and corresponds to the variable r defined in Equation 3.14.

Next in Figure 5.20 we find the allocation data. The first line here indicates the number and size of the programs being multiprogrammed (independent variables). This is followed by a table giving the total space and allocation at each level. The total space is the total capacity in bits or simply the product of the number of units at a given level and the unit capacity. The allocation is the space allocated to an executing program. At level 1, a dedicated level, the allocated space is equal to the total space. At levels 2 and 3, shared levels, the allocated space is $1/13 \times$ (Total Space) as the total space is allocated uniformly to the 13 programs being multiprogrammed. The last column on the right headed %OF PROGRAM contains the percentage of the program (163840 bits) which will fit into the allocated space. Thus 5% of the program will fit into the cache, 70% of the program will fit into the core units and the available space at the drum exceeds the space requirements of the running programs by a factor of 4.69.

```
MEAN CPU ACCESS RATE = 0.64961150E 06

 ***  ALLOCATION  DATA  ***

  13 PROGRAMS WITH EACH USING  0.16384000E 06 BITS

LEVEL     TOTAL SPACE          ALLOCATION     % OF PROGRAM

  1      0.81920000E 04    0.81920000E 04      5.0000

  2      0.15000000E 07    0.11538456E 06     70.4251

  3      0.10000000E 08    0.76923075E 06    469.5010



CPU/PRIMARY EFFICIENCY= 28.845%


        CPU UTILIZATION

USER        22.521%

SYSTEM       3.106%

WAIT        74.373%

 ***  MEAN  TRAFFIC  ***

--OVER ALL TIME--

CPU ACCESS RATE =  0.64961150E 06
LEVEL    % USER ACCESS    READ/WRITE RATE      RECORD SIZE

  1        96.9504089355    0.68861075E 06    0.88480591E 02

  2         3.0177125931    0.39296164E 05    0.10589312E 04

  3         0.0068785138    0.89367203E 02    0.16383992E 05

--OVER PERIODS OF USER PROGRAM EXECUTION--

CPU ACCESS RATE =  0.28844640E 07
LEVEL    % USER ACCESS    READ/WRITE RATE      RECORD SIZE

  1        96.9504089355    0.30576320E 07    0.88480576E 02

  2         3.0177125931    0.17417887E 06    0.10318809E 04
```

Dependent Variables, Part 1, "Standard" Case

Figure 5.20

```
 *** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
     WHEN RUNNING  13 USER(S).

LEVEL  MAX REQUESTS
  1          3
  2         26
  3         26


 ***  ACCESS  TIMES  ***

--SECONDARY ACCESS ENVIRONMENT--

       MEAN             MEAN TOTAL      MEAN QUEUE   MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE     SERVICE TIME    LENGTH       SERVICE TIME     EFFECT %

  2     0.39296E 05      0.92783E-05      0.36460    0.0                0.0

  3     0.89367E 02      0.67129E-01      5.99913    0.14485E-01       23.5376

 EXT    0.16240E 03      0.60000E-01     N.A.        0.47054E-01       76.4624

--PRIMARY ACCESS ENVIRONMENT--

       MEAN             MEAN TOTAL      MEAN QUEUE   MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE     SERVICE TIME    LENGTH       SERVICE TIME     EFFECT %

  1     0.30576E 07      0.41495E-06      1.26876    0.27375E-06       41.4617

  2     0.17418E 06      0.15155E-04      2.63963    0.38642E-06       58.5265
```

Dependent Variables, Part 2, "Standard" Case

Figure 5.21

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  0.6602414E-06

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  0.6153823E-01

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  0.1087519E-02

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  0.3466848E-06
```

Dependent Variables, Part 3, "Standard" Case

Figure 5.22

The cost of storage hardware is simply a summation of the costs associated with each of the storage devices used. In this case we have 1 cache, 3 cores and 1 drum or 5 devices, each assigned a cost of $50,000 or a total of $250,000.

Continuing we find several efficiency and utilization figures. First there is the CPU/PRIMARY EFFICIENCY given as 28.845%. This figure is a measure of CPU efficiency when it is infact executing a user program. This simply says that when a user program is executing, it is executing at 28.8% of the rate that it would execute if the user program were located entirely in the cache. In terms of the model variables this figure is simply

$$(r'/\hat{r}) \times 100$$

where r' is the CPU access rate during user program execution. (r' defined Equation 3.15)

The CPU utilization is given in the form of 3 percentage figures - user, system and wait. The user figure is the percentage of time that the CPU spends executing a user program, 22.5%. The system figure is the percentage of time the CPU spends doing the bookkeeping tasks associated with user program interchanges. Finally the percentage of time the CPU spends in an idle or wait state is given.

The remaining data at the bottom of Figure 5.20 gives us the access behavior and traffic environment in the system. We begin with the mean traffic based on an average over all time. We see first the CPU access rate. This is simply a repeat of the mean CPU access rate given at the top of the Figure.

Continuing we have a table giving data for levels 1 through 3. On the left we have the %USER ACCESS. This column gives us the percentage of all user program accesses which are made to each of the 3 levels. The second column headed READ/WRITE RATE gives us the rate at which records are being read and written at each level. The final column displays the mean record size for each level. In terms of the model the data shown here comes directly from the u_i, C_i, A_i defined in Equations 3.70-3.72, 3.19 and 3.21 respectively.

The final and remaining table is the same as the one we have just considered except here we are looking at the system only over periods of user program execution. The CPU access rate given here is r' the rate taken during periods when CPU is busy. The read/write rate and the mean record size are likewise taken over periods when the CPU is busy. The percent of user access given in this table is the same as in the previous table. The read/write rate is simply C'_i, defined in Equation 3.29 and the record size is A'_i, defined in Equation 3.31.

Turning now to Figure 5.21 we have a group of dependent variables which are closely related to the performance at individual levels. At the top of Figure 5.21 we have a table expressing the maximum number of requests for service which can occur at each of the three levels. The values given here are the same as those used earler when examining the performance characteristics of the individual levels. The values in the table are the B_j's expressed in Equation 4.3 where j indicates the level.

In the lower part of Figure 5.21 we see 2 tables, one labled secondary access environment and another labled primary access environment. In the first of these tables (i.e. describing the secondary access environment) we display information about the traffic at the shared levels of storage as seen by a user program making a secondary access. That is, the averages are taken over all time. In addition to the shared levels, levels 2 and 3, data describing the external interrupts are given.

Under the heading MEAN SERVICE RATE we have the rate (over all time) at which records are being read or written at each level. In the case of external interrupts this is the rate at which these external transactions are being carried out. Notice that this column is identical to the column headed READ/WRITE RATE in Figure 5.20 except the dedicated levels are omitted and external behavior is added.

In the next column we give the mean TOTAL SERVICE TIME (read/write) for a record of the mean record size under mean traffic conditions. This is an after-the-fact calculation based on mean values and thus may not relate directly to the system performance, but it provides a measure of the general congestion and delay at a given level. Based on this mean total service time and the mean service rate we compute a queue length which is found in the next column. It should always be kept in mind that the mean total service time given here and the mean queue length are based on the average record size and average service rate. In some cases this presents a problem and in others not. Note that for level 3 in this system all records read or written are of size 16384 bits and thus the values given have a precise meaning.

Turning now to the MEAN CRITICAL SERVICE TIME, in this column we see the components of the sum which make up the mean secondary access time $\bar{\tau}_s$. The critical service time at level 2 is 0 because delay at level 2 is not considered part of the time required to complete a secondary access. This is of course a function of the "G" array. Notice also that the external interrupts are considered part of the secondary access behavior. The final column simply gives the percentage of each of the components which make up $\bar{\tau}_s$. In this particular system on the average programs ineligible for CPU use spend 23.5% of the time (before becoming eligible for execution) waiting on a drum transfer and 76.4% of the time waiting on some external action to be completed.

The table of data given for the primary access environment is the same with a few exceptions. Here we show only the primary levels and base our calculations on the mean traffic and mean record size taken over user program execution time only. The mean critical service time here consists of the components of the sum which makes up the mean primary access time $\bar{\tau}_p$.

Arriving now at the final of the 3 displays of dependent variables Figure 5.22. This is by far the simplest figure and displays $\bar{\tau}_p$, $\bar{\tau}_s$, $\bar{\tau}_e$ and $\bar{\tau}_c$ from chapter 4. These are given here in the form TAUP, TAUS, TAUE and TAUC respectively.

Having spent a considerable period of time examining the individual numbers presented we will now discuss for a moment what these numbers are telling us about the system. First of all the CPU is executing near 28.8% of its specified capability. This low CPU efficiency was a surprise and the cause was not immediately apparant. However, on more careful examination the deficiency becomes clear.

Notice that the allocated space at the cache is 5% of the program size and that the most likely loop length is half the length of the program ($p = .5$). Thus the cache is operating in a severely under-allocated manner. That is, words brought to the cache are rarely used more than once. This means that in accordance with our defined traffic patterns, each word read or written by the CPU must be brought from core to cache, then read or written while in the cache and then later transferred to core. Most important in all this is that each word accessed by the CPU requires 3 accesses at the cache;

1 to bring it into the cache

1 to remove it from the cache

1 CPU read or write to the cache.

Since as we discussed earlier the cache has a maximum read/write rate for 32 bit records of 12.5 million per second the CPU will be limited to 1/3 of that or 4.17 million accesses per second or 41.7% of the maximum CPU rate $\hat{r}$. Thus we clearly have a bandwidth problem at the cache. It is most interesting to note that in the IBM 360/85 which uses a similar cache. structure, the data flow is such as to avoid traffic at the cache. If we were operating a 360/85 under roughly the same conditions, a read would generate 2 cache operations (one on the core to cache transfer and a second on the cache to CPU transfer). We are ignoring here the first word of a block which need not be read from the cache but is transferred directly from core to the CPU. In the IBM 360/85 [16,37] no downward transfer is required when the data is not modified. In the case of writes the cache is avoided entirely and the words are written directly in core with sufficient buffering to avoid delay. When a write is made to a word already in the cache, that word is modified in the cache but it is also modified directly in core at the same time in order to avoid a later transfer from cache to core adding only one write to the traffic at the cache.

Examining the effect of core delays on the CPU rate we see that the core is having considerable effect. (See the relative effect for primary access environment Figure 5.21.) A Solution here would be to use more core units and a larger record size: more core units to increase the bandwidth at core and a larger record size to incur the delay at core less frequently. Following along the same lines as we did for the cache, one finds that current core bandwidth would limit the CPU access rate to under 6 million per second.

It is interesting to note that the traffic patterns in the IBM 360/85 reduces the traffic at both the cache and the core. Experience with this model indicates that the effort to eliminate unnecessary traffic is important.

Thus far we have concentrated on the efficiency of the CPU when it is executing a user program. This, although very interesting, is not the limiting factor in the performance of the system. From the CPU utilization figures we see that even at its reduced execution rate, the CPU is idle 74.3% of the time. This means that the accesses to the drum and/or the interaction with devices external to the storage system are limiting the performance. In Figure 5.21 the relative effects of these two activities are shown and we can see that neither is negligible. We will consider several changes in the system in the next few sections which will give us some insight into the behavior of the system and expecially the relationship between the primary and secondary levels.

As we progress we will want to compare and contrast the performance of other system configurations with the one we have just examined. We will refer to the system studied here as the "standard" system when making comparisons.

5.3 A Hardware Modification

We are at liberty to modify the system in many ways. We will begin with the simple hardware modification of replacing the current drum at level 3 with a drum indentical to its predecessor in every respect except rotational rate. In the "standard" case just considered, the drum RPM was 1800. We will now double that to 3600 RPM and observe its effect on the system. The results of an analysis for this new system are shown in Figures 5.23, 5.24, and 5.25. The system now being considered is identical in every respect to the "standard" case except for the drum RPM.

Comparing the results of this analysis with that of the "standard" case we see an overall increase in mean CPU access rate from 649 thousand accesses per second to 772 thousand. The CPU utilization has increased by several percentage points. Notice that the CPU/PRIMARY EFFICIENCY has dropped slightly. This is due to the increased core congestion resulting from the increased drum bandwidth.

In Figure 5.24 we see a mean queue length at the drum of 2.2, a drop from 6 in the "standard" case. Notice that the external interrupts now constitute over 90% of the secondary delay. The system could be described as IØ bound.

Notice that access time at level 3 is now less than 1/2 of its previous value. This is a result of both the increase in service rate and the decrease in queuing.

```
MEAN CPU ACCESS RATE = 0.77293700E 06

 ***  ALLOCATION  DATA  ***

  13 PROGRAMS WITH EACH USING  0.1638400E 06 BITS

LEVEL     TOTAL SPACE          ALLOCATION      % OF PROGRAM

  1     0.81920000E 04    0.81920000E 04       5.0000

  2     0.15000000E 07    0.11538456E 06      70.4251

  3     0.10000000E 08    0.76923075E 06     469.5010


 CPU/PRIMARY EFFICIENCY= 28.819%


        CPU UTILIZATION

 USER        26.820%

 SYSTEM       3.696%

 WAIT        69.484%

  ***  MEAN  TRAFFIC  ***

 --OVER ALL TIME--

 CPU ACCESS RATE =  0.77293700E 06
 LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

   1       96.9504089355    0.81934006E 06    0.88480591E 02

   2        3.0177125931    0.46756344E 05    0.10589312E 04

   3        0.0068785138    0.10633311E 03    0.16383992E 05

 --OVER PERIODS OF USER PROGRAM EXECUTION--

 CPU ACCESS RATE =  0.28819230E 07
 LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

   1       96.9504089355    0.30549380E 07    0.88480591E 02

   2        3.0177125931    0.17404250E 06    0.10333845E 04
```

Dependent Variables, Part 1, for Drum RPM = 3600

Figure 5.23

```
*** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
    WHEN RUNNING  13 USER(S).

 LEVEL  MAX REQUESTS
   1        3
   2       26
   3       26


 ***  ACCESS  TIMES  ***

 --SECONDARY ACCESS ENVIRONMENT--

        MEAN             MEAN TOTAL     MEAN QUEUE   MEAN CRITICAL   RELATIVE
 LEVEL  SERVICE RATE     SERVICE TIME   LENGTH       SERVICE TIME    EFFECT %

   2     0.46756E 05     0.94975E-05      0.44407    0.0                0.0

   3     0.10633E 03     0.20705E-01      2.20160    0.44675E-02        8.6713

  EXT    0.19323E 03     0.60000E-01    N.A.         0.47054E-01       91.3287
%GORDON ROBERTS *** PLEASE CALL 763-0197 ***

 --PRIMARY ACCESS ENVIRONMENT--

        MEAN             MEAN TOTAL     MEAN QUEUE   MEAN CRITICAL   RELATIVE
 LEVEL  SERVICE RATE     SERVICE TIME   LENGTH       SERVICE TIME    EFFECT %

   1     0.30549E 07     0.41474E-06      1.26701    0.27354E-06       41.3919

   2     0.17404E 06     0.15186E-04      2.64297    0.38724E-06       58.5964
```

Dependent Variables, Part 2, for Drum RPM = 3600

Figure 5.24

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  0.6608528E-06

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  0.5152114E-01

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  0.1088478E-02

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  0.3469905E-06
```

Dependent Variables, Part 3, for Drum RPM = 3600

Figure 5.25

It is interesting to note carefully the slight changes in the primary levels, At the bottom of Figure 5.24 under primary access environment we see that mean critical service time at the core has increased slightly as a result of the increased traffic from the drum. This in turn has slowed the CPU slightly and reduced the congestion at the cache. This of course in turn reduces the critical service time at the cache. All of this results in a very slight increase in $\bar{\tau}_p$ (TAUP) given in Figure 5.22 and a very slight decrease in the mean CPU access rate when a user program is executing.

5.4 A Change in User Program Characteristics

We will now observe the effects of making changes in the user program characteristics. The specific variable to be changed here will be p. This variable controls the mode of the distribution of loop lengths in the user program model. In the "standard" case $p = .5$ making the most likely loop length 1/2 the length of the program. We will observe the effects of $p = .3$ and $p = .7$ here. The lifecycle function behavior for these values of p is shown in Figure 2.12 and this applies directly to the relationship between primary and secondary storage in this system.

Note that the system now being examined is identical to the "Standard" case except for the variable p.

In the case of $p = .3$, shown in Figures 5.26, 5.27, and 5.28, we have a small performance increase. The CPU is running faster

```
MEAN CPU ACCESS RATE = 0.68033350E 06

 ***  ALLOCATION  DATA  ***

  13 PROGRAMS WITH EACH USING  0.16384000E 06 BITS

LEVEL    TOTAL SPACE          ALLOCATION     % OF PROGRAM

  1      0.81920000E 04     0.81920000E 04      5.0000

  2      0.15000000E 07     0.11538456E 06     70.4251

  3      0.10000000E 08     0.76923075E 06    469.5010



CPU/PRIMARY EFFICIENCY= 30.667%


       CPU UTILIZATION

USER       22.184%

SYSTEM      3.174%

WAIT       74.642%

 ***  MEAN  TRAFFIC  ***

--OVER ALL TIME--

CPU ACCESS RATE =  0.68033350E 06
LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

  1       97.1760253906    0.71812312E 06    0.84494583E 02

  2        2.7928657532    0.38034605E 05    0.10574810E 04

  3        0.0061010942    0.83015533E 02    0.16384000E 05

--OVER PERIODS OF USER PROGRAM EXECUTION--

CPU ACCESS RATE =  0.30667380E 07
LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

  1       97.1760253906    0.32370820E 07    0.84494537E 02

  2        2.7928657532    0.17138269E 06    0.10314402E 04
```

Dependent Variables, Part 1, for p = .3

Figure 5.26

```
 *** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
     WHEN RUNNING  13 USER(S).

LEVEL  MAX REQUESTS
  1          3
  2         26
  3         26


 ***  ACCESS  TIMES  ***

--SECONDARY ACCESS ENVIRONMENT--

        MEAN              MEAN TOTAL     MEAN QUEUE   MEAN CRITICAL     RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME   LENGTH       SERVICE TIME      EFFECT %

  2     0.33085E 05     0.92295E-05     0.35150     0.0                  0.0

  3     0.83016E 02     0.61228E-01     5.08285     0.12011E-01        19.9384

 EXT    0.17008E 03     0.60000E-01    N.A.         0.48230E-01        80.0616

--PRIMARY ACCESS ENVIRONMENT--

        MEAN              MEAN TOTAL     MEAN QUEUE   MEAN CRITICAL     RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME   LENGTH       SERVICE TIME      EFFECT %

  1     0.32371E 07     0.39875E-06     1.29080     0.26752E-06        43.2156

  2     0.17138E 06     0.14932E-04     2.55905     0.35143E-06        56.7705
```

Dependent Variables, Part 2, for p = .3

Figure 5.27

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  0.6190333E-06

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  0.6024083E-01

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  0.1048450E-02

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  0.3260793E-06
```

Dependent Variables, Part 3, for p = .3

Figure 5.28

```
MEAN CPU ACCESS RATE = 0.30155381E 06

 ***  ALLOCATION  DATA  ***

  13 PROGRAMS WITH EACH USING  0.16384000E 06 BITS

LEVEL     TOTAL SPACE          ALLOCATION     % OF PROGRAM

  1      0.81920000E 04    0.81920000E 04       5.0000

  2      0.15000000E 07    0.11538456E 06     70.4251

  3      0.10000000E 08    0.76923075E 06    469.5310



CPU/PRIMARY EFFICIENCY= 28.445%


       CPU UTILIZATION

USER        10.601%

SYSTEM       2.093%

WAIT        87.306%

 ***  MEAN  TRAFFIC  ***

--OVER ALL TIME--

CPU ACCESS RATE =  0.30155381E 06
LEVEL   % USER ACCESS     READ/WRITE RATE      RECORD SIZE

  1       96.8908996582    0.31988612E 06    0.89283875E 02

  2        3.0628204346    0.18600383E 05    0.11299402E 04

  3        0.0212715007    0.12828996E 03    0.16384000E 05

--OVER PERIODS OF USER PROGRAM EXECUTION--

CPU ACCESS RATE =  0.28444660E 07
LEVEL   % USER ACCESS     READ/WRITE RATE      RECORD SIZE

  1       96.8908996582    0.30173890E 07    0.89283859E 02

  2        3.0628204346    0.17436937E 06    0.10353013E 04
```

Dependent Variables, Part 1, for p = .7

Figure 5.29

```
*** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
    WHEN RUNNING  13 USER(S).

LEVEL  MAX REQUESTS
  1         3
  2        26
  3        26


 ***  ACCESS  TIMES  ***

--SECONDARY ACCESS ENVIRONMENT--

       MEAN              MEAN TOTAL      MEAN QUEUE    MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME    LENGTH        SERVICE TIME     EFFECT %

  2     0.18600E 05      0.93388E-05       0.17370     0.0                 0.0

  3     0.12829E 03      0.13017E 00     16.69951      0.59841E-01        64.8623

 EXT    0.75388E 02      0.60000E-01      N.A.         0.32417E-01        35.1377

--PRIMARY ACCESS ENVIRONMENT--

       MEAN              MEAN TOTAL      MEAN QUEUE    MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME    LENGTH        SERVICE TIME     EFFECT %

  1     0.30174E 07      0.41771E-06       1.26040     0.27450E-06        40.9711

  2     0.17437E 06      0.15264E-04       2.66162     0.39533E-06        59.0055
```

Dependent Variables, Part 2, for p= .7

Figure 5.30

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  0.6699914E-06

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  0.9225792E-01

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  0.7597762E-03

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  0.3515598E-06
```

Dependent Variables, Part 3, for p = .7

Figure 5.31

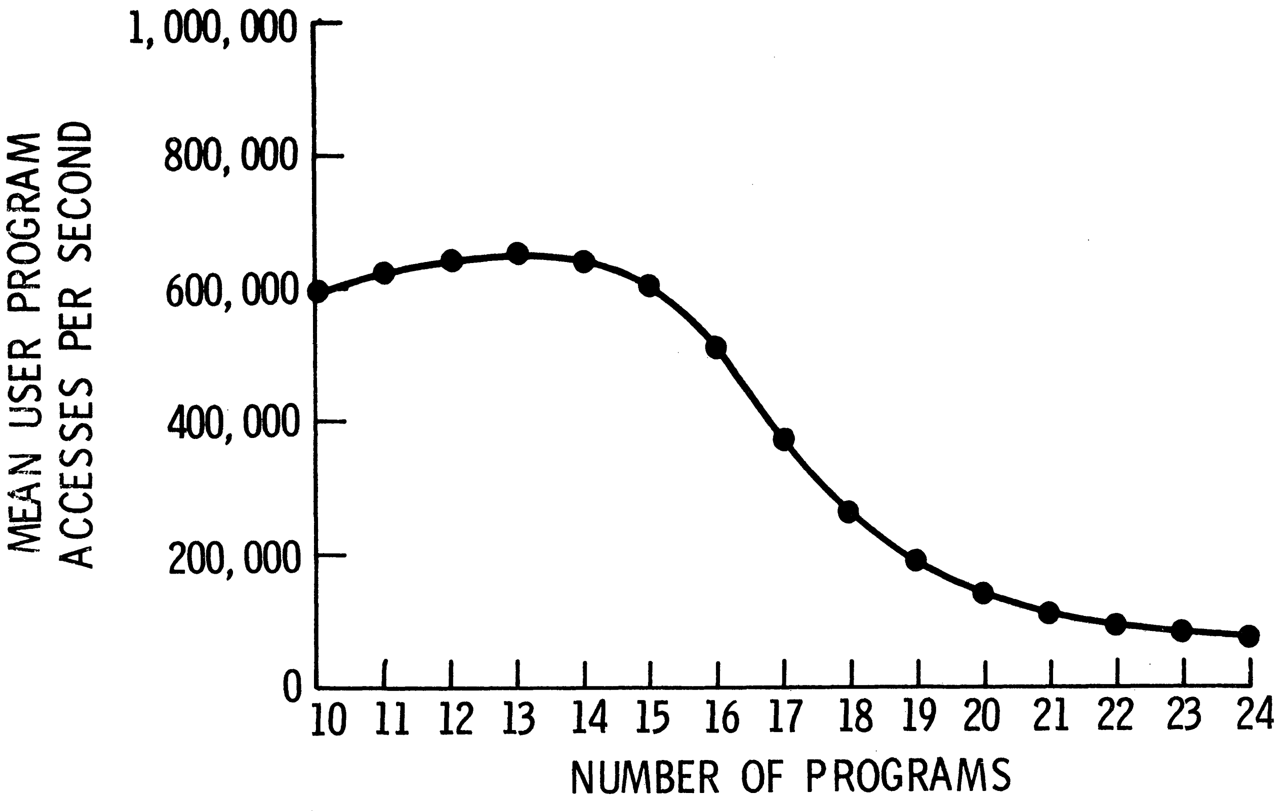
when executing a user program raising the CPU/PRIMARY EFFICIENCY to 30.6%. This increase is influenced by at least 2 factors. First, executing program will require fewer accesses to core. (See %User Access in Part 1.) Second, the congestion at core is reduced by the reduced drum-to-core activity. This all results in an increase in CPU execution rate, which causes an increase in general performance while increasing the CPU idle time slightly.

In this case, both p = .3 and p = .5 (the "standard") represent cases where the core allocation is adequate and thus the difference in performance shown is small. In the case of p = .7 (shown in Figures 5.29, 5.30, and 5.31) we no longer have adequate allocation in core and the performance changes are more dramatic.

The % of user accesses to level 3 jumps by a factor of 3 and the mean queue length at the drum rises to over 16. Here we see the delays causes by the drum begin to predominate over the IØ or external interrupts. The overall performance of the system is reduced to a little less than 1/2 of that in the standard case as a fairly direct result of the increased secondary paging activity and associated congestion.

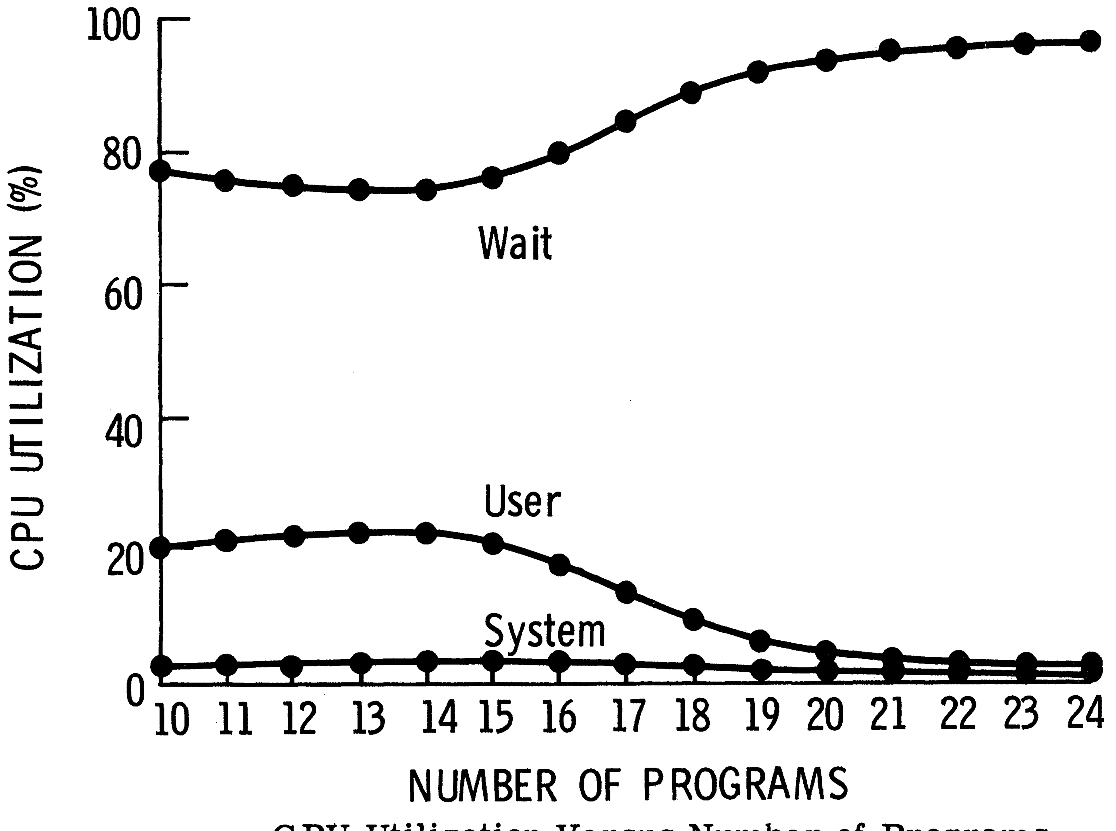
5.5 Changing the Number of User Programs

Here we will observe the changes in performance which occur as a result of varying the number of user programs being multiprogrammed. Our approach will be different in that rather than providing



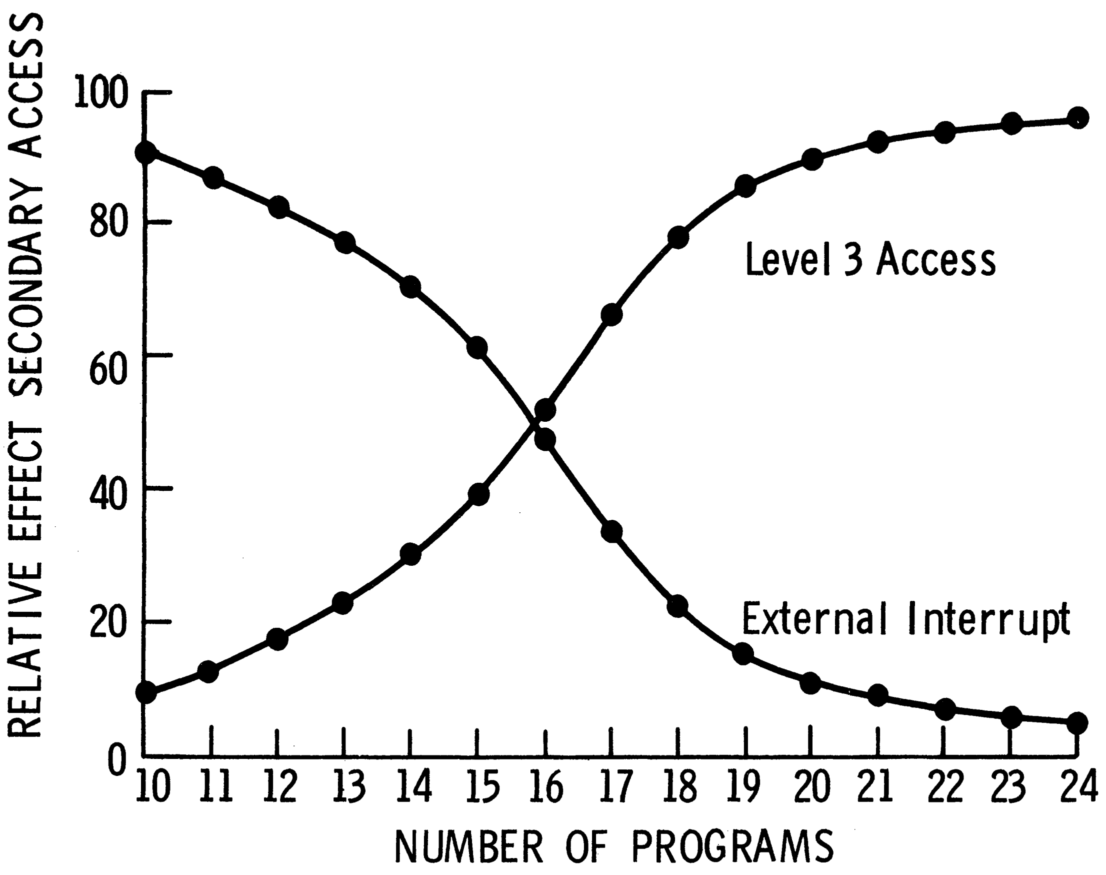
Performance Versus Number of Programs

Figure 5.32

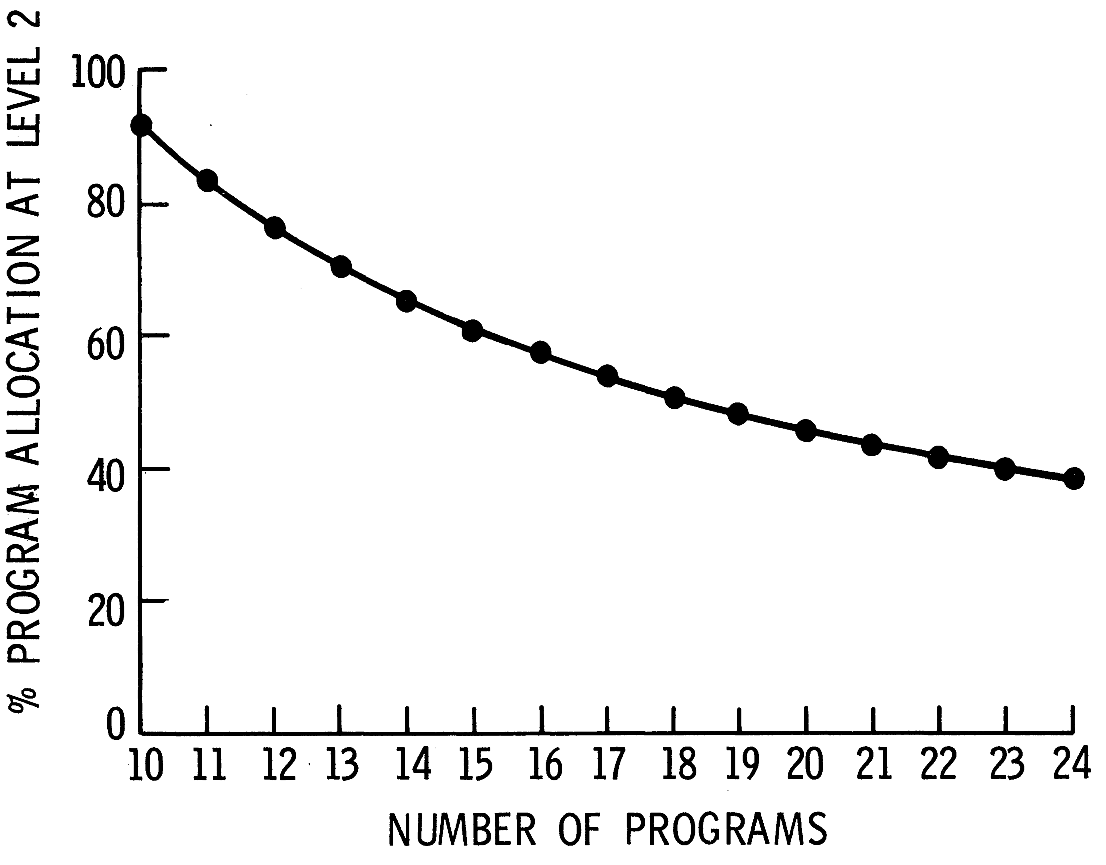


CPU Utilization Versus Number of Programs

Figure 5.33



Relative Effect Versus Number of Programs
Figure 5.34



Allocation Versus Number of Programs

Figure 5.35

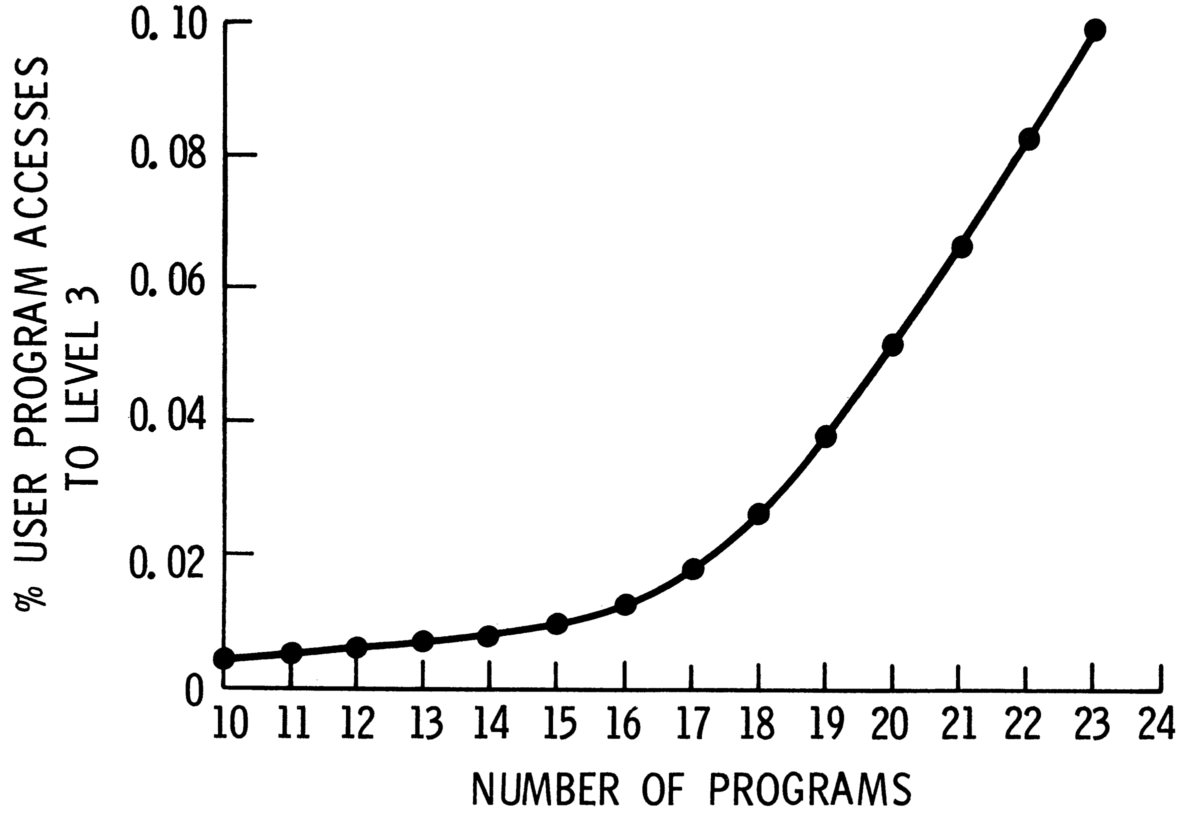
all the details of 1 or 2 cases, we will plot some of the more interesting dependent variables as a function of the number of programs running.

The system performance or mean number of user program accesses per second is shown in Figure 5.32 plotted as a function of the number of programs. We see very quickly that the maximum performance is achieved with 13 user programs, the number chosen for the "standard" example.

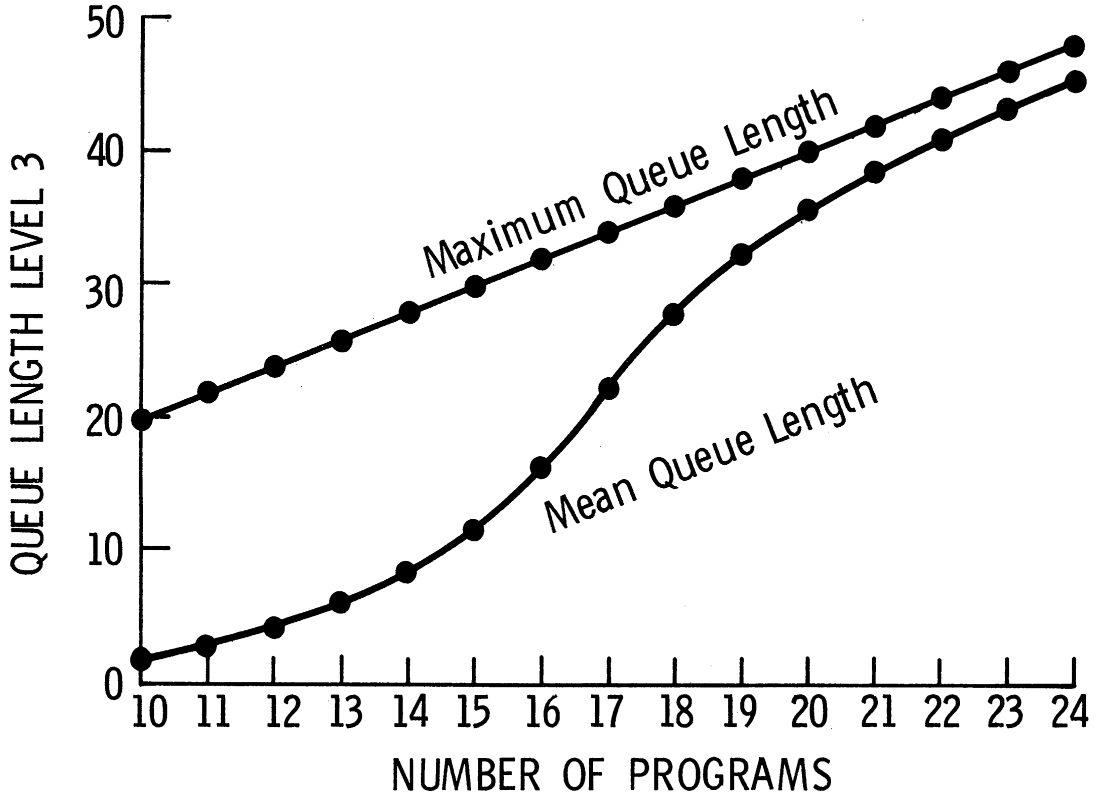
Figure 5.33 displays the CPU utilization figures. Notice that the peak in system activity occurs to the right of 13 user programs. This is a result of a higher rate of user program interchanges occuring for the larger number of programs. The basis for this will become clearer as we proceed.

Figure 5.34 shows the relative effect of the drum at level 3 and the external or IØ interrupts. When we run fewer programs the external interrupts dominate the secondary delays. As we increase the number of programs the drum at level 3 becomes the dominant factor.

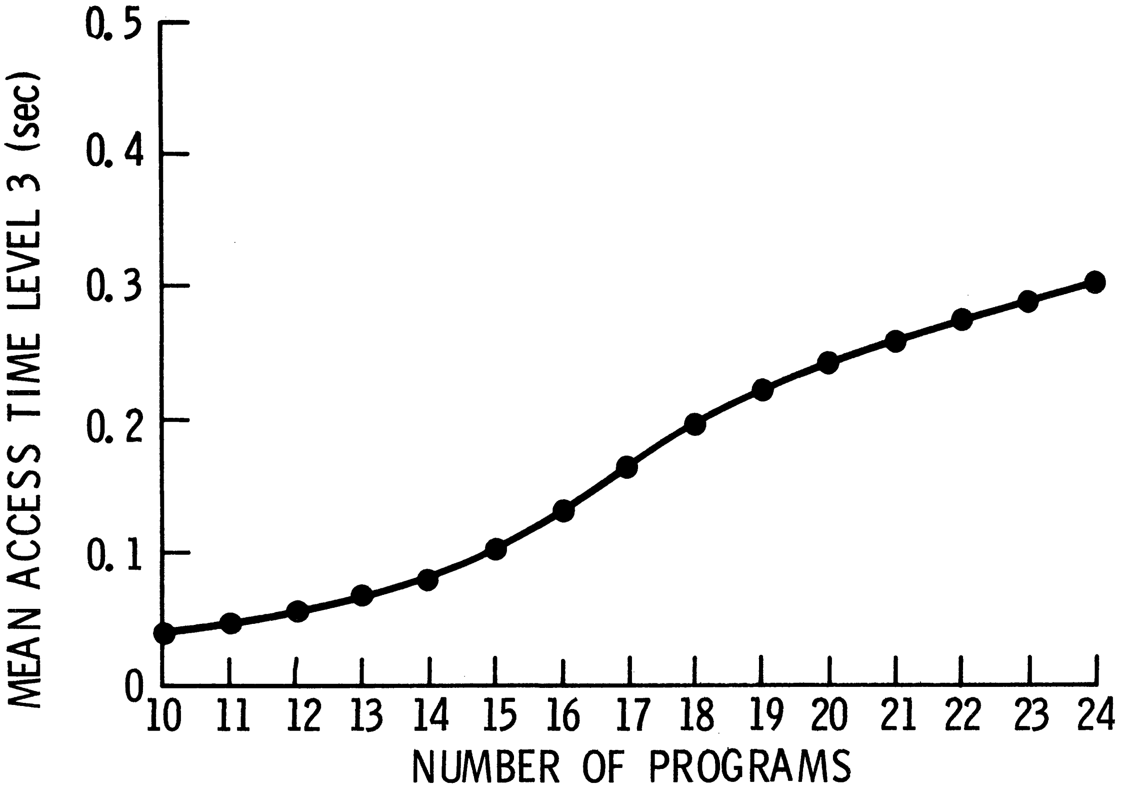
We will begin at Figure 5.35 to show the cause of this shift in dominance. Here we have plotted the core allocation as a function of the number of programs. On the left with 10 user programs over 90% of the user programs can be found in core. As we move to the right to 24 user programs the allocation in core is reduced to less than 40% of the user programs. Thus in the range from 10 to 24 programs we move



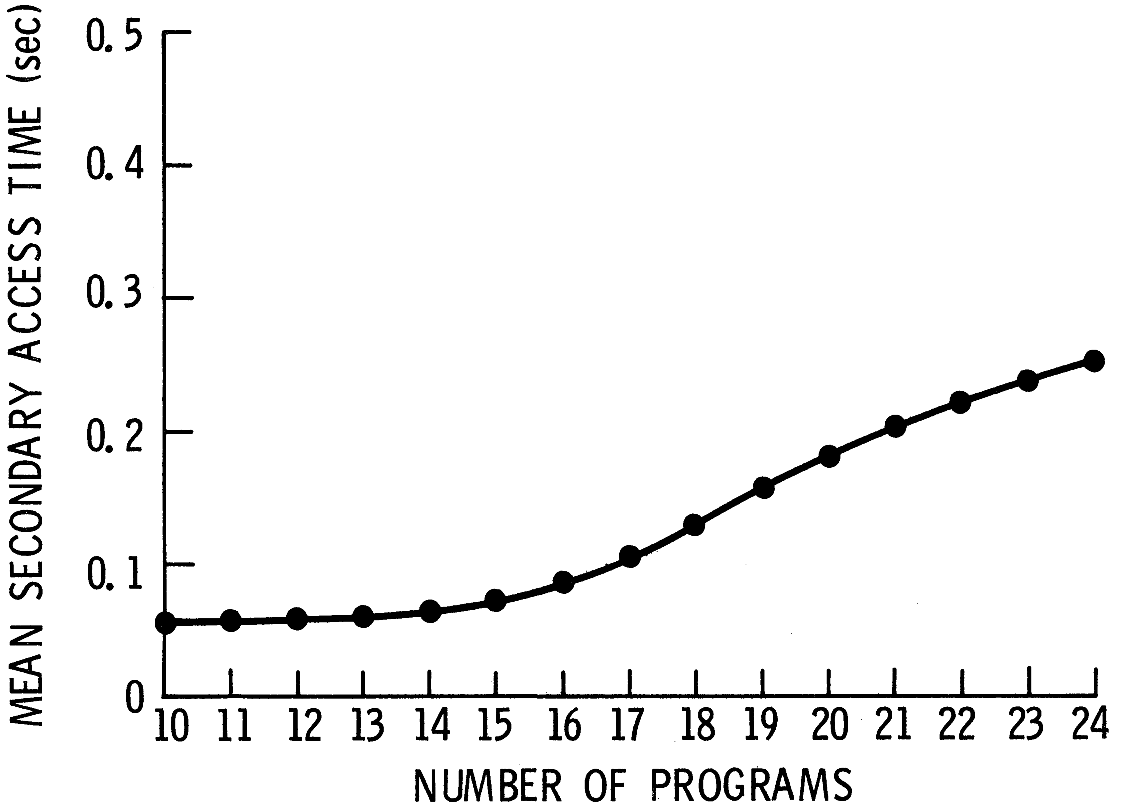
% of Level 3 Accesses Versus Number of Programs
Figure 5.36



Queue Length Versus Number of Programs
Figure 5.37



Access Time Versus Number of Programs
Figure 5.38



Secondary Access Time Versus Number of Programs
Figure 5.39

from a situation of more than sufficient space in core to a condition of severe under-allocation. Figure 5.36 shows the % of user program accesses to the drum as a function of the number of user programs. When compared to the previous figure we can see that the sharp increase in accesses to the drum begins as the allocations drop below 60% of the user program. This should be expected since this is the region of transition or greatest slope in the lifecycle function.

The queueing at the drum is shown in Figure 5.37. First we see that the maximum number in queue at level 3 increases linearly as we would expect. Again as we increase the number of programs running, to the point of causing poor allocation at core, the drum queue increases and approaches the maximum. Figure 5.38 shows the mean access time at the drum. This access time follows the queue length and the effect of the maximum queue length can be seen on the right.

Lastly, Figure 5.39 shows the mean secondary access time. For a smaller number of programs the secondary access time is dominated by the external interrupt behavior and thus is roughly 60 ms. For a larger number of user programs, the secondary access time behaves more like the level 3 access time, which becomes dominant as more user programs are introduced.

5.6 Summary

We have only been able to scratch the surface of analysis possibilities here and it is a short scratch at that. We have not varied the number of levels, types of devices, data paths, logical record sizes, or the CPU behavior,and this certainly is not a complete list.

One of the most interesting cases of analysis was undertaken when attempting to choose a "standard" case to make variations around. Most all of the initial systems worked with had a severe bottleneck. This severe bottleneck dominated the performance of the system to the point of obscuring most of the interesting interrelationships shown in the previous examples. One of the most stifling bottlenecks turns out to be the CPU. The choice of such poor CPU utilization in the "standard" case was most deliberate.

Chapter 6

System Optimization

In this chapter we will be concerned with optimization. After a statement of the problem we will consider several examples.

6.1 Objective Function and Method

The objective of the optimization will be to maximize r, the mean user program access rate, with a dollar constraint placed on the total value of the storage hardware. The choice of r as an objective function means that we are maximizing the system throughput. We are ignoring other system characteristics such as response time.

Decision Variables

Simply stated the decision variables are:

1. The number of user programs being multiprogrammed.
2. The number of storage units at each level.
3. The logical record size at each level except level 1.

More carefully:

The number of user programs, M, must be an integer satisfying

$$M^{min} \leq M \leq M^{max}$$

where

M^{min} = Minimum allowable number of user programs

and

M^{max} = Maximum allowable number of user programs.

We define:

N_i = Number of storage units at level i

(Note that N without subscript is the number of levels)

The decision variables N_i must be an integer satisfying

$$N_i^{min} \leq N_i \leq N_i^{max}$$

where

N_i^{min} = Minimum allowable number of storage units at level i

and

N_i^{max} = Maximum allowable number of storage units at level i

The logical record or page size, q_i, is considered a decision variable at all levels except level 1, or $i = 1$. The variable q_i must satisfy

$$q_i^{min} \leq q_i \leq q_i^{max}$$

where

q_i^{min} = Minimum allowable record size at level i.

and

q_i^{max} = Maximum allowable record size at level i.

In addition q_i must satisfy:

$$q_i = 2^n \text{ for some positive integer } n$$

Thus far we have considered rather direct constraints imposed on the decision variables. We will now consider several less direct constraints which are applied to eliminate infeasible systems.

First we require that the logical record size at level $i + 1$ be greater than or equal to the logical record size at level i or:

$$q_{i+1} \geq q_i$$

If this constraint is not satisfied the paging activity simply does not make sense.

Next we will require that the allocated space at each level increase as we move down in the hierarchy. This can be expressed in the form of 2 relations:

$$N_{i+1}\, S_{i+1}^{\text{unit}} \geq N_i S_i^{\text{unit}}$$

and

$$\frac{N_{i+1}\, S_{i+1}^{\text{unit}}}{M} \geq N_L S_L \qquad \text{for } L > 0$$

where S_i^{unit} = the storage unit capacity at level i

The first of these relations simply says that the total capacity at each level i + 1, $N_{i+1}\, S_{i+1}^{\text{unit}}$, must be greater than the total capacity at level i. This is sufficient to guarantee increasing allocation at each lower level except at the interface between dedicated and shared storage. The second relation applies to this dedicated-shared interface if it exists ($L > 0$).

Next we are concerned with a relationship between storage allocation and logical record size. The space allocated to a user program at level i must be equal to or greater than the record size at the next lower level i + 1. If this is not the case there is simply insufficient space at level i to store a record brought up from level i + 1. This again requires 2 relations - one for the dedicated levels and one for the shared levels:

$$N_i S_i^{unit} \geq q_{i+1} \qquad \text{for } i \leq L$$

and

$$\frac{N_i S_i^{unit}}{M} \geq q_{i+1} \qquad \text{for } i > L$$

Next we will require that there is sufficient capacity at level N to accommodate all of the M user programs:

$$N_N S_N^{unit} \geq MQ$$

where Q is the size of the user programs.

We will also require that an N level system be operated as such. That is we will not allow a level above level N to be sufficiently large to contain all of the M user programs running thus effectively making the lower levels useless. This is simply expressed:

$$N_i S_i^{unit} < MQ \qquad \text{for } i < N.$$

Last and very important is the dollar cost constraint. Defining

C_i^{unit} = dollar cost per unit of storage at level i

and

C^{max} = maximum allowable dollar expenditure

We will limit the number of devices at each level such that:

$$C^{max} \geq \sum_{i=1}^{N} N_i C_i^{unit}$$

and

$$C^{max} - \sum_{i=1}^{N} N_i C_i^{unit} < \min \{C_i^{unit}, C_2^{unit}, \ldots, C_n^{unit}\}$$

The first relation limits the cost of the system to C^{max} or less and the second relation requires that difference between the cost of the system and the maximum allowable cost not equal or exceed the cost of any single unit of storage.

This last constraint requires some discussion. It is reasonable if we can assume that the addition of a storage unit always improves the system. This may be a reasonable assumption for some systems and not for others. For those systems where we are not willing to make this assumption we must vary C^{max} by increments of $\min\{C_1^{unit}, C_2^{unit}, \ldots, C_n^{unit}\}$ to get a series of optimums for ranges of system cost and select the global optimum from that series.

The constraints just discussed leave us with a set of feasible configurations. We will now turn our attention to the process of selecting the optimal configuration from the feasible set.

The approach is simply an intelligently ordered exhaustive search of the set of feasible configurations. Generally speaking there are 2 areas where significant economics are introduced in the search.

The first of these economics involves the system model itself. As you will recall, the process of analysis involves a search over values of r. At each step in this search we assume an r (mean user program access rate) and the model's response simply tells us if r is too large or too small. In the search for an optimal configuration we will need only to determine if a given configuration will perform better or worse than the highest performance thus far encountered in the search. If a configuration's performance falls below the highest previously encountered performance no further analysis is required and the configuration can be eliminated. If a configuration does in fact have a performance higher than any previously encountered, a complete analysis is performed to determine its precise performance. Since only a small minority of the configurations will require a complete analysis a savings of roughly 20 to 1 is experienced.

The second area of economy of effort is found in the ordering of the search. The reduction is achieved by making use of the fact that certain intermediate results can be saved and used repeatedly as the search proceeds.

6.2 A Simple Example

We will now consider an optimization of the "standard" case discussed in the previous chapter.

Before becoming involved in the optimization itself we should say a few words about the computer program performing the optimization. First, optimization and analysis are performed by the same program. From the point of view of analysis any of the models' independent variables may be modified. When an optimization is requested, the decision variables in the model are adjusted within bounds to maximize the mean user program access rate.

Figure 6.1 shows the decision variables. (Note that the record size at level 1 is not considered a decision variable and is included here only for symmetry.) The center column labled CURRENT shows the values of the decision variables. The columns on the left and right show the imposed bounds on these variables.

The current values shown in Figure 6.1 are simply those of the "standard" case discussed in the previous chapters. Following optimization these values will be adjusted tc maximize the mean user program access rate.

Under the given bounds on decision variables we will examine configurations running from 2 to 25 user programs, using from 1 to 4 caches, 2 to 4 cores, and 1 to 4 drums. The logical record size at level 2 may range from 128 bits to 4096 bits and at level 3 from 4096 bits to 65536 bits. The record size at level 1 is not considered a decision variable and thus will be fixed at 32 bits.

For a first optimization we set the maximum cost, C^{max}, at $250,000. This will allow 5 units at $50,000 a piece in the system. Note that this does not change the number in the system but may allow for some rearrangement.

The optimization reported a total of 20,124 possible configurations of which 972 were feasible and required a CPU time (IBM 360/67) of 42 sec. The results of this optimization are shown in Figures 6.2 through 6.5. Figure 6.2 shows the decision variables. We can see that the optimum system differs from the "standard" case in 3 decision variables. The number of user programs has dropped to 12 from 13.

```
DECISION VARIABLES

                         MINIMUM        CURRENT        MAXIMUM

# OF PROGRAMS                2             13             25

* LEVEL 1 *
# OF UNITS                   1              1              4
RECORD SIZE                 32             32             32

* LEVEL 2 *
# OF UNITS                   2              3              8
RECORD SIZE                128           1024           4096

* LEVEL 3 *
# OF UNITS                   1              1              4
RECORD SIZE               4096          16384          65536
```

Decision Variables for "Standard" Case

Figure 6.1

```
  DECISION VARIABLES

                        MINIMUM       CURRENT       MAXIMUM

# OF PROGRAMS                2            12            25

* LEVEL 1 *
# OF UNITS                   1             1             4
RECORD SIZE                 32            32            32

* LEVEL 2 *
# OF UNITS                   2             3             8
RECORD SIZE                128           256          4096

* LEVEL 3 *
# OF UNITS                   1             1             4
RECORD SIZE               4096         32768         65536
```

Decision Variables for Optimal $250,000 System

Figure 6.2

```
MEAN CPU ACCESS RATE = 0.65384700E 06

 ***  ALLOCATION  DATA  ***

  12 PROGRAMS WITH EACH USING  0.16384000E 06 BITS

LEVEL     TOTAL SPACE         ALLOCATION     % OF PROGRAM

  1      0.81920000E 04    0.81920000E 04      5.0000

  2      0.15000000E 07    0.12500000E 06     76.2939

  3      0.10000000E 08    0.83333331E 06    508.6262


CPU/PRIMARY EFFICIENCY= 29.310%


        CPU UTILIZATION

USER       22.308%

SYSTEM      2.807%

WAIT       74.885%

 ***  MEAN  TRAFFIC  ***

--OVER ALL TIME--

CPU ACCESS RATE =   0.65384700E 06
LEVEL    % USER ACCESS     READ/WRITE RATE       RECORD SIZE

  1       88.1710968018     0.80797094E 06     0.74780945E 02

  2       11.8002767563     0.15435869E 06     0.26596411E 03

  3        0.0036176760     0.47308090E 02     0.32767992E 05

--OVER PERIODS OF USER PROGRAM EXECUTION--

CPU ACCESS RATE =  0.29309680E 07
LEVEL    % USER ACCESS     READ/WRITE RATE       RECORD SIZE

  1       88.1710968018     0.36218520E 07     0.74780914E 02

  2       11.8002767563     0.69177156E 06     0.25822339E 03
```

Dependent Variables, Part 1, for Optimal $250,000 System

Figure 6.3

```
 *** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
     WHEN RUNNING  12 USER(S).

LEVEL  MAX REQUESTS
  1          3
  2         24
  3         24


 ***  ACCESS  TIMES  ***

--SECONDARY ACCESS ENVIRONMENT--

       MEAN             MEAN TOTAL        MEAN QUEUE     MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE     SERVICE TIME      LENGTH         SERVICE TIME     EFFECT %

  2     0.15436E 06     0.23266E-05        0.35913       0.0                0.0

  3     0.47308E 02     0.82065E-01        3.88234       0.10374E-01       16.5222

 EXT    0.16346E 03     0.60000E-01      N.A.            0.52415E-01       83.4779

--PRIMARY ACCESS ENVIRONMENT--

       MEAN             MEAN TOTAL        MEAN QUEUE     MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE     SERVICE TIME      LENGTH         SERVICE TIME     EFFECT %

  1     0.36219E 07     0.35093E-06        1.27100       0.24397E-06       37.5785

  2     0.69177E 06     0.37721E-05        2.60941       0.40521E-06       62.4134
```

Dependent Variables, Part 2, for Optimal $250,000 System

Figure 6.4

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  0.6492402E-06

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  0.6278926E-01

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  0.1192216E-02

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  0.3411842E-06
```

Decision Variables, Part 3, for Optimal $250,000 System

Figure 6.5

The record size at level 2 has dropped from 1024 bits to 256 bits and the record size at level 3 has risen from 16384 to 32768.

It is interesting to note that the change in record sizes has caused a shift in the optimal number of user programs. (Section 5, Chapter 5 shows that the optimal number of user programs is 13 when all else is fixed.)

Comparing the dependent variables of this optimal system, shown in Figures 6.3, 6.4, and 6.5, with the "standard" case we see that the performance has improved only slightly. In fact the mean user program access rate has increased less than 1%. Thus the "standard" case is very near optimal.

We will now see what happens when we increase the maximum expenditure, C^{max}. Here we will increase C^{max} by $50,000 to $300,000, allowing a single additional storage unit in the system. The optimization reported 20,124 possible configurations of which 1362 were feasible. The optimization time was 62 sec. of CPU time.

The new optimal system is shown in Figures 6.6 through 6.9. We see that an additional core unit was added to the system and the performance has increased considerably. The new performance, r, is 25% greater than that of the previous optimal system. We see also that the number of user programs has increased and the record size at level 2 has decreased again.

```
  DECISION VARIABLES

                        MINIMUM       CURRENT       MAXIMUM

# OF PROGRAMS                 2            15            25

* LEVEL 1 *
# OF UNITS                    1             1             4
RECORD SIZE                  32            32            32

* LEVEL 2 *
# OF UNITS                    2             4             8
RECORD SIZE                 128           128          4096

* LEVEL 3 *
# OF UNITS                    1             1             4
RECORD SIZE                4096         32768         65536
```

Decision Variables for Optimal $300,000 System

Figure 6. 6

```
MEAN CPU ACCESS RATE = 0.81687000E 06

 ***  ALLOCATION  DATA  ***

  15 PROGRAMS WITH EACH USING  0.16384000E 06 BITS

LEVEL    TOTAL SPACE          ALLOCATION     % OF PROGRAM

  1      0.81920000E 04    0.81920000E 04      5.0000

  2      0.20000000E 07    0.13333331E 06     81.3332

  3      0.10000000E 08    0.66666662E 06    406.9009


 CPU/PRIMARY EFFICIENCY= 31.364%


        CPU UTILIZATION

 USER        26.044%

 SYSTEM       3.448%

 WAIT        70.508%

  ***  MEAN  TRAFFIC  ***

 --OVER ALL TIME--

 CPU ACCESS RATE =   0.81687000E 06
 LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

   1       76.5190429688    0.11997970E 07    0.62657791E 02

   2       23.4528045654    0.38320894E 06    0.13236734E 03

   3        0.0031385105    0.51275055E 02    0.32767992E 05

 --OVER PERIODS OF USER PROGRAM EXECUTION--

 CPU ACCESS RATE =   0.31364430E 07
 LEVEL   % USER ACCESS    READ/WRITE RATE     RECORD SIZE

   1       76.5190429688    0.46367250E 07    0.62657776E 02

   2       23.4528045654    0.14712180E 07    0.12913759E 03
```

Decision Variables, Part 1, for Optimal $300,000 System

Figure 6.7

```
*** MAXIMUM POSSIBLE NUMBER OF REQUESTS FOR SERVICE
    WHEN RUNNING  15 USER(S).

LEVEL  MAX REQUESTS
  1          3
  2         30
  3         30


 ***  ACCESS  TIMES  ***

--SECONDARY ACCESS ENVIRONMENT--

       MEAN              MEAN TOTAL      MEAN QUEUE    MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME    LENGTH        SERVICE TIME     EFFECT %

  2     0.38321E 06     0.11478E-05      0.43986     0.0                  0.0

  3     0.51275E 02     0.95643E-01      4.90410     0.10668E-01        16.6748

 EXT    0.20422E 03     0.60000E-01     N.A.         0.53308E-01        83.3251

--PRIMARY ACCESS ENVIRONMENT--

       MEAN              MEAN TOTAL      MEAN QUEUE    MEAN CRITICAL    RELATIVE
LEVEL  SERVICE RATE      SERVICE TIME    LENGTH        SERVICE TIME     EFFECT %

  1     0.46067E 07     0.30536E-06      1.40672     0.22872E-06        37.8335

  2     0.14712E 07     0.16041E-05      2.36003     0.37580E-06        62.1632
```

Dependent Variables, Part 2, for Optimal $300,000 System

Figure 6.8

```
MEAN TIME TO COMPLETE A PRIMARY ACCESS      TAUP=  Ø.6Ø45366E-Ø6

MEAN TIME TO COMPLETE A SECONDARY ACCESS    TAUS=  Ø.6397551E-Ø1

MEAN EXECUTION TIME
BEFORE SECONDARY PAGE FAULT                 TAUE=  Ø.113Ø383E-Ø2

MEAN TIME BETWEEN PRIMARY ACCESSES          TAUC=  Ø.3188325E-Ø6
```

Dependent Variables, Part 3, for Optimal $300,000 System

Figure 6.9

6.3 Changing the User Program Size

In this section we will study how the behavior and characteristics of an optimal system change when the size of the user programs being run changes. As a simplification we will only optimize the number of user programs being run. To do this we simply set:

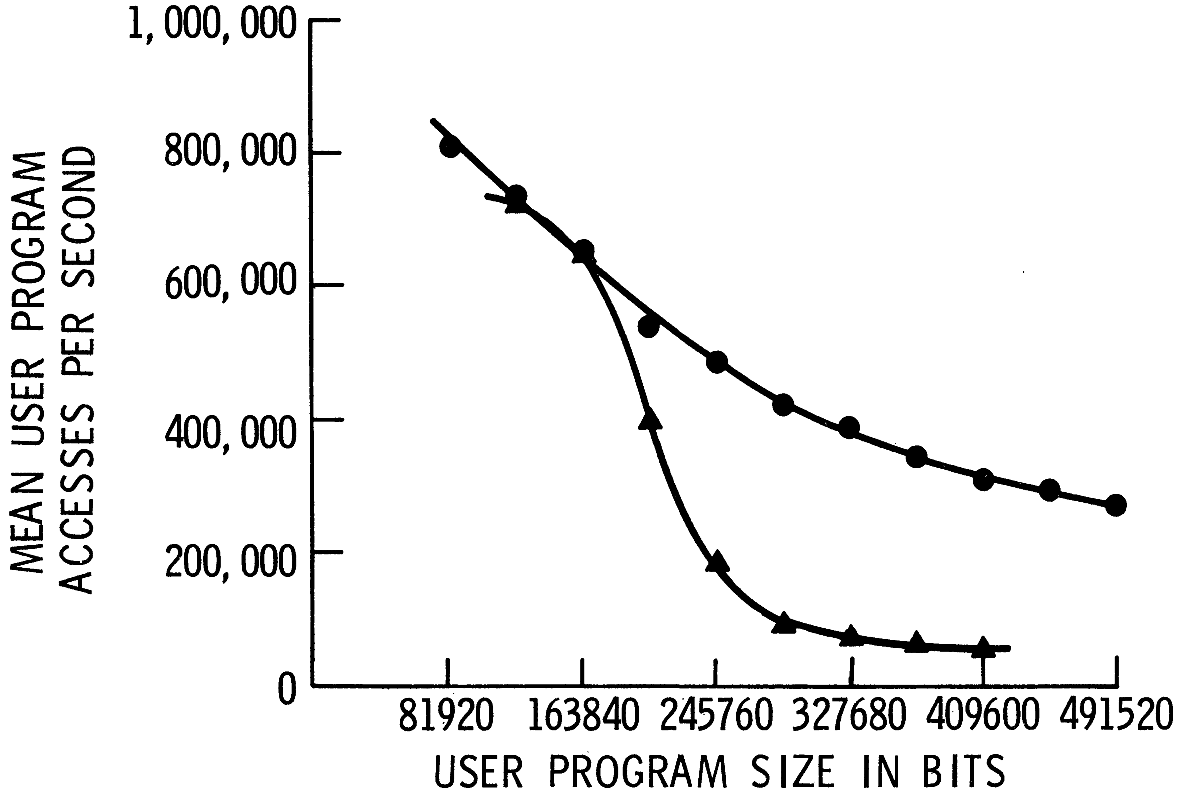
$$N_i^{max} = N_i^{min} = N_i \qquad \text{for all } i,$$

and

$$q_i^{max} = q_i^{min} = q_i \qquad \text{for all } i.$$

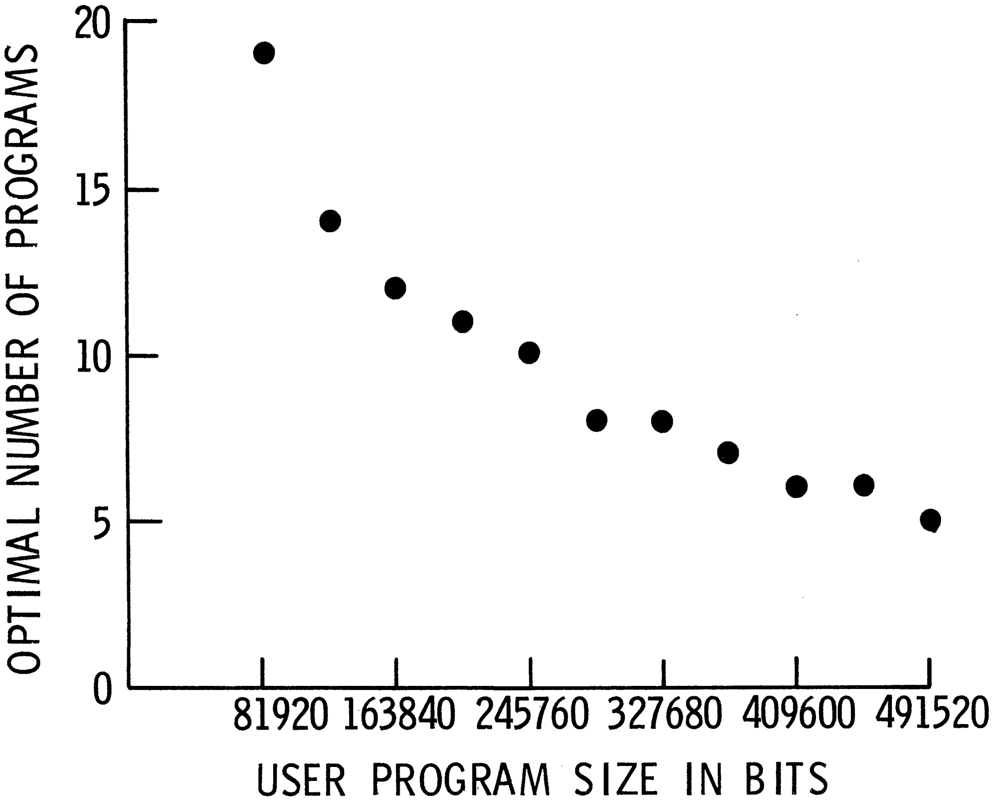
The performance as a function of user program size is shown in Figure 6.10. Here we see a variation in program size from 81920 bits to 491520 bits. The upper curve gives the performance of the system when an optimal number of user programs are run as a function of user program size. The lower curve shows how the system performance varies with program size when a fixed number of user programs are running (13). This lower curve simply shows what happens to the "standard" case when the user program size is changed. In general both the optimal and the fixed configuration decrease in performance with increased program size. The optimal system performance always exceeds the fixed system except for a program size of 163840 bits where the fixed and optimal coincide.

The optimal number of user programs is plotted against user program size in Figure 6.11. The core allocation at level 2 in terms of the % of user program size is shown in Figure 6.12.

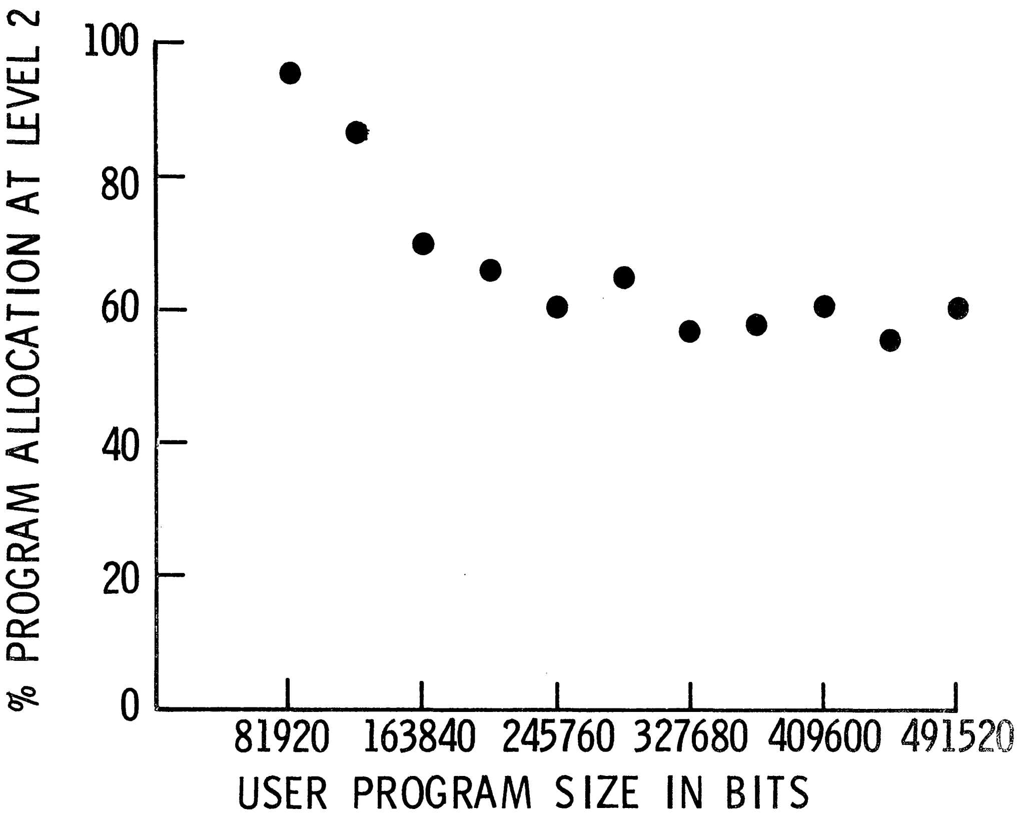


Performance Versus User Program Size

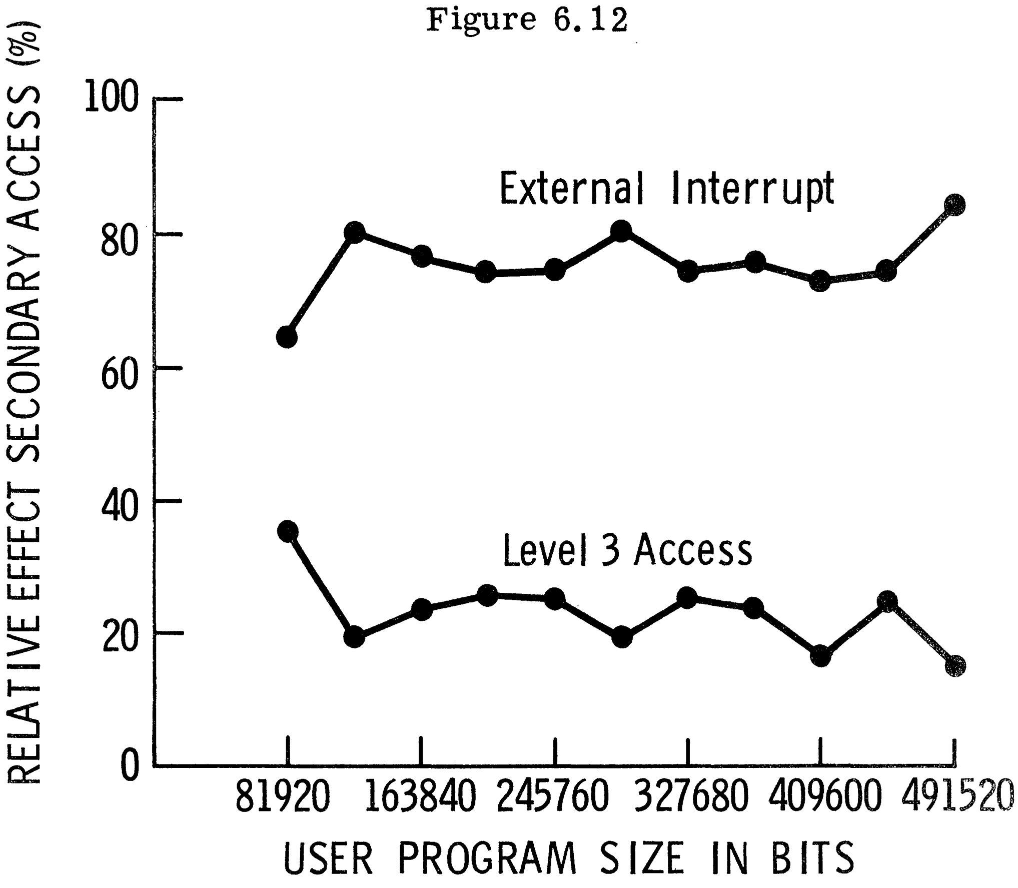
Figure 6.10



Optimal Number of Programs Versus Program Size
Figure 6.11



Allocation Versus User Program Size
Figure 6.12



Relative Effect Versus User Program Size
Figure 6.13

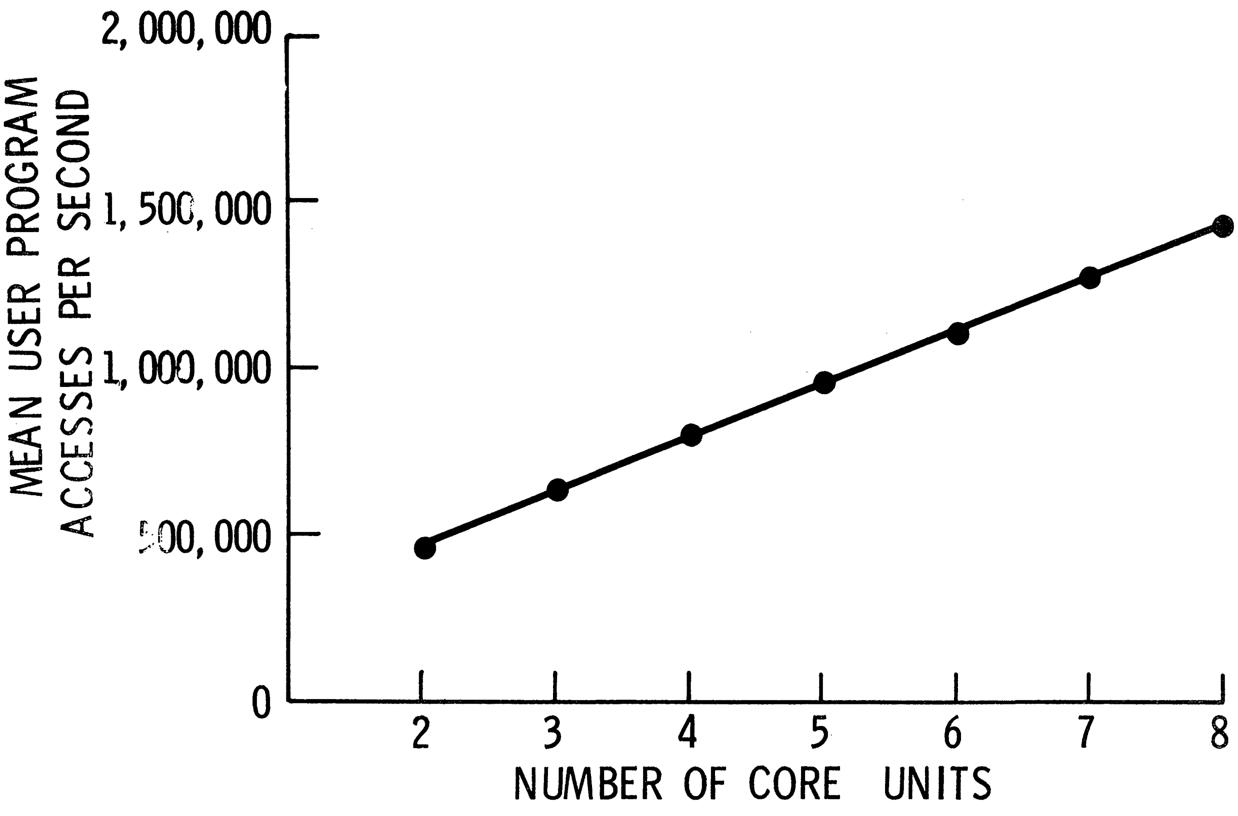
As one would expect, the optimal number of user programs decreases as the user program size increases. Notice that the resulting allocation also decreases as user program size increases. This means that the total size of collection of M user programs tends to increase with increased user program size. This suggests that as programs get bigger we are willing to run them with lower allocation in order to reduce the effects of external or IØ interrupts.

An interesting result of this experiment in optimization is shown in Figure 6.13. Here we see the relative effect of the external interrupt and the drum at level 3 on the secondary delay. Notice that for an optimal number of user programs these values remain relatively constant. This suggests that measures of the relative delay in a system might be useful parameters for use in a scheduling algorithm.

6.4 Changing the Number of Core Units

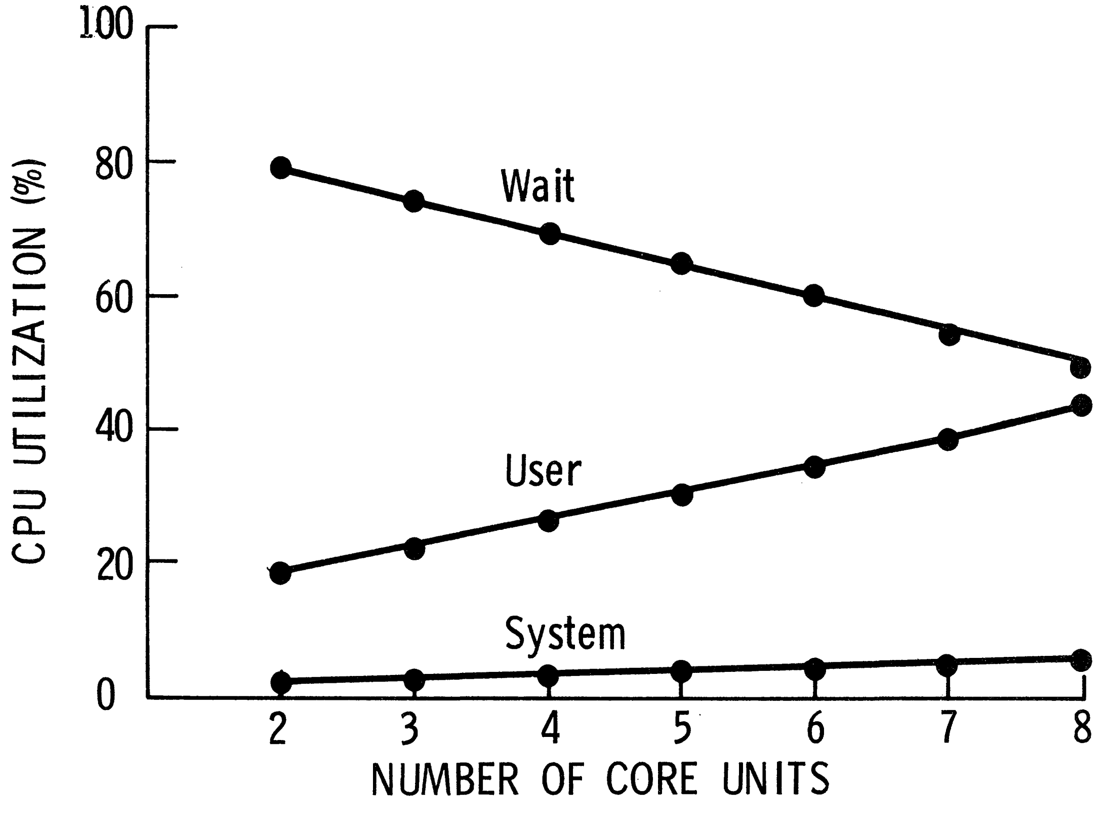
In this section we consider an optimization experiment similar to the optimization carried out in the previous section. Here we will study the characteristics of the optimal system as we increase the number of core units. We will again use the "standard" case with the exception of the variation in number of units at level 2 and the number of user programs being run. We will vary the number of core units in the system in each case optimizing the number of user programs running.

The optimal system performance versus the number of core units is shown in Figure 6.14. The performance increases with an



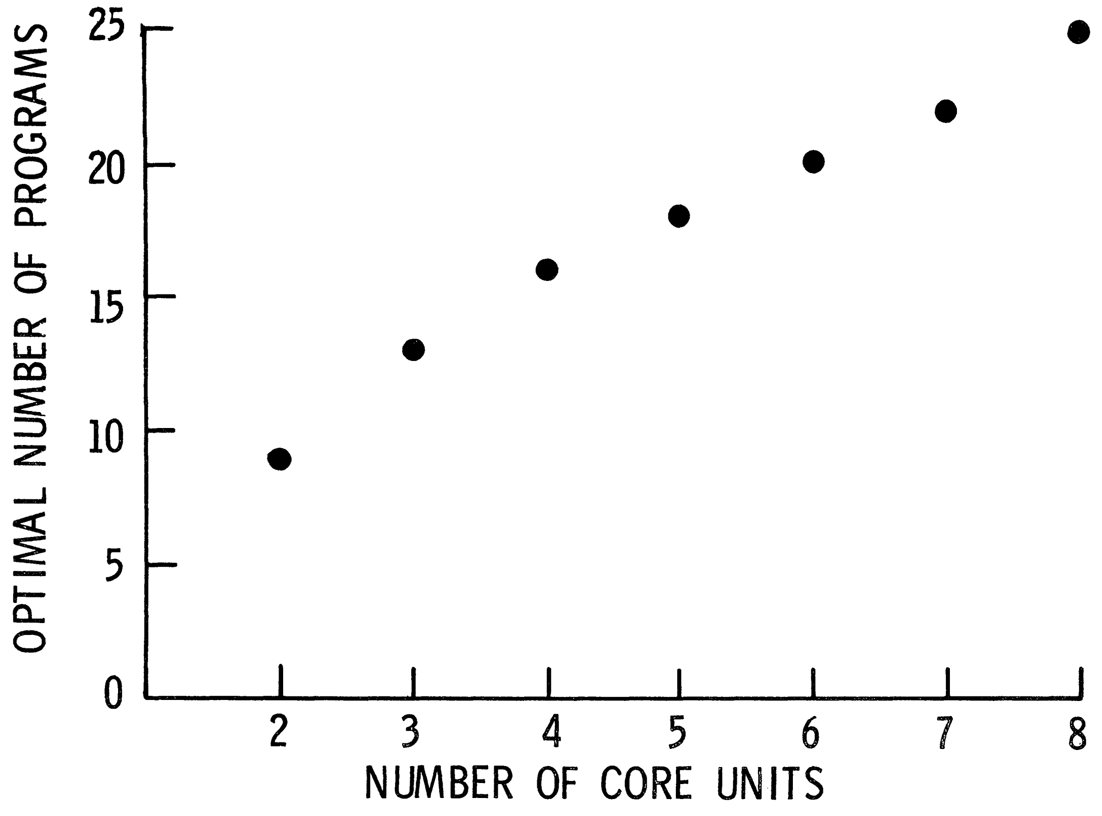
Optimal Performance Versus Number of Core Units

Figure 6.14

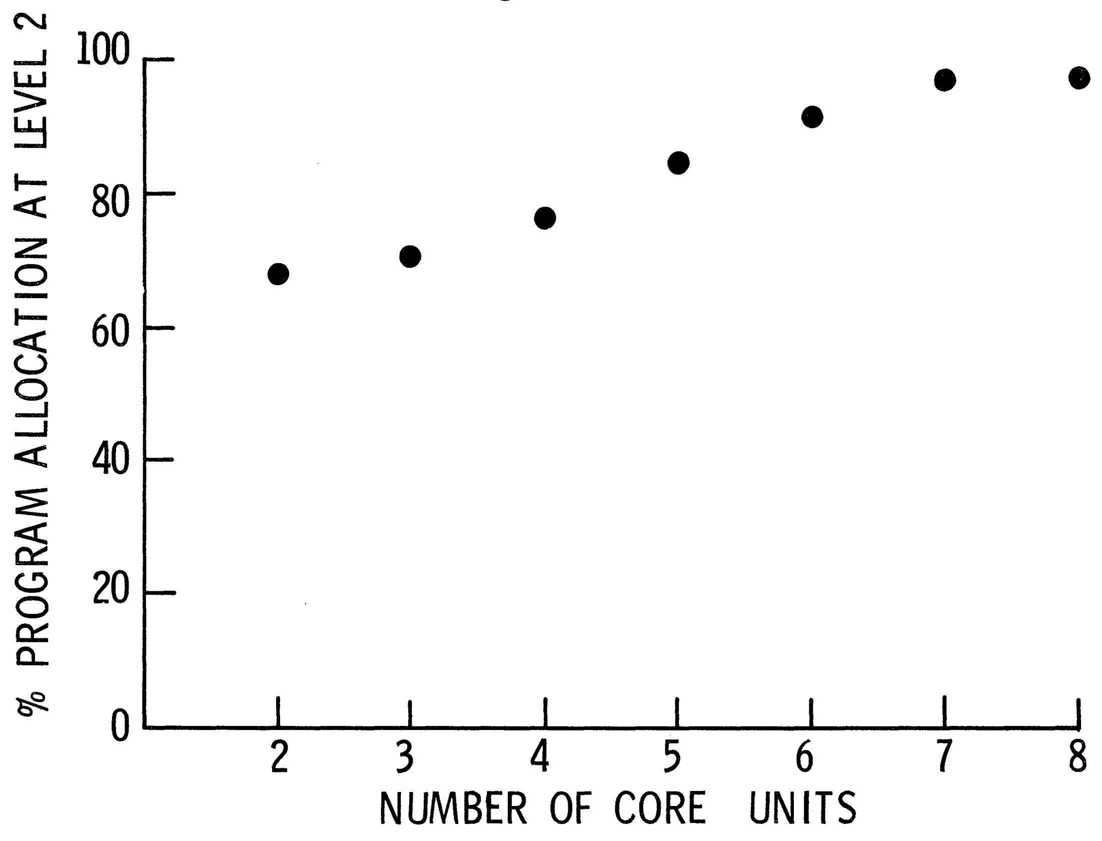


CPU Utilization Versus Number of Core Units

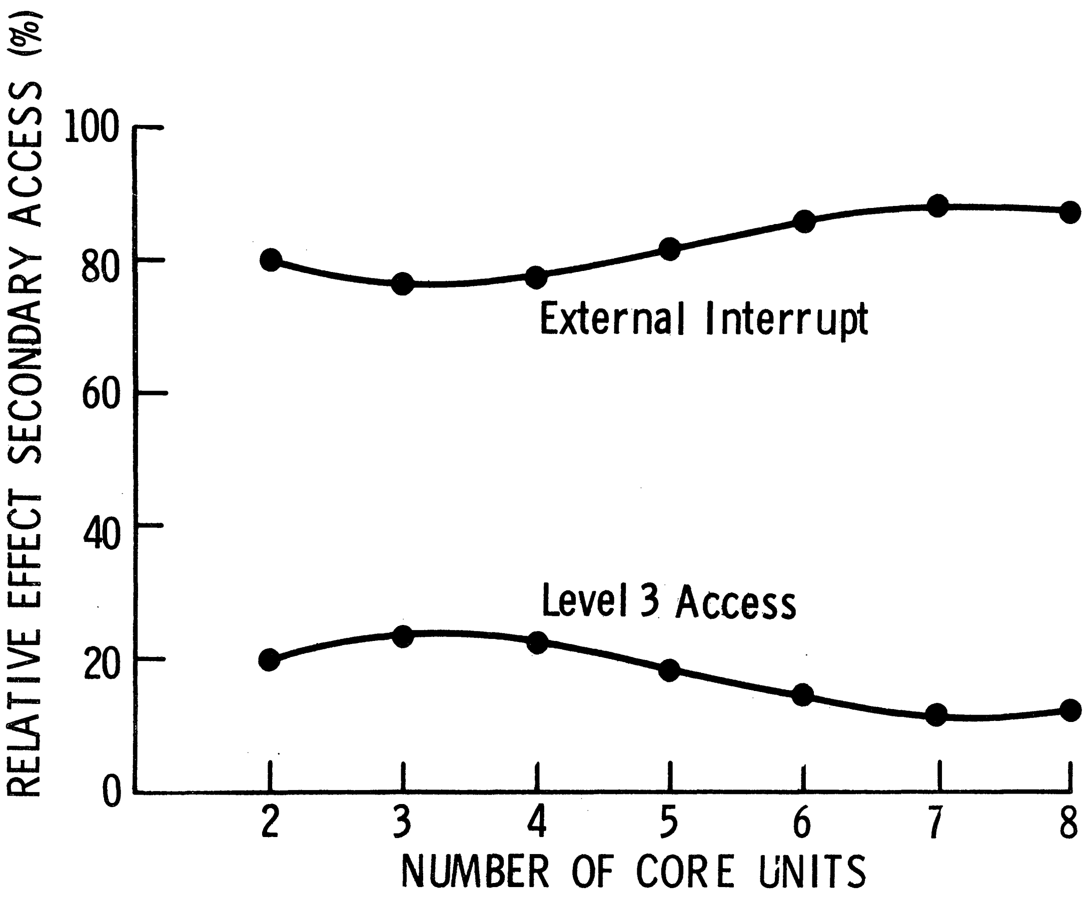
Figure 6.15



Optimal Number of Programs Versus Number of Core Units
Figure 6.16



Allocation Versus Number of Core Units
Figure 6.17



Relative Effect Versus Number of Core Units

Figure 6.18

increasing number of core units and in a surprisingly linear manner. Figure 6.15 shows the CPU utilization figures and we see that we never approach CPU saturation.

Figures 6.16 and 6.17 show the optimal number of programs and the corresponding allocation at level 2.

Figure 6.18 shows the relative effect of the external or IØ interrupts and the drum delay. Here we can again observe relatively constant values over a wide variation in number of core units.

The optimizations in this and the previous section are simple in that there is only one effective decision variable, M. In all cases M was allowed to assume values from 2 to 30 inclusive. The CPU time required to perform an individual optimization is approximately 2 seconds. In both this section and the previous section, 15 distinct optimizations were carried out for a total of 30 optimal solutions. The total CPU time required to find all of these optimal solutions was less than 1 minute. The point to be made is that we can not only analyze and optimize individual computing systems, but we can also study the characteristics of sets of optimal systems.

Chapter 7

Conclusions

The work described here has been directed toward the analysis, optimization and better understanding of computer systems. The core of the effort has been a mathematical model. The mathematical model is developed in Chapters 2, 3 and 4. It involves user program behavior, storage hardware behavior, CPU behavior, and system hardware and system software architecture.

After the model was developed and programmed a period of experimentation began in which a variety of system configurations were analyzed and optimized. Some impressions gained concerning both the model itself and the systems being studied are of interest.

Many of the results obtained from the model were counterintuitive. This was true for both analysis and optimization. In every case additional experimentation provided a better understanding of the system. The initially counterintuitive results were found to be consistent with an improved understanding of the system.*

The capability to explore the system, by modifying parameters and observing changes in performance, was extremely useful in identifying the important performance controlling characteristics of the system. This capability to explore a system is the result of 3 factors:

*Early in the experimentation several cases of true inconsistency were found in the model. These were traced to either program error or an error in the model itself. In each case the error was corrected.

1. Low cost of solution
2. Model flexibility
3. An interactive implementation of the model.

The CPU time required for analysis and/or optimization was small. In the case of analysis and the simpler optimizations, the cost of solving for the result was less than the cost of printing the result. The interactive nature of the implementing program is such that the user spends much more time deciding what changes to make than he does actually making the changes.

The approach taken in the design of the model deserves some discussion. The model was developed specifically for use in the form of a computer program. This has affected a number of important model design decisions.

The model is modular. The most important aspect of this modularity is the ease with which diverse mathematical methods can be applied to different portions of the problem. In choosing a method for modeling storage hardware there was no need to use the techniques employed to model the user program behavior. In each case the method used was developed for the specific application with little regard for methods used elsewhere. This modularity of method, which would lead to a uselessly cumbersome model outside the context of computer implementation, was instrumental in providing the realism and speed of solution demonstrated in Chapters 5 and 6.

If there is one characteristic of the mathematical model to be identified as contributing most to its worth as a tool for modeling computer systems,it would have to be this modularity of method.

Appendix A

Solution for Integrals of Equation 2. 54

Here we will evaluate the sum of integrals

$$\int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta + \int_{s-q}^{s} H_2(\Delta) f_\Delta(\Delta)\, d\Delta$$

$$+ \int_s^{q[Q/q]} H_3(\Delta) f_\Delta(\Delta)\, d\Delta \qquad \text{(A-1)}$$

where $H_1(\Delta)$, $H_2(\Delta)$, $H_3(\Delta)$ and $f_\Delta(\Delta)$ are defined in Equations 2.44, 2.52, 2.47 and 2.28 of Chapter 2.

There are three distinct integrals in the above expression. We will treat each separately beginning with

$$\int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta = A_1 \int_0^{s-q} \frac{\Delta \; d\Delta}{(\Delta + q)(2(q\lceil Q/q\rceil - s) + \Delta)\left(\frac{1}{e^2} + \left(p - \frac{\Delta}{q\lceil Q/q\rceil}\right)^2\right)}$$

where (A-2)

$$A_1 = \frac{2\,\sigma\,(Q/\lceil Q/q\rceil)}{q'e\,[\tan^{-1} e(1-p) + \tan^{-1} ep]}$$

We will define

$$a_1 = q \qquad \text{(A-3)}$$

$$a_2 = 2(q\lceil Q/q\rceil - s) \qquad \text{(A-4)}$$

$$a_3 = \frac{1}{e^2} + p^2 \qquad \text{(A-5)}$$

$$a_4 = -\frac{2p}{2\Gamma[Q/q]} \tag{A-6}$$

$$a_5 = \frac{1}{q^2\Gamma[Q/q]^2} \tag{A-7}$$

then we may write

$$\int_o^{s-q} H_1(\Delta)\, f_\Delta(\Delta)\, d\Delta = A_1 \int_o^{s-q} \frac{\Delta \ d\Delta}{(a_1+\Delta)(a_2+\Delta)(a_3 + a_4\Delta + a_5\Delta^2)} \tag{A-8}$$

Expanding using partial fraction

$$\frac{\Delta}{(a_1+\Delta)(a_2+\Delta)(a_3 + a_4\Delta + a_5\Delta^2)} =$$

$$\frac{K_1}{a_1 + \Delta} + \frac{K_2}{a_2 + \Delta} + \frac{K_3\Delta + K_4}{a_3 + a_4\Delta + a_5\Delta^2} \tag{A-9}$$

multiplying by the denominator

$$\Delta = K_1(a_2+\Delta)(a_3+a_4\Delta + a_5\Delta^2) + K_2(a_1+\Delta)(a_3+a_4\Delta + a_5\Delta^2)$$
$$+ (K_3\Delta + K_4)(a_1+\Delta)(a_2+\Delta) \tag{A-10}$$

Letting $\Delta = -a_1$ we see that

$$K_1 = \frac{-a_1}{(a_2 - a_1)(a_3 - a_4\, a_1 + a_5\, a_1^2)} \tag{A-11}$$

and letting $\Delta = -a_2$

$$K_2 = \frac{-a_2}{(a_1 - a_2)(a_3 - a_4\, a_2 + a_5\, a_2^2)} \qquad \text{(A-12)}$$

We will solve for K_3 and K_4 by equating powers of

$$\begin{aligned}\Delta = {} & K_1(a_2\, a_3 + (a_3 + a_2\, a_4)\Delta + (a_4 + a_2\, a_5)\Delta^2 + a_5\Delta^3) \\ & + K_2(a_1\, a_3 + (a_3 + a_1\, a_4)\Delta + (a_4 + a_1\, a_5)\Delta^2 + a_5\Delta^3) \\ & + K_3(a_1\, a_2\Delta + (a_1 + a_2)\Delta^2 + \Delta^3) \\ & + K_4(a_1\, a_2 + (a_1 + a_2)\Delta + \Delta^2) \qquad \text{(A-13)}\end{aligned}$$

Summing the constants

$$0 = K_1\, a_2\, a_3 + K_2\, a_1\, a_3 + K_4\, a_1\, a_2$$

$$K_4 = -K_1 \frac{a_3}{a_1} - K_2 \frac{a_3}{a_2} \qquad \text{(A-14)}$$

where K_1 and K_2 are known. Summing the coefficients of Δ^3

$$0 = K_1\, a_5 + K_2\, a_5 + K_3$$

$$K_3 = -(K_1 + K_2)\, a_5 \qquad \text{(A-15)}$$

Thus returning to our original integral we may break it into four separate integrals

$$\begin{aligned}\int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta = A_1 \Bigg[& K_1 \int_0^{s-q} \frac{d\Delta}{a_1 + \Delta} + K_2 \int_0^{s-q} \frac{d\Delta}{a_2 + \Delta} \\ & + K_3 \int_0^{s-q} \frac{\Delta\, d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} + K_4 \int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} \Bigg]\end{aligned}$$

(A-16)

Working with each integral separately

$$K_1 \int_0^{s-q} \frac{d\Delta}{a_1 + \Delta} = K_1[\ell n\,(a_1 + \Delta)\,]_0^{s-q}$$

$$= K_1\, \ell n\,(\frac{a_1 + s - q}{a_1}) \qquad (A\text{-}17)$$

and

$$K_2 \int_0^{s-q} \frac{d\Delta}{a_2 + \Delta} = K_2\, \ell n\,(\frac{a_2 + s - q}{a_2}) \qquad (A\text{-}18)$$

and

$$K_3 \int_0^{s-q} \frac{\Delta d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} = K_3[\frac{1}{2\,a_5}\,\ell n\,(a_3 + a_4\Delta + a_5\Delta^2)]_0^{s-q}$$

$$- \frac{a_4}{2\,a_5}\, K_3 \int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2}$$

$$= \frac{K_3}{2\,a_5}\,\ell n\,(\frac{a_3 + a_4(s\text{-}q) + a_5(s\text{-}q)^2}{a_3})$$

$$- \frac{a_4}{2\,a_5}\, K_3 \int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} \qquad (A\text{-}19)$$

Substituting what we have done so far back into Equation A-16.

$$\int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta = A_1[K_1\, \ell n(\frac{a_1 + s - q}{a_1}) + K_2\, \ell n(\frac{a_2 + s - q}{a_2})$$

$$+ K_3 \frac{1}{2\,a_5}\, \ell n(\frac{a_3 + a_4(s\text{-}q) + a_5(s\text{-}q)^2}{a_3}$$

$$+ (K_4 - K_3 \frac{a_4}{2\,a_5}) \int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2}\,]$$

(A-20)

Examining the remaining integral we see that

$$\int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} = \int_0^{s-q} \frac{d\Delta}{\frac{1}{e^2} + \left(p - \frac{\Delta}{[Q/q]q}\right)^2} \qquad \text{(A-21)}$$

which we have previously solved for different limits in Chapter 2.

Using the results of Equation 2.27 in Chapter 2.

$$\int_0^{s-q} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} = q[Q/q]e\left[\tan^{-1}\left(\frac{e}{q[Q/q]}\Delta - ep\right)\right]_0^{s-q}$$

$$= q[Q/q]e\left[\tan^{-1}\left(\frac{e(s-q)}{q[Q/q]} - ep\right) + \tan^{-1}(ep)\right] \qquad \text{(A-22)}$$

Substituting into Equation 2.75,

$$\int_0^{s-q} H_1(\Delta) f_\Delta(\Delta)\, d\Delta = A_1\left[K_1\, \ell n\left(\frac{a_1 + s - q}{a_1}\right) + K_2\, \ell n\left(\frac{a_2 + s - q}{a_2}\right)\right.$$

$$+ K_3 \frac{1}{2a_5} \ell n\left(\frac{a_3 + a_4(s-q) + a_5(s-q)^2}{a_3}\right)$$

$$\left. + \left(K_4 - K_3\frac{a_4}{2a_5}\right) q[Q/q]e\left[\tan^{-1}\left(\frac{e(s-q) - ep}{q[Q/q]}\right) + \tan^{-1}(ep)\right]\right] \qquad \text{(A-23)}$$

Thus completing the most complex of the these integrals.

The second integral takes the form

$$\int_{s-q}^{s} H_2(\Delta) f_\Delta(\Delta)\, d\Delta = A_2 \int_{s-q}^{s} \frac{K_5\Delta + K_6}{\left(\frac{1}{e^2} + \left(p - \frac{\Delta}{q[Q/q]}\right)^2\right)} d\Delta \qquad \text{(A-24)}$$

where

$$A_2 = \frac{Q}{q^2[Q/q]^2 e[\tan^{-1} e(1 - p) + \tan^{-1}(ep)]} \qquad \text{(A-25)}$$

$$K_5 = \frac{s}{q'(s + q - \frac{q}{2\sigma[Q/q]}(s^2/q^2 + s/q))} - \frac{2\sigma q[Q/q](s - q)}{q' s (2(q[Q/q] - s) + s - q)} \qquad \text{(A-26)}$$

$$K_6 = \frac{(q - 5) s}{q'(s + q - \frac{q}{2\sigma[Q/q]}(s^2/q^2 + s/q))} + \frac{2\sigma q[Q/q](s-q)}{q'(2(q[Q/q]s) + s - q)} \qquad \text{(A-27)}$$

Using the previously defined terms a_3, a_4 and a_5 we can write

$$\int_{s-q}^{s} H_2(\Delta) f_\Delta(\Delta)\, d\Delta = A_2[K_5 \int_{s-q}^{s} \frac{\Delta d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} + K_6 \int_{s-q}^{s} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2}] \qquad \text{(A-28)}$$

which is identical in form to the last two integrals of Equation A-16. Using those results we can write

$$\int_{s-q}^{s} H_2(\Delta) f_\Delta(\Delta)\, d\Delta = A_2[K_5 [\frac{1}{2 a_5} \ell n (a_3 + a_4\Delta + a_5\Delta^2)]_{s-q}^{s} + (K_6 - K_5 \frac{a_4}{2 a_5}) q[Q/q] e[\tan^{-1}(\frac{e}{q[Q/q]}\Delta - ep)]_{s-q}^{s}]$$

$$= A_2[K_5 \frac{1}{2 a_5} \ell n (\frac{a_3 + a_4 s + a_5 s^2}{a_3 + a_4(s-q) + a_5(s-q)^2}) + (K_6 - K_5 \frac{a_4}{2 a_5}) q[Q/q] e[\tan^{-1}(\frac{es}{q[Q/q]} - ep) - \tan^{-1}(\frac{e(s-q)}{q[Q/q]} - ep)]]$$

(A-29)

completing the second integral.

Continuing we will consider the final integral

$$\int_s^{q\lceil Q/q\rceil} H_3(\Delta) f_\Delta(\Delta)\, d\Delta = A_3 \int_s^{q\lceil Q/q\rceil} \frac{\Delta d\Delta}{\left(\Delta + \left(q - \frac{q}{2\sigma\lceil Q/q\rceil}\left(s^2/q^2 + s/q\right)\right)\right)\left(\frac{1}{e^2} + \left(p - \frac{\Delta}{\lceil Q/q\rceil q}\right)^2\right)}$$

(A-30)

where

$$A_3 = \frac{Q/q}{q'\lceil Q/q\rceil^2 e[\tan^{-1} e(1-p) + \tan^{-1} ep]} \qquad \text{(A-31)}$$

defining

$$a_6 = q - \frac{q}{2\sigma\lceil Q/q\rceil}\left(s^2/q^2 + s/q\right) \qquad \text{(A-32)}$$

and using the previously defined a_3, a_4 and a_5 we may write

$$\int_s^{q\lceil Q/q\rceil} H_3(\Delta) f_\Delta(\Delta)\, d\Delta = A_3 \int_s^{q\lceil Q/q\rceil} \frac{\Delta d\Delta}{(\Delta + a_6)\,(a_3 + a_4\Delta + a_5\Delta^2)} \qquad \text{(A-33)}$$

Expanding using partial factions

$$\frac{\Delta}{(\Delta + a_6)\,(a_3 + a_4\Delta + a_5\Delta^2)} = \frac{K_7}{(\Delta + a_6)} + \frac{K_8\,\Delta + K_9}{a_3 + a_4\Delta + a_5\Delta^2} \qquad \text{(A-34)}$$

Multiplying by the denominator

$$\Delta = K_7(a_3 + a_4\Delta + a_5\Delta^2) + (K_8\Delta + K_9)\,(\Delta + a_6) \qquad \text{(A-35)}$$

Letting $\Delta = -a_6$ and solving for K_7

$$K_7 = \frac{-a_6}{a_3 - a_4 a_6 + a_5 a_6^2} \tag{A-36}$$

Rewriting Equation 2.95,

$$\begin{aligned}\Delta = {} & K_7(a_3 + a_4\Delta + a_5\Delta^2) \\ & + K_8(0 + a_6\Delta + \Delta^2) \\ & + K_9(a_6 + \Delta + 0)\end{aligned}$$

Summing the constant terms

$$0 = K_7 a_3 + K_9 a_6$$

or

$$K_9 = -K_7 \frac{a_3}{a_6} \tag{A-37}$$

Summing the Δ terms

$$0 = K_7 a_4 + K_8 a_6 + K_9 - 1$$

or

$$\begin{aligned}K_8 &= \frac{1 - [K_7 a_4 + K_9]}{a_6} \\ &= \frac{1 - [K_7 a_4 - K_7 \frac{a_3}{a_6}]}{a_6} \\ &= \frac{1 + K_7 (\frac{a_3}{a_6} - a_4)}{a_6}\end{aligned} \tag{A-38}$$

In its expanded form we may write

$$\int_s^{q[Q/q]} H_3(\Delta) f_\Delta(\Delta)\, d\Delta = A_3[K_7 \int_s^{q[Q/q]} \frac{d\Delta}{a_6 + \Delta} + K_8 \int_s^{q[Q/q]} \frac{\Delta d\Delta}{a_3 + a_4\Delta + a_5\Delta^2}$$

$$+ K_9 \int_s^{q[Q/q]} \frac{d\Delta}{a_3 + a_4\Delta + a_5\Delta^2} \qquad \text{(A-39)}$$

These three integrals are identical to the last three of Equation A-16 except for constants and integral limits. Using these previous results we may write

$$\int_s^{q[Q/q]} H_3(\Delta) f_\Delta(\Delta)\, d\Delta = A_3[K_7\, [\ell n(a_6 + \Delta)]_s^{q[Q/q]}$$

$$+ K_8[\frac{1}{2\, a_5}\, \ell n(a_3 + a_4\Delta + a_5\Delta^2)]_s^{q[Q/q]}$$

$$+ (K_9 - K_8 \frac{a_4}{2\, a_5})\, q[Q/q]e[\tan^{-1}(\frac{e}{q[Q/q]}\, \Delta - ep)]_s^{q[Q/q]}\ ]$$

$$= A_3[K_7\, \ell n(\frac{a_6 + q[Q/q]}{a_6 + s})$$

$$+ K_8 \frac{1}{2\, a_5}\, \ell n\ \frac{a_3 + a_4\, q[Q/q] + a_5 q^2[Q/q]^2}{a_3 + a_4\, s + a_5\, s^2})$$

$$+ (K_9 - K_8 \frac{a_4}{2\, a_5})\, q[Q/q]e[\tan^{-1}e(1-p) - \tan^{-1}(\frac{es}{q[Q/q]} - ep)]]$$

(A-40)

Thus completing the last integral. A summary of the results and relevant constants can be found in Equations 2. 56 through 2. 76 of Chapter 2.

Appendix B

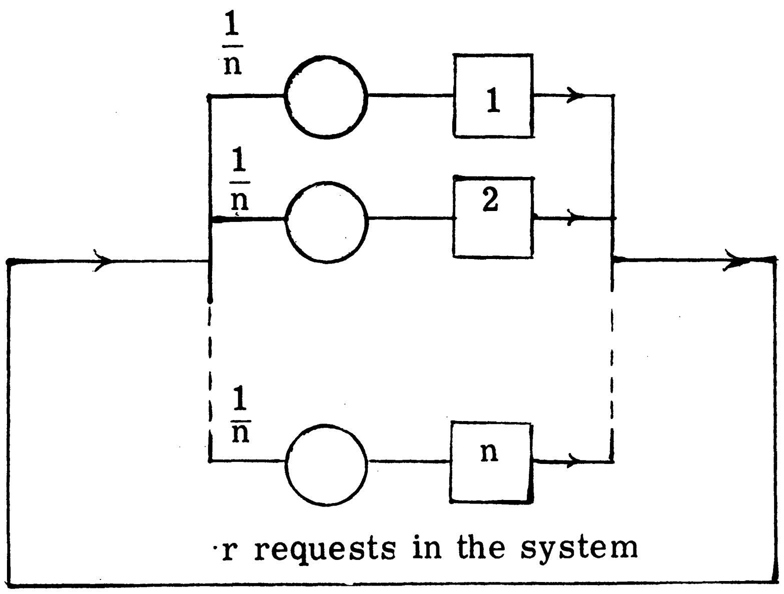
Request Distribution

Our concern here will be to consider how the requests for service distribute themselves among the various devices at a given level. In the case of disks we will want to know the probability that m disk arms are busy when there are n disk arms and r requests in the system. In the case of core boxes, drum sectors and data cells we will ask the same basic question.

We will assume that an arriving request will require the use of a specific device such as a disk arm or core box and that the probability of any specific device being chosen is simply $\frac{1}{n}$ where n is the number of devices in the system.

Consider the diagram of Figure B1. Here we have a queueing model with n queues and servers in parallel where the servers are independent and have the same mean service time. When service completes at any one of the n servers the request immediately re-enters one of the n queue and server combinations. Thus we see that there will always be a fixed number of requests in the system. This corresponds to a system of n devices where it is assumed that we have r requests in the system and each service completion is accompanied by an arrival.

We will want to know the probability of a given distribution of requests in queues and servers after the system has reached a steady



Circulating Queue

Figure B1

state condition.

We will describe the state of the system with an n-tuple

$$r_1, r_2, \dots, r_n \tag{B-1}$$

where r_i is the number of requests in the i-th queue and server combination.

We will define

$$r = \text{total number of requests in the system} = \sum_{i=1}^{n} r_i \tag{B-2}$$

The probability of a transition from any given state $r_1, r_2, \dots, r_n$ to any other state $r'_1, r'_2, \dots, r'_n$, (where other state implies that there exists an i such that $r_i \neq r'_i$) is either 0 or $\frac{1}{n^2}$. In order for a transition to be possible, transition probability $\neq 0$, $r_1, r_2, \dots, r_n$ and $r'_1, r'_2, \dots, r'_n$ must differ in 2 positions.

$$r'_j = r_j - 1 \tag{B-3}$$

$$r'_k = r_k + 1 \qquad k \neq j \tag{B-4}$$

$$r'_i = r_i \qquad j \neq i \neq k \tag{B-5}$$

If the transition is possible the probability of the service completion occurring at server j is $\frac{1}{n}$ and the probability of this circulating request for service entering server or queue k is $\frac{1}{n}$. Since each of these selections is independent the joint probability is simply $\frac{1}{n^2}$. In terms of the state transition matrix we are simply stating that the off diagonal forms are either 0 or $\frac{1}{n^2}$.

It should be clear that if a transition from state $r_1, r_2, \ldots, r_n$ to $r'_1, r'_2, \cdots, r'_n$ is possible then transition from $r'_1, r'_2, \ldots, r'_n$ to $r_1, r_2, \ldots, r_n$ is possible. Thus a non-zero off diagonal term in the transition matrix implies a symmetrical non-zero term. And since all non-zero off diagonal terms are $\frac{1}{n^2}$ the transition matrix must be symmetric and therefore doubly stochastic.

It can be shown (p. 141 of [42] or p. 255 of [43]) that the stationary probabilities resulting from this doubly stochastic state transition are given by

$$\pi_1 = \pi_2 = , \ldots, = \pi_a = \frac{1}{a} \qquad \text{(B-6)}$$

where a is the number of states.

The number of states is the number of distinguishable distributions of r balls in n urns or the number of different solutions to Eq. (B-2). It can be shown (pp. 36-37 of [37]) that

$$a = \binom{n+r-1}{r} \qquad \text{(B-7)}$$

Thus the probability of being in any given state $r_1, r_2, \ldots, r_n$ may be written

$$\pi_i = \binom{n+r-1}{r}^{-1}. \qquad \text{(B-8)}$$

This may be recognized as the classical Bose-Einstins statistics for which many results are available. (See Chapter 2 of [37]) Specific results which may be of use here are]

The number of distinguishable distributions

$$A_{r,n} = \binom{n+r-1}{r} = \binom{n+r-1}{n-1} \qquad \text{(B-9)}$$

The number of distinguishable distributions in which no server remains empty

$$A'_{r,n} = \binom{r-1}{n-1} \qquad \text{(B-10)}$$

The probability that exactly m servers remain empty

$$p'_m(r,n) = \frac{\binom{n}{m}\binom{r-1}{n-m-1}}{\binom{n+r-1}{r}} \qquad \text{(B-11)}$$

and lastly the probability that there are exactly m non-empty servers.

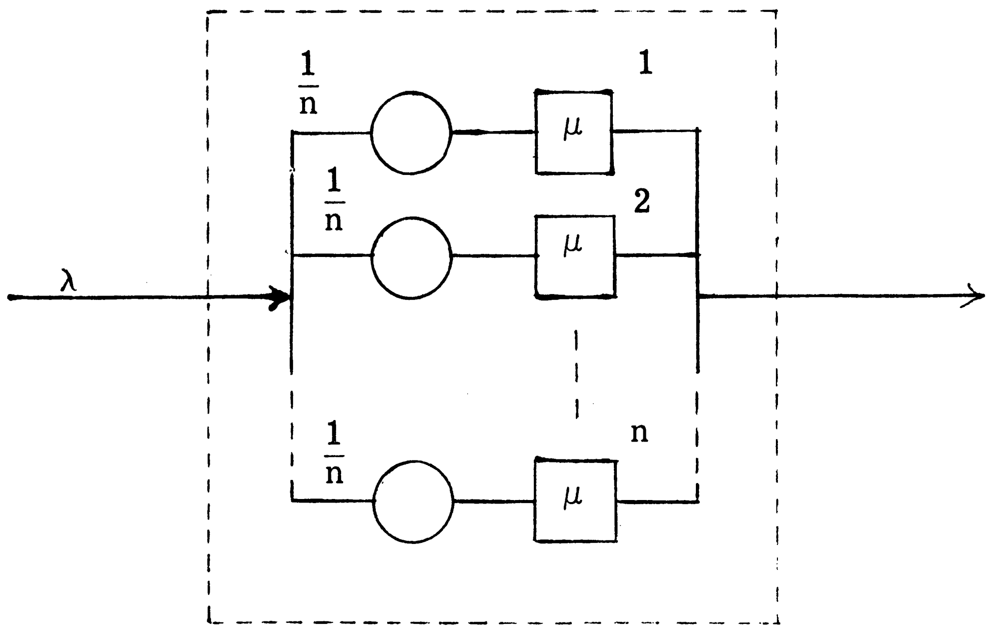
$$p_m(r,n) = \frac{\binom{n}{n-m}\binom{r-1}{m-1}}{\binom{n+r-1}{r}} . \qquad \text{(B-12)}$$

Consider the queueing diagram shown in Figure B2. The arrivals join one of the n queues for service with a probability of $\frac{1}{n}$. The arrival rate is λ and the service rate at each of the n servers is μ. The arrivals and service will be taken as Poisson. We will limit the total number of requests in the n servers and n queues to a total of B requests. Arrivals which would cause the total to exceed B are simply lost. This model differs from the previous model in that exponential service is required but here we will not require that each service completion be accompanied by a corresponding arrival.

We would now like to show again that the steady state probability of being in any state $r_1, r_2, \ldots, r_n$ is equal to the steady state probability of being in any state $r'_1, r'_2, \ldots, r'_n$ whenever $\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} r'_i$. Using the transitions in and out of the state $r_1, r_2, \ldots, r_n$ we will write an equation for the steady state probability of being in this state given the steady state probability of being in the states which have direct transitions to or from the states $r_1, r_2, \ldots, r_n$. We will see that for all states this equation is satisfied by the assumed steady state probability

$$\mathrm{Prob}(r_1, r_2, \ldots, r_n) = \frac{\left(\frac{\lambda}{n\mu}\right)^r}{1 + \left(\frac{\lambda}{n\mu}\right) + \left(\frac{\lambda}{n\mu}\right)^2 + \ldots + \left(\frac{\lambda}{n\mu}\right)^B} \qquad \text{(B-13)}$$

where



Maximum Number in System B

Open Queueing Model

Figure 2B

$$r = \sum_{i=1}^{n} r_i$$

We will write the probability of being in a state with r requests $(\sum_{j=1}^{n} r_i = r)$ at time t as $P_r(t)$ and at time $t + \Delta t$ as $P_r(t + \Delta t)$.

We will first develop an equation for all states in which $r < B$. We must consider three ways of arriving at state $r_1, r_2, \ldots, r_n$ at time $t + \Delta t$. They are :

1. Being in a state with r+1 requests in the system and having a service completion.
2. Being in a state with r-1 requests and having an arrival.
3. Being in the state $r_1, r_2, \ldots, r_n$ and having neither an arrival or service completion.

Beginning with the first of these possibilities we see that there are n states which have r+1 requests from which a service completion can place us in the state $r_1, r_2, \ldots, r_n$. They are :

$$\begin{array}{l} r_1+1,\ r_2,\ \ldots,\ r_n \\ r_1,\ r_2+1,\ \ldots,\ r_n \\ \quad\vdots \\ r_1,\ r_2,\ \ldots,\ r_n+1 \end{array}$$

Since the service rate at each server is μ we may write the incomplete equation

$$P_r(t + \Delta t) = n \mu P_{r+1}(t) \Delta t + \_\_\_\_\_\_ + \_\_\_\_\_\_$$

where we have indicated 2 remaining terms.

The second way to arrive at the state $r_1, r_2, \ldots, r_n$ is to be in one of the states:

$$\begin{array}{l} r_1 - 1, r_2, \ldots, r_n \\ r_1, r_2 - 1, \ldots, r_n \\ \vdots \\ r_1, r_2, \ldots, r_{n-1} \end{array}$$

which does not have a negative term. We will assume for the moment the state $r_1, r_2, \ldots, r_n$ will produce K such states with r-1 requests. Since the arrival rate is λ and there are K states each with a state probability of $P_{r-1}(t)$ we may add the term $K \frac{\lambda}{n} P_{r-1}(t)$ to our equation giving us

$$P_r(t + \Delta t) = n \mu P_{r+1}(t) \Delta t + K \lambda P_{r-1}(t) \Delta t + \_\_\_\_\_\_\_\_$$

The final term in the equation is simply the probability of being in the state $r_1, r_2, \ldots, r_n$ at time t and renaming these during the period Δt or $[1 - (\lambda + K\mu)\Delta t] P_r(t)$. Here the probability of not leaving states $r_1, r_2, \ldots, r_n$ is expressed as one minus the probability of leaving. The complete equation is

$$P_r(t + \Delta t) = n \mu P_{r+1}(t) \Delta t + K \frac{\lambda}{n} P_{r-1}(t) \Delta t + [1 - (\lambda + K\mu)\Delta t] P_r(t) \tag{B-14}$$

Rewriting and equation to 0 for the steady state solution

$$\frac{P_r(t + \Delta t) - P_r(t)}{\Delta t} = n \mu P_{r+1}(t) + K \frac{\lambda}{n} P_{r-1}(t)$$

$$- (\lambda + K\mu) P_r(t)$$

$$= 0 \qquad \text{(B-15)}$$

Solving for P_r

$$P_r = \frac{n \mu P_{r+1} + K \frac{\lambda}{n} P_{r-1}}{\lambda + K\mu} \qquad r < B \qquad \text{(B-16)}$$

In the case of r = B we may write

$$P_B(t + \Delta t) = K \frac{\lambda}{n} P_{B-1}(t) \Delta t [1 - K \mu \Delta t] P_B(t) \qquad \text{(B-17)}$$

which becomes

$$P_B = \frac{K \frac{\lambda}{n} P_{B-1}}{K \mu}$$

$$= \frac{\lambda}{n \mu} P_{B-1} \qquad \text{(B-18)}$$

From Equation B-13 we see that

$$P_{r-1} = \frac{n \mu}{\lambda} P_r \qquad \text{(B-19)}$$

and

$$P_{r+1} = \frac{\lambda}{n \mu} P_r \qquad \text{(B-20)}$$

substituting in Equation B-16 for $r < B$ we have

$$P_r = \frac{\lambda + K\mu}{\lambda + K\mu} P_r = P_r \qquad \text{(B-21)}$$

and Equation for $r = B$

$$P_B = \frac{\lambda}{n\mu}\frac{n\mu}{\lambda}P_B$$

$$= P_B \qquad \text{(B-22)}$$

Thus we see that Equation B-13 is the steady state solution to the model of Figure B2 and that the probability of being in any state $r_1, r_2, \ldots, r_n$ is equal to the probability of being in state $r'_1, r'_2, \ldots, r'_n$ given $\sum_{i=1}^{n} r_i = \sum_{i=1}^{n} r'_i$. Thus again showing that if it is given that there are r requests in the system the probability of being in any given state $r_1, r_2, \ldots, r_n$ where $\sum_{i=1}^{n} r_i = r$ is $\binom{n+r-1}{r}^{-1}$.

A Note on Computation

It will be necessary to compute numerous values of $P_m(r, n)$ for values of r and m. In order to simplify calculation we may express $P_m(r, n)$ as:

$$P_m(r, n) = \frac{n!(r-1)!r!(n-1)!}{(n+r-1)!(n-m)!m!(m-1)!(r-m)!} \tag{B-23}$$

By simply substituting m+1 for m we may write

$$P_{m+1}(r, n) = \frac{n!(r-1)!r!(n-1)!}{(n+r-1)!(n-m)!m!(m-1)!(r-m)!}$$

$$\times \frac{(n-m)(r-m)}{(m+1)m}$$

$$= P_m(r, n) \frac{(n-m)(r-m)}{(m+1)m} \tag{B-24}$$

Similarly we may substitute r+1 for r obtaining

$$P_m(r+1, n) = P_m(r, n) \frac{r(r+1)}{(n+r)(r+1-m)} \tag{B-25}$$

Knowing that

$$P_1(1, n) = 1 \tag{B-26}$$

we may compute all the necessary values of $P_m(r, n)$ rather simply.

Appendix C

Computational Difficulties in Computing β'

The use of Equation 4.17 directly in computation for β' results in possible exponent overflow and underflow conditions. The basic problem lies in the term of Equations 4.17, 4.18, and 4.20 of the form

$$\frac{\lambda^i}{\mu_1\mu_2 \cdots \mu_i}$$

Since i can be as high as several bunched these terms tend to become very small or very large. These problems can be overcome by rearranging terms and special care in taking the summations.

Repeating 4.17, 4:18 and 4.20

$$\lambda' = \lambda + \frac{\left(P(\lambda) + \frac{\lambda^B}{\mu_1 u_2 \cdots \mu_B}\right)\left[C\left(P(\lambda) + \frac{\lambda^B}{\mu_1\mu_2 \cdots \mu_B}\right) - \lambda P(\lambda)\right]}{P(\lambda)\left[P(\lambda) - \frac{\lambda^B}{\mu_1\mu_2 \cdots \mu_B}(B-1) + \lambda P'(\lambda)\frac{\lambda^B}{\mu_1\mu_2 \cdots \mu_B}\right.} \qquad \text{(C-1)}$$

$$P(\lambda) = 1 + \frac{\lambda}{\mu_1} + \frac{\lambda^2}{\mu_1\mu_2} + \ldots + \frac{\lambda^{B-1}}{\mu_1\mu_2 \cdots \mu_{B-1}} \qquad \text{(C-2)}$$

$$\lambda P'(\lambda) = \frac{\lambda}{\mu_1} + 2\frac{\lambda^2}{\mu_1\mu_2} + 3\frac{\lambda^3}{\mu_1\mu_2\mu_3} + \ldots + (B-1)\frac{\lambda^{B-1}}{\mu_1\mu_2 \cdots \mu_{B-1}} \qquad \text{(C-3)}$$

We will now rewrite Equation (C-1) by dividing both numerator and denominator by $P^2(\lambda)$.

$$\lambda' = \lambda + \frac{\left(1 + \left\{\dfrac{\dfrac{\lambda^B}{\mu_1\mu_2\cdots\mu_B}}{P(\lambda)}\right\}\right)\left[C\left(1 + \left\{\dfrac{\dfrac{\lambda^B}{\mu_1\mu_2\cdots\mu_B}}{P(\lambda)}\right\}\right) - \lambda\right]}{\left[1 - \left\{\dfrac{\dfrac{\lambda^B}{\mu_1\mu_2\cdots\mu_B}}{P(\lambda)}\right\}(B-1)\right] + \left\{\dfrac{\lambda P'(\lambda)}{P(\lambda)}\right\}\left\{\dfrac{\dfrac{\lambda^B}{\mu_1\mu_2\cdots\mu_B}}{P(\lambda)}\right\}} \tag{C-4}$$

We will now consider the behavior of the terms in brackets. Beginning with

$$\frac{\lambda P'(\lambda)}{P(\lambda)} = \frac{\dfrac{\lambda}{\mu_1} + 2\dfrac{\lambda^2}{\mu_1\mu_2} + 3\dfrac{\lambda^3}{\mu_1\mu_2\mu_3} + \ldots + (B-1)\dfrac{\lambda^{B-1}}{\mu_1\mu_2\cdots\mu_{B-1}}}{1 + \dfrac{\lambda}{\mu_1} + \dfrac{\lambda^2}{\mu_1\mu_2} + \dfrac{\lambda^3}{\mu_1\mu_2\mu_3} + \ldots + \dfrac{\lambda^{B-1}}{\mu_1\mu_2\cdots\mu_{B-1}}} \tag{C-5}$$

Multiplying numerator and denominator by: $\dfrac{\mu_1\mu_2\cdots\mu_B}{\lambda^B}$

$$\frac{\lambda P'(\lambda)}{P(\lambda)} = \frac{\dfrac{\mu_2\cdots\mu_B}{\lambda^B} + 2\dfrac{\mu_3\cdots\mu_B}{\lambda^{B-2}} + \ldots + (B-1)\dfrac{\mu_B}{\lambda}}{\dfrac{\mu_1\cdots\mu_B}{\lambda^B} + \dfrac{\mu_2\cdots\mu_B}{\lambda^{B-1}} + \dfrac{\mu_3\cdots\mu_B}{\lambda^{B-2}} + \ldots + \dfrac{\mu_B}{\lambda}} \tag{C-6}$$

We will sum the terms of the numerator and denominator from left to right. At each step of the summation we will generate and maintain a term of the form $\frac{\mu_{i+1} \cdots \mu_B}{\lambda^{B-i}}$. This term will be generated by multiplying the previous term by the appropriate $\frac{\mu_{i+1}}{\lambda}$. Note that since the μ_i's are nonincreasing with decreasing index i. The terms $\frac{\mu_{i+1}}{\lambda}$ will decrease as the sum proceeds. Thus if the complete term, $\frac{\mu_{i+1} \cdots \mu_B}{\lambda^{B-i}}$ in the denominator or $i \frac{\mu_{i+1} \cdots \mu_B}{\lambda^{B-i}}$ in the numerator begins to grow smaller with decreasing i, it will continue to grow smaller with decreasing i. Thus if the complete term becomes much much less than current sum further terms may be ignored. The value of $\frac{\lambda P'(\lambda)}{P(\lambda)}$ will vary from $\frac{\lambda}{\mu_1}$ when λ is very small to (B-1) when $\frac{\mu}{\mu_B}$ is large. The summation proposed here will be successful for values of $\frac{\lambda}{\mu_1}$ which are near 1 or large and will fail when $\frac{\lambda}{\mu_1}$ is small and B is large. This is in effect because we are taxing a number larger than $(\frac{\mu}{\lambda})$ and raising it to the power B which can be as large as 300. The important point however is that $\frac{\lambda P'(\lambda)}{P(\lambda)} \approx \frac{\lambda}{\mu_1}$ when this occurs.

Turning to the second term in brackets of Equation C-4 we may write:

$$\frac{\dfrac{\lambda^B}{\mu_1 \cdots \mu_B}}{P(\lambda)} = \frac{\dfrac{\lambda^B}{\mu_1 \cdots \mu_B}}{1 + \dfrac{\lambda}{\mu_1} + \dfrac{\lambda^2}{\mu_1 \mu_2} + \ldots + \dfrac{\lambda^{B-1}}{\mu_1 \cdots \mu_{B-1}}}$$

$$= \frac{1}{\dfrac{\mu_1 \cdots \mu_B}{\lambda^B} + \dfrac{\mu_2 \cdots \mu_B}{\lambda^{B-1}} + \dfrac{\mu_3 \cdots \mu_B}{\lambda^{B-2}} + \ldots + \dfrac{\mu_B}{\lambda}}$$

(C-7)

We see that the denominator here is identical to the previous case and numerator is a constant. Here if we can complete the summation, either by summing all the terms or eliminating them because of their small size, and have a valid result the summations for $\frac{\lambda P'(\lambda)}{P(\lambda)}$ should do likewise (note summation for $\lambda P'(\lambda)$ will be the first to fail). If on the other hand $\frac{\lambda}{\mu_1}$ is small and the summation grows too rapidly it is clear that the term

$$\frac{\dfrac{\lambda^B}{\mu_1 \cdots \mu_B}}{P(\lambda)} \quad \text{is very near 0}$$

as is $\left\{\dfrac{\lambda P'(\lambda)}{P(\lambda)}\right\} \left\{\dfrac{\dfrac{\lambda^B}{\mu_1 \cdots \mu_B}}{P(\lambda)}\right\}$.

Examining Equation C-4 it is clear that these terms may be considered 0 everywhere they occur and Equation C-4 becomes simply

$$\lambda' = \lambda + \frac{C - \lambda}{1} \tag{C-8}$$

or

$$\lambda' = C \tag{C-9}$$

Thus, if we are unable to perform the necessary summations do to excessively large terms, the result being sought is given by Equation C-9.

BIBLIOGRAPHY

[1] Abd-Alla, A.M., Analysis of the Storage Organization for a Multiprogrammed Computer System, Dept. of Electrical Engr., University of Maryland, College Park, Maryland, (Sept. 1969).

[2] Anacker, W. and C.P. Wong, "Performance Evaluation of Computing Systems With Memory Hierarchies", IEEE Trans. on Elec. Comp. 16, 6 (Dec. 1967), pp. 765-773.

[3] Arden, B.W., "Time Sharing Systems: a review", Michigan Summer Conference on Computer and Program Organization, 1967.

[4] Arden, B.W., et.al., "Program and Addressing Structure in a Time-Sharing Environment", Journal of the ACM 13, 1, (Jan. 1966), pp. 1 - 16.

[5] Belady, L.A., "A Study of Replacement Algorithms for a Virtual-Storage Computer", IBM Systems Journal 5, 2, 1966, pp. 78-101.

[6] Belady, L.A., and C.J. Kuehner, "Dynamic Space-Sharing in Computer Systems", Communications of the ACM 12, 5, (May 1969), pp. 282-288.

[7] Belady, L.A., et. al., "An Anomaly in Space-Time Characteristics of Certain Programs Running in a Paging Machine", Communications of the ACM 12, 6, (June 1969).

[8] Bryan, G.E., "Joss: 20,000 Hours at a Console - a Statistical Summary", AFIP Conference Proceedings 31, (Fall 1967).

[9] Buchholz, W., "A Selected Bibliography on Computer System Performance Evaluation", Computer Group News, (March 1969), pp. 21-22.

[10] Burnside, W.S. and A.W. Panton, The Theory of Equations 1, Dover Press, New York, 1960.

[11] Calingaert, P., "System Performance Evaluation: Survey and Appraisal", Communications of the ACM 10, 1, 1967, pp. 12-18.

[12] Chandy, K.M. ,and C.V. Ramamoorthy, "Optimization of Information Storage Systems", Information and Control 13, 6, (Dec. 1968).

[13] Coffman, E.G., "Analysis of a Drum Input/Output Queue Under Scheduled Operation in a Paged Computer System", Journal of the Assoc. of Computing Machinery 16,1, (Jan. 1969) pp. 73-90.

[14] Coffman , E.G. and L.C. Variam, "Further Experimental Data on the Behavior of Programs in a Paging Environment", Communications of the ACM 11, 7, (July 1968), pp. 471-474.

[15] Conti, C.J., "Concepts for Buffer Storage", Computer Group News 12, 8, (March 1969), pp. 9-13.

[16] Conti, C.J., et. al., "Structural Aspects of the System 360 Model 85, I General Organization", IBM Systems Journal 7, 1, 1968, pp. 2-14.

[17] Cox, D.R. and Walter L. Smith, Queues , John Wiley & Sons, Inc., New York, 1961.

[18] Denning, P.J., "Thrashing and Its Cause and Prevention", Proc. AFIPS FJCC 33, 1968, pp. 915-922.

[19] Denning, P.J., "Virtual Memory", Computing Surveys 2, 3, (Sept. 1970), pp. 153-189.

[20] Denning, P.J., "Effects of Scheduling on File Memory Operations", AFIPS Conference Proceedings 31 (Spring 1967).

[21] Denning, P.J., "The Working Set Model for Program Behavior", Communications of the ACM 11, 5, (May 1968), pp. 323-333.

[22] Feller, William, An Introduction to Probability Theory and Its Applications I, John Wiley & Sons, Inc., New York, 1950.

[23] Fenichel, R.R. and A.J. Grossman, "An Analytic Model of Multiprogrammed Computing", Proc. AFIPS SJCC 34, 1969, 1. 717.

[24] Fife, D.W. and J.L. Smith, "Transmission Capacity of Disk Storage Systems with Concurrent Arm Positioning", IEEE Tran on Elec. Comp. 14, 4, (Aug. 1965), pp. 575-582.

[25] Fisher, T.O. and C.D. Shepard, "Time Sharing On a Computer With a Small Memory", Communications of the ACM 10, 2, (Feb. 1967), pp. 77-81.

[26] Freibergs, I.F., "The Dynamic Behavior of Programs", Proc. AFIPS FJCC 33, 1968, pp. 1163-1138.

[27] Hellerman, H., "On the Average Speed of a Multiple Module Storage System", IEEE Tran. on Elec. Comp. 15, 4, (Aug. 1966), pp. 534-550.

[28] Hobbs, L.C., "Present and Future State of the Art in Computer Memories", IEEE Tran. on Elec. Comp. 15, 4, (Aug. 1966), pp. 534-550.

[29] International Business Machines Corporation, "IBM System/360 Fortran IV Language", IBM System Reference Library, file no. S360 25, Form C28-6515-4, IBM, White Plains, New York.

[30] International Business Machines Corporation, "Functional Characteristics of System/360 Model 67", Form A27-2719, IBM, White Plains, N.Y.

[31] International Business Machines, "System/360 Component Descriptions DASD for 2841", Form A26-5988, IBM, White Plains, N.Y.

[32] International Business Machines Corporation, "Analysis of Some Queueing Models in Real-Time Systems", IBM Data Processing Techniques, Form F20-0007-1, IBM, White Plains, New York.

[33] Jewell, William S., A Simple Proof of : $L = \lambda W$, University of California, Berkeley, Calif., 1966.

[34] Kuck, D.J. and D.H. Lawrie, The Use and Performance of Memory Hierarchies: A Survey, University of Illinois, Urbana, Illinois, (Dec. 1969).

[35] Lee, F.F., "Study of 'Look-Aside' Memory", IEEE C-18, 11, (Nov. 1969), pp. 1062-1064.

[36] Lehman, M.M. and J.L. Rosenfeld, "Performance of a Simulated Multiprogramming System", Proc. AFIPS FJCC 33, 1968, pp. 1431-1442.

[37] Liptag, J.S., "Structural Aspects of the System/360 Model 85, II The Cache", IBM Systems Journal 7, 1, 1968, pp. 15-21.

[38] Martin, James, Design of Real-Time Computer Systems, Prentice-Hall, Englewood Cliffs, N.J., 1967.

[39] University of Michigan Computing Center, The MTS Manual 2 Volumes, (Dec. 1967).

[40] Neilson, N.R., "The Simulation of Time-Sharing Systems", Communications of the ACM 10, 7, 1967, pp. 397-412.

[41] Opperheimer, G. and N. Weizer, "Resource Management for a Medium Scale Time Sharing Operating System", Communications of the ACM 11, 5, (May 1968), p. 313.

[42] Parzen, Emanuel, Modern Probability Theory and Its Applications, John Wiley & Sons, New York, 1960.

[43] Parzen, Emanuel, Stochastic Processes, Holden-Day, San Francisco, 1952.

[44] Pinkerton, Tad B., "Performance Monitoring in a Time-Sharing System", Communications of the ACM 12, 11, (Nov. 1969), pp. 608-610.

[45] Pinkerton, T., Program Behavior and Control in Virtual Storage Computer Systems, University of Michigan, CONCOMP Report 4,(April 1968).

[46] Pirtle, Mel, "Intercommunication of Processors and Memory", Proc. AFIPS FJCC 31, 1967.

[47] Ramamoorthy, C.V. and K.M. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems", Journal of the Assoc. for Computing Machinery 17, 3, (July 1970), pp. 426-445.

[48] Randell, B. and C.J. Kuehmer, "Dynamic Storage Allocation Systems", Communications of the ACM 11, 5, (May 1968).

[49] Randel, B., "A Note on Storage Fragmentation and Program Segmentation", Communications of the ACM 12, 7, (July 1969), pp. 365-370.

[50] Rehmann, S.L. and S.G. Gangwere Jr., "A Simulation Study of Resource Management in a Time-Sharing System", Proc. AFIPS FJCC 33, 1968, pp. 1411-1430.

[51] Saltzer, J.H. and J.W. Gintell, "The Instrumentation of Multics", Communications of the ACM 13, 8, (Aug. 1970), pp. 495-500.

[52] Shemer, J.E. and G.A. Shippey, "Statistical Analysis of Paged and Segmented Computer Systems", IEEE Tran. on Elec. Comp. 15, 6, (Dec. 1966), pp. 855-863.

[53] Sisson, S.S. and M. Flynn, "Addressing Patterns and Memory Handling Algorithms", Proc. AFIPS FJCC 33, 2, 1968, pp. 957-967.

[54] Smith, J.L., "Multiprogramming Under a Page on Demand Strategy", Communications of the ACM 10, 10 (Oct. 1967), pp. 636-646.

[55] Varian, L.C. and E.G. Coffman, "An Empirical Study of the Behavior of Programs in a Paging Environment", Communications of the ACM 11, 7, (July 1968), pp. 471-474.

[56] Wallace V.L. and D.L. Mason, "Degree of Multiprogramming in Page on Demand Systems", Communications of the ACM 12, 6, (June 1969), p. 305.

[57] Wallace, V.L. and R.S. Rosenberg, RQA-1 The Recursive Queue Analyzer, Systems Engineering Lab, University of Michigan, Ann Arbor, Mich., (Feb. 1966).

[58] Weizer, N. and G. Oppenheimer, "Virtual Memory Management in a Paging Environment", Proc. AFIPS FJCC 34, 1969, p. 249.

[59] Wilkes, M.V., "Slave Memories and Dynamic Storage Allocation", IEEE Tran. on Elec. Comp. 14, 2, (April 1965), pp. 270-271.

[60] Wilkes, M.V., "A Model for Core Allocation in a Time-Sharing System", Proc. AFIPS SJCC 34, 1969, p. 265.

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY *(Corporate author)* Dept of Electrical Engineering Systems Engineering Laboratory University of Michigan	2a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED 2b. GROUP

3. REPORT TITLE

ANALYSIS AND OPTIMIZATION OF MULTIPROGRAMMED COMPUTER SYSTEMS USING STORAGE HIERARCHIES

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Interim report

5. AUTHOR(S) *(First name, middle initial, last name)*

Ashby M. Woolf

6. REPORT DATE August 1971	7a. TOTAL NO. OF PAGES 320	7b. NO. OF REFS 60

8a. CONTRACT OR GRANT NO. F30602-69-C-0214	9a. ORIGINATOR'S REPORT NUMBER(S) SEL Tech. Report No. 53
b. Job Order No. 55010000	
c.	9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)*
d.	RADC-TR-71-165

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES RADC Project Engineer Rocco F. Iuorno AC 315/330-7011	12. SPONSORING MILITARY ACTIVITY Rome Air Development Center (ISIS) Griffiss Air Force Base, New York 13440

13. ABSTRACT

The research described in this report centers around the development and application of a general and comprehensive mathematical model of computer systems which use storage hierarchies consisting of 2, 3 or more levels and in which the storage management is carried out automatically (i.e. transparent to the user.) This model has been implemented in the form of a highly interactive computer program which provides the user with an animated view of the system performance as model parameters are varied. There are approximately 50 independent variables in the model. These variables describe the system architecture, data paths and routing, storage device characteristics, CPU performance, and user program behavior.

UNCLASSIFIED
Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
MULTIPROGRAMMED SYSTEMS STORAGE HIERARCHIES MATHEMATICAL MODELING						

UNCLASSIFIED
Security Classification