1-1-1978

A feasibility study of a linear zero-one memory allocator.

Paul Kodjo Niamkey

A FEASIBILITY STUDY OF A LINEAR ZERO-ONE MEMORY ALLOCATOR

by

Paul Kodjo Niamkey

A Thesis

Presented to the Graduate Faculty

of Lehigh University

in Candidacy for the Degree of

Master of Science

in

Industrial Engineering

Lehigh University

1978

This thesis is accepted and approved in partial fulfillment of the requirement for the degree of Master of Science.

1-27-78
(date)

RDJ
2-4-78

Professor in Charge

Chairman of Department

ACKNOWLEDGEMENTS

Most of all, I must thank Dr. Ben L. Wechsler, my advisor, for his continuing confidence in me, and for his assistance, inspiration and guidance through all the various stages of this research and of Master's study at Lehigh University.

Especial thanks go to Dr. Gary E. Whitehouse for his encouragements and his many discerning comments and research suggestions.

I express my appreciation for the excellent secretarial services provided by Mrs. Jerilou Newhall.

I am particularly indebted to the African American Institute for the opportunities it has provided for financial assistance, making it possible for me to embark on and complete my Master's studies, and to enter the career of my choice.

Through it all, the patient understanding of my parents, Jean Niamkey and Louise Chicaya, has eased the burden of my educational experience in America.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

The allocation of limited non-paged main memory among users of a computer system is investigated as a zero-one cost/priority problem in a free enterprise environment. The work is divided into three phases.

The first phase is the investigation of the jobs' parameters likely to influence scheduling decisions in a critical way.

The second phase is to define and study the properties of the major constraints of the zero-one machine used to arrive at optimum scheduling decisions. The process is regarded as the basic memory demand entity. Its system characteristic is a pair (net-pay, memory demand). Net-pay represents the net returns the scheduling of the process will bring to the computer system. Main memory demand represents the process' immediate memory requirement (intensity and duration).

Processes are segregated into two distinct classes at the scheduling time. The first class, the wait class, is temporarily denied execution. The second class or ready class is granted the use of central memory for the next planning interval. The ready class contains those processes whose system characteristics are solutions of the zero-one problem, and which result in optimal revenue to the computer

system.

Simulation of the behavior of the zero-one linear scheduler and comparative analysis of schedules derived by the zero-one scheduler, the best fit and first fit schedulers, constitute the basis of the work in the third phase.

This work is intended to introduce a relatively new approach to modelling scheduling operations. It is intended to spark an economic/value way of thinking about computer storage allocation.

I. INTRODUCTION

A. Background

The extended capabilities of today's computer systems have increased the need to control the sequence of jobs which are processed by the computer. Complex operating systems have been designed to that effect. Memory management plays a very important role in the design of those systems and has proven to be a major bottleneck [5]. In today's computer systems, programmable memory amounts to one third of the systems' costs; hence, great care must be exercised in the design of executable memory allocators. The displacement of serial processing systems by sophisticated multiprogramming systems is outstanding evidence of the importance of memory resources utilization.

1. Multiprogramming

Multiprogramming in the simplest sense means that more than one process* can be executing within the same computing system and at the same time; those processes are referred to as concurrent processes. In the early serial processing single allocation systems, only one job could

---

* A process or job is the smallest unit of work that can be presented to the computing system by the user. It is defined through the Job Control Language (JCL).

be run at the same time. Note that the definition of multiprogramming does not mean that simultaneous operations are possible; parallel processing can take place only where there is the possibility of simultaneous execution of more than one instruction as for example in multiprocessing systems. In multiprogramming, concurrent processes can, however, be in different states of execution as they alternate in their use of the instruction processor. In typical schemes, the central processor unit (CPU) time is allocated to each job on a time-slice priority basis. For each "quantum of time", the operating system causes the CPU to execute a program in its address-space until one of the following conditions occurs:

- -job is terminated;
- -error is detected;
- -program requests I/O operations;
- -quantum of time expires.

The job is then purged from memory (first two cases) or is temporarily suspended; the processor is automatically assigned to the next job with the highest priority. The rationale behind multiprogramming systems is that a computer system cannot perform efficiently in a serial processing scheme because of the rather large disparity between the fast speed of the CPU and the slow activity rate of the I/O devices. The larger that disparity, the more concurrent processes should be executing in order to minimize the total computing system wait time*. Although many attempts have been made to improve to a maximum the efficiency of I/O processing (example: VS2 Release 2 Operating System developed by IBM), it is indisputable that much more needs to be accomplished in that direction before multiprogramming is considered obsolete.

The advantages of multiprogramming are much more obvious in situations wherein the job mix of processes accessing the system is well balanced, i.e. when CPU bound jobs are evenly mixed with I/O bound jobs**. Maximum system utilization is achieved when the job mix is perfect, i.e. a job is always ready to use the CPU when it becomes available.

2. The problem of fragmentation.

There exists three states a job may assume once it gains access to the computer system: Active, Ready, and Wait. A job is said to be active when it is currently using the CPU. If the job is ready to use the CPU and cannot because of the execution of another process, the job

---

* Wait time is defined as the time during which the system is not utilizing fully its processing resources.

** CPU bound jobs make heavy use of the CPU and little use of I/O devices; I/O bound jobs are the exact opposite.

is said to be in a ready state. A job temporarily suspended and waiting for the completion of some activity (Input/Output, operator's intervention ....) is said to be in a wait state.

A well balanced situation will find the system filled with sufficient jobs so that the probability is high some jobs will always be in a ready state.

a. Storage fragmentation.

Jobs in a ready state are normally resident in core memory. In partioned memory allocation, the ready jobs residence status creates the important problem of fragmentation. Fragmentation is the development of unusable "holes" or fragments in memory and it has a statistical nature. It is due to the fact that memory is allocated in arbitrary sized segments which are in turn returned in an essentially random order. Fragmentation degrades the performance of the system by decreasing CPU utilization and system throughput whenever the total interspersed free space is sufficient to honor a request but cannot be found in a single segment. A number of different approaches can be used to tackle the problem, some of which will be assessed later in this chapter. Before this is done, we shall consider the different kinds of fragmentation encountered in multiprogramming systems.

a.1. Internal fragmentation

When memory is subdivided into fixed size partitions, and when every job is required to use one or more blocks of fixed size, the resulting fragmentation, if any, is called internal fragmentation. As shown in figure 1.1a, after a period of time, the core memory configuration changes into that of figure 1.1b. The resulting free space of 60 units cannot accomodate a new arriving process requesting 30 units because of internal fragmentation.

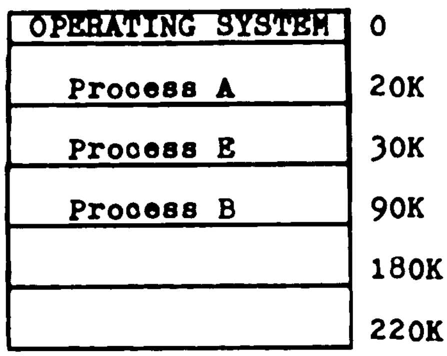


figure 1.1a

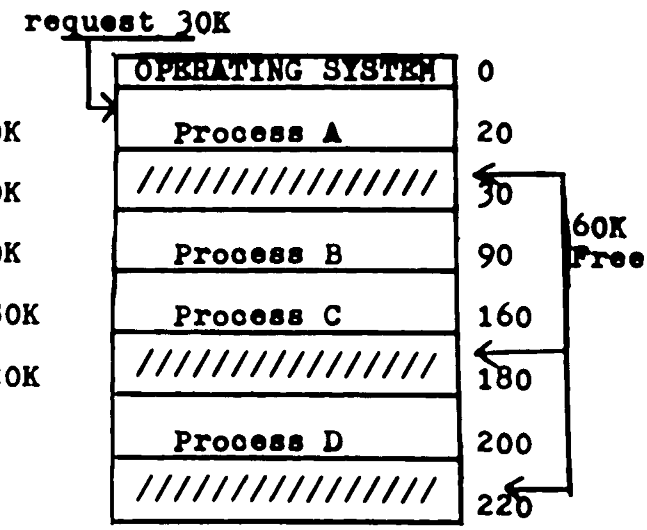


figure 1.1b

(1K = 1024 WORDS)

a.2. External fragmentation

If internal fragmentation becomes too severe, storage allocation can be made dynamic and memory partitions can be made to fit the process space request. After a period of time, a checkerboard pattern of allocated spaces interspersed with available spaces causes a loss of usable memory. This type of fragmentation is said to be external (figures 1.2a and 1.2b). Efforts to eliminate internal fragmentation have now introduced a different kind of memory waste.

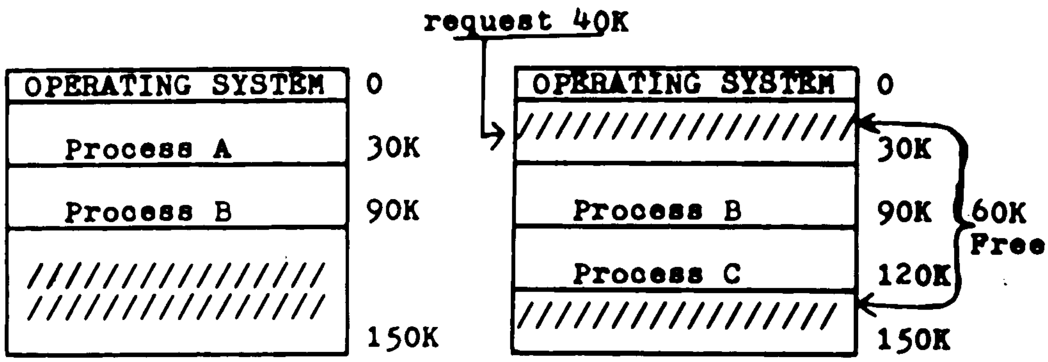


figure 1.2a figure 1.2b

b. <u>Measurement of fragmentation.</u>

Randell[14] and Shore[19] have developed some very relevant methods for measuring storage fragmentation. The conclusion of Randell's simulations was that internal fragmentation can rapidly exceed any saving in external fragmentation as the rounding size increases. A measure that would be a direct function of the free storage distribution would probably not provide a reasonable measure of performance since the usefulness of the free storage depends on future segments requests. For example, two requests of size 90 units on two free blocks of size

100 units create very little fragmentation, whereas if the next request is of size 110 units, the memory is highly fragmented. Randell suggests that given a set of n segments requests, fragmentation be measured as the ratio of the time taken to allocate requests on a totally compacted memory over the time taken to satisfy the same requests with the strategy in use. This is expressed mathematically as:

$$R_x(n) = \frac{T_c(n)}{T_x(n)}$$

$R_x(n)$ is always less than one and, in general, the greater its value, the better the strategy X.

Shore suggested two different measures of performance; they are the time memory product efficiency, E, and the storage utilization fraction, U. If n requests $r_i, i = 1, 2, \ldots\ldots, n$, are allocated for times t_i, on a memory of size M during a total elapsed time T, then the efficiency is defined by Shore as:

$$E = \frac{1}{MT} * \sum_{i=1}^{n} r_i t_i$$

If $\{r_i(t)\}$ is the set of requests that happen to be resident in memory at time t, then $U_t = (1/r) * \sum_i r_i(t)$ and U is the average of U_t over time.

Those three measures of performance are very useful for statistics collection by simulators and the time memory product efficiency measure will be used in the simulations presented in chapter III of this thesis.

3. Some solutions to the fragmentation problems.

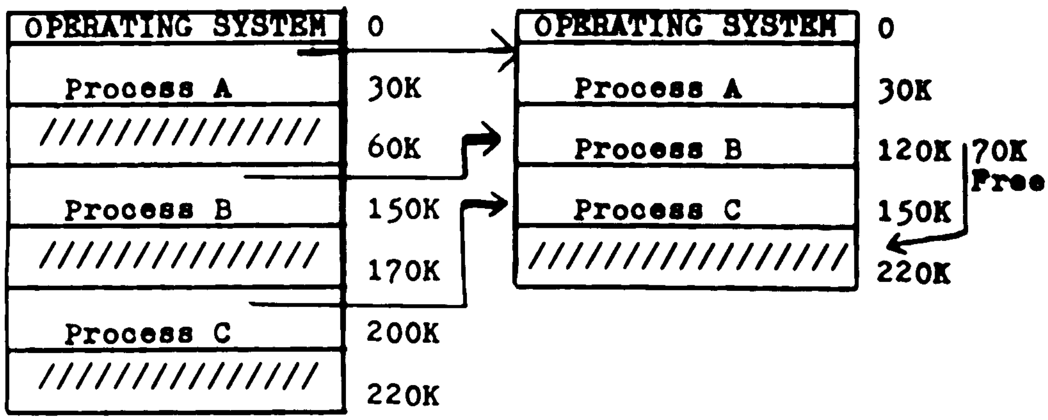
There are two kinds of solutions that exist today; they are: memory compaction and dynamic memory management systems.

a. Memory compaction.

Memory compaction is a treatment of external fragmentation and is just one form of garbage collection*. This technique is a simple straightforward approach which lets fragmentation take place and then deals with it when it becomes a problem. The technique is particularly used with list processing languages such as LISP which have the peculiarity of requesting and releasing large amounts of memory in an unpredictable way. The compaction algorithm uses in fact a very simple scheme. Each request for space is satisfied by allocating a block or a partition following the most recently allocated block and no attempt is made to reuse any memory that may have been released. When appropriate, the algorithm performs memory compaction. All

* Garbage collection refers to any technique which makes unused memory areas available for use.

allocated spaces are moved to one end of the memory in a contiguous area and the available space is collected at the other end of memory. (figure 1.3). Decisions concerning when and how to perform compaction are in general critical and very decisive. Some systems elect to perform compaction whenever the CPU is idle, e.g. waiting for input, in order to make the most efficient use of the control processor.



(1K = 1024 words)

Snapshot of memory before GARBAGE COLLECTION

Snapshot of memory after GARBAGE COLLECTION

figure 1.3

The compaction process is, however, very costly. On the average, even with special hardware implementing Move instructions, it costs one or more memory cycles (0.5 or more microseconds depending on the computer) to move each word when compaction is required. Moreover, the compaction overhead is increased by the requirements that all address constants that refer to any segment that happened to be moved be updated. This updating costs additional execution time and requires extra space in order to keep a record of every constant and the segment to which it refers. The costs are particularly higher in multiprogramming systems where updating problems exist even without compaction, in deciding which of the segments of an executing process should be kept in primary memory, which to move out of memory and when to move them. These high costs have made the compaction process very undesirable and has prompted the design of allocation algorithms which do not require compaction. These algorithms are classified as dynamic memory management algorithms and shall now be investigated.

b. Dynamic Memory Management Systems.

Because garbage collection is limited to external fragmentation only, and because the process itself is very costly, it is recommended that it be used only when unavoidable. A "better" solution to the fragmentation problem seems to be the prevention of fragmentation and the reduction of the frequency of compaction. The allocation algorithms will then increase in complexity but, through careful management, external fragmentation can be considerably reduced; internal fragmentation, paradoxically, becomes the problem. Two very important algorithms will be described shortly; they are referred to as Best Fit and First Fit strategies in the computer field.

b.1. <u>Best Fit and First Fit strategies.</u>

The Best Fit algorithm searches through the entire list of available memory areas and allocates the smallest area of sufficient size to satisfy the request. The algorithm allocates only the amount requested and returns the leftover space unless it is too small to be of any use, in which case the entire block is allocated.

The First Fit algorithm searches through the list until it finds the first available block that is large enough to hold the request. The unused portion is returned to the list of available memory blocks.

Historically, the Best Fit method was widely used for several years; it was thought to be a good policy since it saves the larger available areas for a later time when they might be needed. Unfortunately, scanning the entire list to find the best fit could use an excessive amount of time, which makes the strategy rather slow. Furthermore, Best Fit tends to increase the number of very small blocks and such a proliferation is not very desirable. There are many simple situations where the First Fit method is clearly better than the Best Fit method. As an example, suppose we are given the following list of available blocks: 1200, 1000, and 3000, and our list of requests is 700, 500, 900, and 2200 units. Storage will be allocated in the following way:

request for memory	available areas First Fit	available areas Best Fit
-	1200, 1000, 3000	1200, 1000, 3000
700	500, 1000, 3000	1200, 300, 3000
500	1000, 3000	700, 300, 3000
900	100, 3000	700, 300, 2100
2200	100, 800	-----stuck------

There are, however, some simple instances where Best Fit outperforms First Fit: example:

request for memory	available areas First Fit	available areas Best Fit
-	200, 150, 100, 50	200, 150, 100, 50
150	50, 150, 100, 50	200, 100, 50
100	50, 50, 100, 50	200, 50
150	---stuck---	50, 50

In general, any system that offers its largest block first to satisfy a requirement which is followed by exact duplicates of the requests sizes will be better handled by Best Fit. Extensive simulations of both strategies have been conducted and First Fit was found to be more efficient than Best Fit under general operating conditions. It has nevertheless been shown [19] that Best Fit would outperform First Fit whenever the distribution of requests is an exponential distribution which has been truncated. The point of all the complexity of the above two algorithms is the avoidance of memory compaction. Since compaction is required when no free block is large enough to satisfy the current request, the smaller the blocks are, the more likely compaction will be needed. This fact creates one of the worst disadvantages of Best Fit when compared to First Fit; as it was pointed out earlier, Best Fit tends to multiply the number of very small blocks whereas, First Fit tends to do exactly the reverse. By cleverly combining contiguous free blocks and by using a cyclic search, that is always starting the search after the free block from which the previous allocation was taken, the First Fit algorithm tends to distribute small blocks more uniformly through memory. This uniform distribution increases the probability that when a small block is released it will be combined with a larger block; simulation studies [14] have shown that for some classes of segments, these features are so effective that if the need for compaction arises, the total amount of free space available will not be large enough to satisfy any request. Therefore, this approach virtually eliminates the need for compaction.

A final word on those two strategies is that the allocators may be given the ability to take into account knowledge of the statistics of the requests sizes and memory residence distributions. They could then conduct their own look-ahead simulations. But at such a level of sophistication, the overhead would be very high.

b.2 Other dynamic memory management systems.

The two strategies presented above are the most widely used core allocation methods to date. However, other systems have been developed and call for some attention.

- The Buddy System.

In this system, memory is always allocated in sizes which are a power of two. The idea of the method is to keep separate lists of available blocks each of size 2^k words, the entire memory consisting of 2^m words. Originally, the entire 2^m words of memory are available. When a block of size 2^k is desired, and if nothing of that size is available, a larger available block is split into two equal parts, each of size being a power of two. These blocks are called buddies. If both buddies are available, they coalesce into a single larger block. The key fact underlying the usefulness of the method is that the location of a block's buddy is easy to compute given the address of the block and its size. This is because each block size is a power of two and division can be done by register shifting instead of by using any division instruction which would be too slow. The possibility of the simultaneous availability of several blocks of the same size requires the inclusion of a link pointer within each block of a given size. That link will point to the next available block of the same size. This linking system evidently increases the overhead of the system.

The Buddy system suffers external fragmentation when free blocks of the same size cannot be combined into a larger block because they are not buddies. Furthermore, if requests are not a power of two, they are rounded up to the nearest power of two and the result may very well be internal fragmentation. Even though the elimination of fragmentation is not totally achieved, the Buddy system minimizes memory waste by satisfying requests as much as possible and by its ability to split large blocks of memory and coalesce buddies.

- Fibonacci Memory Management System.

The Fibonacci system which is in fact a generalization of the Buddy system, creates blocks of size S_n where S_n is a generalized Fibonacci sequence:

$$S = 0; \quad S_1 = S_2 = S_3 = \ldots\ldots\ldots\ldots = S_{k+1} = 1$$
$$S_n = S_{n-1} + S_{n-k-1}$$

for some integer k. When k = 0, the Buddy system is realized with blocks of size $S_n = 2^n$.

The central problem with generalized Fibonacci systems is not in allocating, but in locating adjacent buddies subject to coalesce into free blocks. This problem drives the overhead of the system to a high level since numerous buddy counters must be included in the systems programs realizing the algorithm.

- Modified First Fit.

Modified First Fit is similar to First Fit except that it uses a cyclic search. We have seen earlier that this particular search tends to distribute small blocks more uniformly through memory. This uniform distribution increases the probability of combining free small blocks with adjacent larger ones, thus making it virtually needless to provide for compaction.

- Half Fit.

This algorithm searches for a segment that is approximately twice the size of the segment request. If the search fails, the algorithm changes to First Fit. Simulations [9] have shown that this strategy performs rather successfully when there is a strong bias to segments of a given size.

B. Statement of the Problem - Objective.

The performance of a computer system executing in a multiprogramming environment is very much limited by the capacity of its executable memory and the flexibility of the scheduling algorithm monitoring core memory allocation. The core allocation strategies used in today's systems were designed with the objective of reducing fragmentation to a minimum; they have unfortunately proven to be relatively unsuccessful in dealing adequately with other aspects of scheduling. Worse, they do not allow the computer system users to interact with their jobs and be able to influence scheduling decisions. It is true that operating systems should free the users from having to know details of the computer internal processing; on the other hand, in most computing environments, since billing is not directly controlled by the users, there is no economic incentive for a user to request only that amount of core time resource actually needed to execute his or her programs.

The objective of this thesis is the development of a storage allocation algorithm which will use an optimization machine to arrive at scheduling decisions. Users' interaction with the scheduling algorithm will be permissible to the extent that scheduling decisions will in fact be made by the users in an indirect way. In the new algorithm's environment, fragmentation will not always be a problem and will not be considered as a performance measure. The maximum amount of money every user will be willing to pay in order to have a share of storage resources and the optimum returns that scheduling will provide to the computer system will be the determining factors for scheduling.

The chapters to follow will examine in more depth the design of the algorithm. Chapter III is directed toward establishing the feasibility of the new approach to scheduling. The reader should realize that this research presents an original concept which offers many opportunities for additional exploration.

II. DESCRIPTION OF THE ALGORITHM

We have discussed in the preceding chapter the complex nature of storage scheduling decisions that have to be made within a computer system. Regardless whether a systematic scheduling process is followed or not, the schedules do get made. These schedules may be prepared by default using a very simple procedure like first-come, first-served or by manipulation of some complex existing procedures such as Best Fit, First Fit, Modified First Fit, Half Fit that try to minimize the amount of storage left unused. However, none of the central memory pricing systems associated with these different algorithms reflect the value the user attributes to the information obtained through access to the system. Nonetheless, the need for a cost/priority system has been discussed by Nielsen [13], whereas Marchand [12] has introduced a utility function model applicable to time sharing systems in which a linear combination of individual utility functions has to be maximized.

In this chapter, we will devise and develop a system for allocation of executable memory storage in a multiprogramming environment which will provide a cost of storage subjective to the individual's value for information obtained. In this system, the data processing center performs as a profit organization whose excess returns are channeled back into the company or corporation maintaining the system.

A. Justification of a Scheduling System Based On Economics.

The allocation is accomplished by considering executable memory as an economic good. In the system, execution will be denied to those users whose subjective proposed prices are below a fixed minimum price based on the cost of maintenance. The system will be described in the following section. This section is directed toward describing the concept of a price based memory scheduling system.

A computer system is similar to an economic system in the sense that it must solve the problem of how to use and distribute scarce resources and goods between customers. The scarcity of finished goods forces the economic system to be closed since any goods consumed by a customer reduces the consumption possibilities of all others. A computer system is analogous to a closed system, insofar as computer resources are concerned.

Moreover, a price system is a mechanism by which determination can be made of the preferences of different economic units for the same economic resources or products. It establishes a priority of users and also a priority of wants and it is generally designed to convey sufficient information in order to determine the flow of resources among different allocations over time, while optimally distributing the goods and services among competing consumers. Prices are not a mechanism for recovering cost; they are a rationing device and as such they are allowed to fall below or rise above cost in order to convey the proper information on the behavior of the consumers and of the market. However, in the system presented here, prices will never be allowed to fall below cost; if they were, the system would very rapidly degrade and would always be in imbalance. This particular situation will be assessed in more detail in the penultimate section of this chapter.

B. The Model.

The system consists of a single processor or CPU operating in a multiprogramming environment on an executable memory of size S words. The system contains two queues of infinite capacity. It is multiprogrammed under a variable number of tasks (MVT). The total core capacity of the machine is distributed into any size partitions dynamically. The core is generally allocated to each program according to its specific requirements. Storage is therefore allocated in units of one word so that no more than the requested amount is ever allocated. Dynamic partitioning in general makes the scheduling function much more complex than multiprogramming partitioning under a fixed number of tasks (MFT). Under MFT, the total core of the machine is semipermanently allocated into fixed size partitions. For this system, MVT has been chosen instead of MFT, because MVT tends to make better utilization of the total core available.

The algorithm also utilizes a static sequencing method. At the time of preparing the storage allocation and schedule, information on the number of jobs that need processing, i.e. jobs in the wait queue will be available. The schedule and pricing for the next planning period is then prepared assuming that these jobs are the only jobs that will be processed during the planning period. In fact jobs will keep arriving throughout the scheduling period, but static sequencing offers a means to plan allocation and schedule with the information already known. It is also less complex and easier to achieve than a sequencing that will schedule the jobs dynamically.

Storage is thus priced and scheduled only for the next planning period, also referred to as next operating interval. Let T be the duration of the operating interval. T is in effect the multiprogramming turnaround time for the jobstream present in the wait queue at the control period*,

---

* The control period is the point in time where scheduling decisions are made for the next operating interval. It is a point in time in the current operating interval.

and represents also the turnaround time of the jobstream if it were processed serially. Consequently if t_j is the execution time limit requested by process j, then the maximum value of T can be determined as the summation of t_j's over all processes in the wait queue at the control period.

Prices fixed at the end of the control period will remain constant throughout the operating interval, and core residency is guaranteed to any program for the length of time necessary to complete execution. This says in effect that swapping is not possible. Swapping, if permitted, would make the algorithm much more sophisticated but would complicate the pricing system to a great degree. The one shot central memory residency requirement of this thesis may introduce some imbalance in the system in cases where a job is killed*, dropped* or temporarily suspended** by the system operator. Section D of this chapter will provide more details on that point.

Sequencing is performed through a zero-one linear machine whose objective is the maximization of the returns to the system during the planning period considered at the current control period. The zero-one machine is invoked at

---

* Drop, Kill: premature termination of a job due to an operator Drop or Kill command (Scope operating system).

** Rerun: termination of a job due to an operator Rerun command (Scope operating system). (See Appendix for further details) each control period and determines optimum storage allocation, optimum sequencing and storage pricing. In fact, storage is allocated to a process at the price the owner of that process values the information he or she obtains through the service of the system, provided that that price is above a minimum cost of maintenance and overhead referred to as min-pay in this thesis and noted by c_j.

Let B_j be the pay-will* of process j and c_j its min-pay. For reasons of simplicity B_j and c_j will be expressed in dollars ($). B_j represents in fact the purchasing power of the process and cannot exceed the amount of money available under the login account of the job. In the cases where it does, the job will be denied execution by the algorithm scheduler. The determination of c_j will be assessed later in this section.

Not all processes present in the wait queue at the control period will be selected for execution during the next planning period. The processes which will be selected will be placed in the second queue of the system, the ready queue. The wait queue will then only contain those processes which were rejected; it will eventually comprise the processes accessing the system after the control period and

* Pay-will is the maximum price the owner of the process is willing or able to pay to share storage.

during the operating interval. The order of the jobs in the wait queue is immaterial and does not affect the scheduling decisions of the algorithm. The relevant parameters influencing the decisions are the process's B_j, its execution field length s_j and its execution time t_j. t_j is expressed in systems seconds*. The min-pay c_j is determined as the cost of the memory time product requirement of job j. If C represents the cost per unit of memory time product, then c_j will be $C*s_j*t_j$. The parameter C is expressed in $/word/second and is independent of the users. It is calculated and fixed by the administration of the computer center and represents the cost of memory maintenance. C should be changed periodically in order to reflect the stochastic short-run fluctuations of hardware and software maintenance requirements.

The algorithm will issue periodically a report status to the user community and will accept any changes made by the users in the B_j of the processes they created.

C. Zero-one Programming Approach.

1. Variables definition.

Structuring the problem as a zero-one problem requires

* System seconds are accounting units which combine a central memory factor, CP time and channels usages, according to a formula predefined by the center.

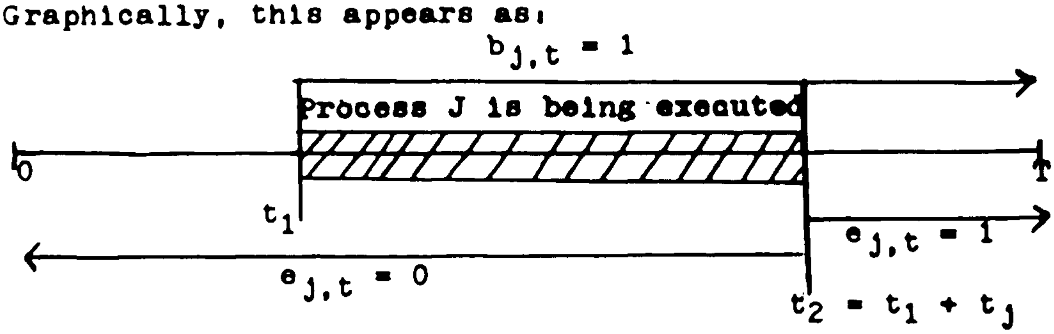
that we define variables and establish appropriate relationships between them to reflect the problem constraints. We let j represent the number of the job and t represent the time. Let us define two step functions for each job. We will say about job j:

$$b_{j,t} = \begin{cases} 1 \text{ if job } j \text{ is begun by period } t \\ 0 \text{ otherwise.} \end{cases}$$

and also:

$$e_{j,t} = \begin{cases} 1 \text{ if job } j \text{ is completed by the beginning of period } t \\ 0 \text{ otherwise.} \end{cases}$$

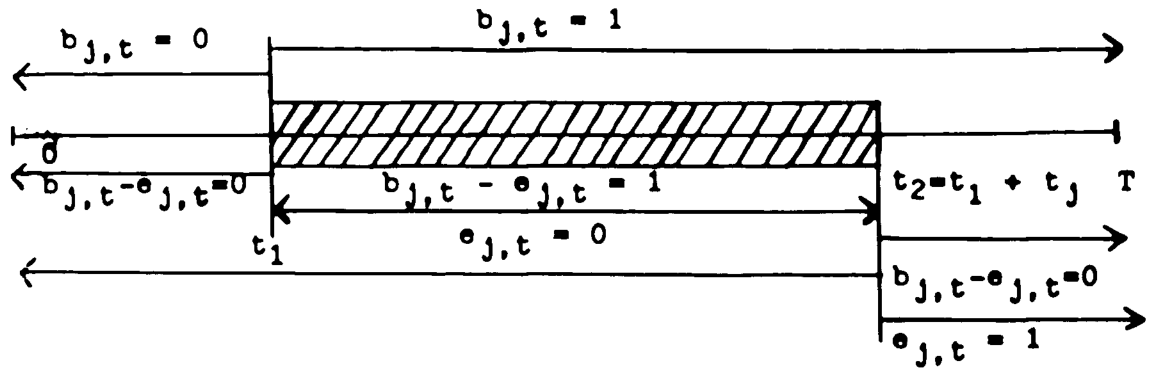
Graphically, this appears as:



for a job of execution time limit of t_j systems seconds. With this definition, the following is true about any job j in any period t:

$$b_{j,t} - e_{j,t} = \begin{cases} 1 \text{ if job } j \text{ is being processed during } t \\ 0 \text{ otherwise.} \end{cases}$$

Graphically, this gives:



We can guarantee algebraically that variables $b_{j,t}$ and $e_{j,t}$ have the required step characteristics by the following inequalities:

$$b_{j,t} \leq b_{j,t+\Delta t} \quad \text{or} \quad b_{j,t} - b_{j,t+\Delta t} \leq 0 \qquad (0)$$

for all j and for $t = \Delta t, 2\Delta t, \ldots\ldots, T - \Delta t, T$

and

$$e_{j,t} - e_{j,t+\Delta t} \leq 0 \text{ for all j and for} \qquad (1)$$

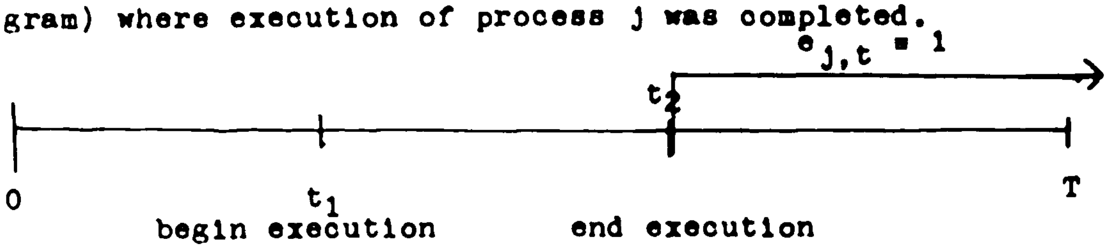
$t = \Delta t, 2\Delta t, \ldots\ldots, T - \Delta T, T$

where T represents the operating interval length and Δt represents a small increment of time used to render the continuous aspect of the problem. The length of Δt is left to the discretion of the computing center and is dependent upon the accuracy required of the algorithm. An upper bound for Δt would be job mix dependent and would be equal to the shortest execution time request present in the wait queue at the control period. The Scheduler could be designed and programmed to check that the upper bound on Δt is not violated and to make the necessary adjustments if required. The requirement that once a job i is begun, processing continues until termination is algebraically translated into:

$$(1') \quad b_{i,t} = e_{i,t+t_i} \quad \text{for all } t \text{ and } i.$$

Equation (1') enables us to eliminate variables $b_{j,t}$ and suggests that the zero-one formulation can be obtained with variables $e_{j,t}$ only.

Moreover, by definition of $e_{j,t}$, it is clear that $e_{j,t} \equiv 1$ for all time t coming after time t_2 (on the diagram) where execution of process j was completed.



In particular, $e_{j,T}$ will be equal to 1 if process j completed execution during the operating interval*. Because it is impossible to predict the job mix characteristics, it is likely that some processes will begin execution during T but will complete execution some time after T. For those processes, $e_{j,T} = 0$. Although such

* Operating Interval: also referred to as control interval in the sequel.

processes will contribute in the returns to the system during T, the zero-one machine will consider only the greater majority of processes with $e_{j,T} = 1$. This approach is used in order to simplify the algorithm. More will be said about this situation (called overlap crisis) in section E of this chapter.

2. Resources constraints.

At the control period the scheduler must determine the number of processes waiting for acess to memory. This number will be the number of jobs resident in the wait queue of the system. We will denote that number by the letter q. Since we are considering a system operating in a multiprogramming environment, adequate utilization of memory capacity will be achieved if at least one job is in process at any time period t during the planning interval T. We can therefore formulate the second constraints as:

$$\sum_{j=1}^{q} (b_{j,t} - e_{j,t}) \geq 1 \text{ for } t = \Delta t, 2\Delta t, \ldots\ldots, T-\Delta t, T$$

With the definition of $b_{j,t}$ and $e_{j,t}$, it is true that:

$$b_{j,t} - e_{j,t} = \begin{cases} 1 & \text{if job } j \text{ is being processed during } t \\ 0 & \text{otherwise.} \end{cases}$$

Since equation (1'): $b_{j,t} = e_{j,t} + t_j$ for all t and j, suggests that variables $b_{j,t}$ can be replaced without any loss of generality by $e_{j,t} + t_j$, the reformulation of the second constraints can be restated as:

$$\sum_{j=1}^{q} (e_{j,t} + t_j - e_{j,t}) \geq 1 \text{ for } t = \Delta t, 2\Delta t, \ldots\ldots, T - \Delta T, T \qquad (2)$$

where t_j represents the execution time of process j.

The factor $b_{j,t} - e_{j,t} + t_j$ is 0 or 1 and indicates whether or not process j is in execution at time period t. If s_j denotes the central memory field length allotted to process j, then the product $(e_{j,t} + t_j - e_{j,t})*s_j$ will represent the memory area occupied by process j at time t.

Since central memory is limited and is of size S, it is clearly evident that memory space utilized by multi-programmed jobs at any time period t cannot exceed S. This is algebraically expressed in the form of the third constraints as:

$$0 \leq \sum_{j=1}^{q} (e_{j,t} + t_j - e_{j,t})*s_j \leq S \text{ for } t = \Delta t, 2\Delta t, \ldots\ldots, T \qquad (3)$$

q represents the number of jobs present in the wait queue at the control period. Although not all the jobs in the wait queue will be active in the next operating time interval, the summation over all q is nonetheless utilized in constraints (3). The zero-one machine will perform the selection and hence constraints (3) in fact take only into consideration those processes that will be permitted to reside in central memory during T.

Constraints (3) impose an upper bound on memory space utilized at any time t. The combination of constraints (2) and (3) will force the zero-one machine to schedule at least one program in central memory at any time of the control interval. Constraints (3), however, do allow storage fragmentation. In order to minimize that fragmentation and guarantee that the zero-one machine solution will reflect the multiprogramming aspect of this application, the following constraint (4) must be satisfied. Constraint (4) expresses the concept of maximum efficiency in the multiprogramming environment.

We have described in chapter 1 of this thesis a measure of performance suggested by Shore [19] and called the time memory product efficiency E. If n requests $r_i, i = 1, 2, \ldots, n$ are allocated for times t_i on a memory of size M during a total elapsed time T, then the time memory product efficiency is formulated as:

$$E = \frac{1}{MT} * \sum_{i=1}^{n} r_i * t_i$$

We have argued earlier in this chapter that variable $e_{j,T}$ would be equal to 1 if process j completed execution during the operating time interval T. Moreover, we have decided that the zero-one machine would not consider the future possible returns to the computer center provided by those processes in overlap crisis*. This attitude was adopted in order to avoid the introduction of complex job mix parameters predictions in the algorithm. The time memory product of a non-crisis-process** j will be mathematically expressed as:

$e_{j,T}*s_j*t_j$, where s_j and t_j represent the central memory field length request and the execution time of process j.

Recalling that $e_{j,T} = \begin{cases} 1 & \text{if the process is selected by the zero-one machine} \\ 0 & \text{otherwise.} \end{cases}$

we can now formulate the maximum time product efficiency constraint as:

$$0 \leq \frac{1}{ST} * \left(\sum_{j=1}^{q} e_{j,T} * s_j * t_j \right) \leq 1 \qquad (4)$$

* term used to describe situations where the execution of a job overlaps T.

** a crisis-process is a process caught in overlap crisis.

Constraint (4), combined with the objective function which will be described shortly will guarantee that the zero-one machine will attempt to keep fragmentation to a minimum.

Constraints (1), (2), (3), and (4) are believed to describe the scheduling problem adequately enough. Additional zero-one constraints are however necessary to complete the constraints formulation of the zero-one problem. Zero-one constraints (constraints 5) state that:

$$e_{j,t} = 0,\ 1 \text{ for all } j \text{ and } t \qquad (5)$$

3. Objective function formulation.

The data processing center is considered as a profit making service organization within the corporation. Its objective is to deliver service (information) to a group of users and make an optimum profit sufficient to at least cover the cost of the center. Excess profit will be channeled back to the corporation. Let us assume that it costs the center C $/word/second for maintenance of memory. The scheduling system assigns to each job accessing the system a min-pay equal to the product of C by the central memory field request and by the execution time request. For a process j, the min-pay would be $c_j = C*s_j*t_j$.

The pay-will B_j of process j must exceed or at least be equal to c_j in order for the job to be accepted in either of the queues of the system. This restriction adds other constraints to the zero-one machine. These constraints state that:

$$e_{j,T} * (B_j - C_j) \geq 0 \text{ for all } j \qquad (6)$$

Constraints (6) are in fact implicit and will not appear in the zero-one formulation since processes j such that $B_j < c_j$ are automatically rejected by the system at their entry in the wait queue.

The pay-will B_j of job j will determine its priority within the system job mix. As a means of reflecting the importance of jobs relative to each other, we introduce the parameter denoted as the <u>job priority index</u> and algebraically defined as:

$$\frac{B_j - C_j}{\sum_{j=1}^{q} B_j}$$

for any job present in the queue at the control period. The objective function of this formulation is expressed as:

$$\sum_{j=1}^{q} e_{j,T} * (B_j - C_j)$$

The zero-one machine maximizes that objective function while staying within the boundaries defined by constraints 1 thru 5.

The zero-one machine maximizes the sum of the differences between B_j and c_j subject to the fact that process j be executed before expiration of T, in order to provide every user with a "fair" share of the computer storage resources. An attempt at straightforward maximization of the summation of B_j's for example could be disadvantageous to urgent processes j with small c_j. With a formulation of $\sum_{j=1}^{q} B_j$ as the objective function, the zero-one machine would have the tendency of scheduling those processes which present a large pay-will. Urgent jobs with small min-pay, i.e. jobs not requiring excessive amount of time memory product resource would then have to access the system with a very large pay-will in order to have a chance of being scheduled by the algorithm during T.

The priority index factor selected for this algorithm is thought to guarantee adequate fairness in the share of storage resources. It is implicit in the objective function and indicates that users have a partial control over the position in which they desire their jobs to execute.

4. Summary of the zero-one formulation.

The objective of this algorithm scheduler is to:

Maximize $\sum_{j=1}^{q} e_{j,T} * (B_j - C_j)$ over the next operating interval, subject to the following constraints:

(1) -- $e_{j,t} - e_{j,t+\Delta t} \leq 0$ for all j and for

$t = \Delta t, 2\Delta t, 3\Delta t, \ldots\ldots, T - \Delta t, T$

-- resources constraints:

(2) -- $\sum_{j=1}^{q} (e_{j,t+t_j} - e_{j,t}) \geq 1$ for $t = \Delta t, 2\Delta t,$

$\ldots\ldots, T - \Delta t, T$

(3) -- $0 \leq \sum_{j=1}^{q} (e_{j,t+t_j} - e_{j,t}) * s_j \leq S$ for $t = \Delta t,$

$2\Delta t, \ldots\ldots, T$

(4) -- $0 < \frac{1}{ST} * (\sum_{j=1}^{q} e_{j,T} * s_j * t_j) \leq 1$

-- zero-one constraints:

(5) -- $e_{j,t} = 0, 1$ for all j and $t = \Delta t, \ldots\ldots, T - \Delta t, T$

where j represents a job, B_j its owner's pay-will, c_j its owner's min-pay, t_j its execution time in systems seconds, s_j its execution field length in words of central memory and variable

$$e_{j,t} = \begin{cases} 1 & \text{if job } j \text{ is completed by the beginning of period } t; \\ 0 & \text{otherwise.} \end{cases}$$

This zero-one machine will automatically select and schedule the processes for the next control interval T.

D. Penultimate Section.

1. Users collusion.

We have pointed out in section B of this chapter that prices should be considered not as a mechanism for recovering cost, but rather as a rationing device and thus should be permitted to fall below or rise above cost. These fluctuations would be a reliable source of information on the behavior of the consumers and of the market. However, in this algorithm, prices are not allowed to fall below cost. In fact, constraints (6) state that any user j accessing the system with a pay-will B_j inferior to the associated min-pay c_j, would be denied residency in central memory. This apparent contradiction in the system is necessary for this algorithm to perform efficiently.

Let us consider the case where the user's B_j could be allowed to be less than the user's corresponding min-pay, c_j. The customers of the system will then tend to lower their respective pay-will to the extent that the computer is likely to be operating at loss. Without any minimum level of acceptance for the user's pay-will, each user will probably decide to fix his or her job's pay-will at a common minimum level, 0 for example. This case of users collusion will create a system imbalance and the price system will no more reflect the importance the user attaches to the services provided by the computer system;

such an imbalance would degrade the performance of the algorithm as far as maximization of revenues is concerned. Moreover, there will not be any adequate rule to go about for the scheduling of the programs in central memory since the objective function of the zero-one formulation will appear to be useless. The scheduling system will degenerate.

The restriction of this thesis that any job's pay-will be greater than or at least equal to its min-pay is therefore necessary to guarantee an adequate performance of the algorithm; in addition, that restriction justifies the existence of the data processing center as a profit making service department within the corporation.

2. System imbalance due to central memory one shot residency.

We have assumed earlier in this chapter that central memory residency is guaranteed to any program after it has begun execution. This assumption has led us to conclude that no swapping consideration needed to be included in the design of the algorithm and has therefore simplified the design process in itself. Unfortunately, this limitation may have a degrading effect on the performance of the algorithm. Most computer systems used today are provided with a console and highly interactive capabilities enabling the operator to kill* or drop* a program if necessary. When this occurs, the program is swapped out and the control point** at which it was executing is freed and made available to another eventual process. The same policy is followed whenever a job is suspended for rerun* or terminates abnormally. In the system presented here, such situations will not be handled similarly.

Since scheduling is determined via an optimization algorithm, it is impossible for the system to adjust itself to unforseen situations without the risk of running into a bottleneck. The solution given by the zero-one machine is optimal and represents the equilibrium of the system. Any change to that solution is therefore likely to create a system imbalance unless the change happens to reflect an equivalent solution. Chances for obtaining an equivalent basic solution are very slim and it might be better to adopt a passive stand rather than free the control point whenever a drop, kill, rerun or abnormal termination occurs. The storage space occupied by the program in question will be wasted, but the user will only be charged for the time

---

* Premature termination of a job due to an operator Drop, Rerun or Kill command. (see Appendix)

** Control point area contains information such as the job name, processing time accumulated, related control statements, etc. ..... (Scope Operating System: see Appendix) memory product used up to the time of the condition. This policy is however not reflected in the objective function since it is mathematically impossible to introduce its concept in the zero-one formulation. The frequency distribution of operator drop, kill or rerun due to abnormal situations is very much dependent upon the environment the center is operating in and cannot be generalized. The algorithm described in this chapter is fully valid only under the assumption that all programs submitted to the center will execute to completion without any problem. Further research is needed to encompass real situations such as those described above. Another area requiring further investigation is described below as "overlap crisis".

3. Overlap crisis.

Let t_1 be the time at which process j gains access to memory and $t_2 = t_1 + t_j$ the time at which its execution is completed. Suppose in addition that $t_1 < T < t_2$. For such a process $e_{j,T}$ will be equal to 0. We say that the process is in overlap crisis. As pointed out earlier in this chapter, the zero-one machine will not consider the contributions of processes in overlap crisis.

Because it is impossible to know in advance the job mix characteristics, it is probable that in applications, overlap crisis situations will occur within each scheduling decision. A policy of not considering any crisis process in the basic solution as contributing in the profit to the center will be followed in this thesis in order to simplify the complexity of the algorithm.

It was primarily thought that crisis processes could be considered by the zero-one machine with the introduction of an additional variable $e_{j,T+t_j}$ in the set* $\{e_{j,\Delta t}, e_{j,2\Delta t}, \ldots\ldots, e_{j,T}, t_j, s_j, C_j, B_j\}$ of variables describing the scheduling characteristics of any process j. Because of the step function character of variable $e_{j,t}$, and by definition of $e_{j,t}$, it is clear that $e_{j,T+t_j} = 1$ and $e_{j,t} \equiv 1$ for any time $t \geq t + t_j$ and any process j of execution time t_j. Consequently, variable $e_{j,T+t_j}$ represents a dependable indication of whether or not process j was selected in the basic solution by the zero-one program. Thus, the formulation of the optimum profit generated by the center during the next control interval would seem to be more accurate with the introduction of variable $e_{j,T+t_j}$; on the other hand, maximum efficiency would be impossible to achieve during T and storage space left unused would increase because the scheduler would tend to postpone the scheduling as much as possible. For this

* this set is referred to as the characteristic set in the sequel.

reason, the approach was rejected and the policy described in section B was adopted. Therefore, the characteristic set of any process j contains only the following elements:

$$e_{j,\Delta t},\ e_{j,\ 2\Delta t},\ \ldots\ldots,\ e_{j,T-\Delta t},\ e_{j,T},\ t_j,\ s_j,\ C_j,\ B_j.$$

It is, however, possible to reduce the number of variables involved in the scheduling decisions. We shall now present how this could be done.

4. Simplifications.

Suppose that at the current control period, the jobs present in the wait queue have the following execution time characteristics:

jobs (j):	1	2	3	4	5	6	7	8	9	10
duration: t_j*	05	01	20	25	02	10	10	25	15	05

Let us assume that all jobs are heavy CPU bound; the maximum length of T is automatically determined to be 118 systems seconds ($T = \sum_{j=1}^{10} t_j$). With a Δt of 1 second, which represents in fact the upper bound permissible here, the non-simplified zero-one linear programming formulation will contain a total of 1180 variables (118*10) and 2597 constraints (1180 of type (1), 118 of type (2), 118 of type (3), 1180 of type (5) and 1 of type (4)).

A few variables can be eliminated. For example, no job can be completed in a time period less than its duration. Therefore, for each j, variables $e_{j,t}$, for $t = \Delta t, 2\Delta t, \ldots\ldots, t_j - \Delta t$ can be ignored. Approached in this fashion, the zero-one linear programming problem requires 1072 variables and 2381 constraints distributed in the following manner:

		constraints				
jobs	number of variables	(1)	(2)	(3)	(4)	(5)
1	114					114
2	118					118
3	99					99
4	94					94
5	117					117
6	109					109
7	109					109
8	94					94
9	104					104
10	114					114
total	1072	1072	118	118	1	1072

total number of constraints is: 2381.

With the introduction of the above simplifications, the zero-one linear formulation can be made substantially smaller. This will result in faster scheduling decisions.

5. Macroscheduling.

Because of the competitive nature of the pricing system on which this algorithm is based, it may be difficult for users with limited budget to ever gain access to the computer system central memory. A system of prime, non-prime and off hours costing and users charging could be utilized that will attempt to lower the load on the system at critical periods of the day. It would then be much more advantageous for some users to schedule their utilization of the data processing center during non-prime or off-hours time when the C parameter of the system is much lower.

E. <u>Conclusions and Remarks.</u>

In this chapter, we have presented a conceptual framework for developing an overall zero-one linear storage allocator. The allocator prices storage resource according to the individual user's estimate of the value of service obtained through the computer system. In order to simplify the design of the algorithm, we have had to assume that the job mix was uniform, and that all programs accessing the system would terminate normally. In the next chapter of this thesis, we present simulations of the behavior of the allocator for some jobstream.

We want to point out one final word. The storage scheduling approach followed in this thesis is believed to provide the data center user with a "feel" of what is happening to his or her jobs at the microlevel. Provided with the ability to interact with the scheduler and change his or her processes priority through the modification of the pay-will parameters, the user will be encouraged to take the external scheduling process more seriously and to design his or her jobs very carefully before submission to the data center. In this respect, the whole user's attitude toward data processing center macroscheduling will be changed from that of a relatively passive one to that of a more active and dynamic one. With such encouragement from the users community, the data center macroscheduling could be made easier.

III. SIMULATION MODEL

In a computing system with finite resources and a demand for resources that periodically exceeds capacity, the operating system has to make many policy decisions. Policy decisions will include as many relevant factors as possible. For example, the core allocator algorithm will mostly consider such factors as the amount of memory requested, the amount of memory available, the job priority, the estimated job run time, other outstanding requests and the availability of other requested peripherals. Disagreement arises as to how factors should be weighed and the strategies that are most appropriate for the installation workload. The component of the operating system that decides which jobs should be allowed to compete for the CPU and hence for core storage is the job scheduler.

The first two chapters of this thesis were written with the objective of fully describing the requirements and problems intrinsic to the development and design of a scheduler for any given computing environment. We have discussed the theoretical framework for the design and the description of a new type of scheduler in Chapter II. The scheduler is based on the concept that core memory should be based on the value of the output to the user. We have explained and emphasized how relevant scheduling decisions could be made through the solution of a zero-one linear machine.

This chapter will present the results and analysis of simulation experiments conducted in the course of the design. Before such a presentation can be made, a brief description of the hardware and software used, as well as of the data operated upon is in order.

A. Simulation Environment.

The Lehigh University CDC 6400 computer system is in the environment in which the experiments were conducted. The system operates under the SCOPE 3.4.4 Operating System and consists of one CPU and ten peripheral processors or PP's. The peripheral processors are virtual machines with their own CPU and memory, operating independently of each other and of the main CPU. The PP's may access both central memory and their own 4K of core (K = 1024 in octal). Central memory consists of 120K (octal) 60 bits-words. The operating system supports two concurrent modes of service; batch (local and remote) and time sharing. The system is multiprogrammed up to fifteen jobs may be active at one time. Each active job is said to reside at a control point and may be in one of five stages of execution; executing with the CPU, waiting for some PP activity to complete, waiting for an operator action, or swapped out.

The data used for the study were generated on the CDC 6400 and represent actual resource requests on the system for a normal working day in the university. (appendix)

Part of the software used in the study is the MPOS (Multi-purpose Optimization System) developed by Northwestern University. MPOS is an integrated system of computer programs to solve optimization problems on CDC 6000/CYBER computer systems. Because of its relatively simple structure and repertoire of algorithms, MPOS has been used by many students in several universities across the United States. The system is designed for university uses of small to medium size optimization problems and was a limiting factor in this study. Other commercial mathematical programming systems, such as CDC's APEX, directed at the solution of very large problems stemming from corporate or industry models, were not available. Access to such larger and often faster systems would have provided greater flexibility to this study and would have facilitated the work greatly.

The computational procedure used in the interpretations of the zero-one linear program is Gomory's cutting plane algorithm for the all-integer programming problem. Gomory's algorithm was chosen instead of other zero-one algorithms such as the Branch and Bound Mixed Integer Program, (BBMIP), or the DSZ1IP algorithm, because of its ability to change the boundaries of the solution space without slicing off any of the feasible integer solutions to the original problem. BBMIP and DSZ1IP proceed by enumeration of all possible solutions to the integer problem and were prohibitively expensive in terms of storage space required for their adequate execution.

B. Simulation Procedures.

1. Methodology.

The object of the simulation experiments was not the study of the Lehigh University computer system operating under a zero-one scheduler. The simulation study presented in this chapter was conducted in order to provide visibility on the behavior of the zero-one scheduler. Thus the sample data file provided by the University Computing Center is read and interpreted by a FORTRAN program [23] so as to generate situations whereby the utilization of the zero-one scheduler capabilities becomes a necessity. The FORTRAN program does not take into account the jobs' entry time in the computer system. As the file is read, resource utilization data such as memory requests, CPU time requests, CP time requests and PP time requests are used by the program in the generation of the zero-one linear formula. The program translates the scheduling problem into the zero-one formulation file which is in turn used as input to the MPOS package. MPOS then processes the formula via Gomory's algorithm. MPOS output contains the optimum scheduling decisions and the optimum value of the objective function for the planning horizon considered. The output file is analyzed for determination of which jobs are granted access to memory, and which are not. The jobs j such that $e_{j,T} = 1$ (where T is the length of the operating interval) are released from the system and will not be considered active during the next operating interval. Figure 3 - 1 displays a partial representation of the flow of information within the simulation model.

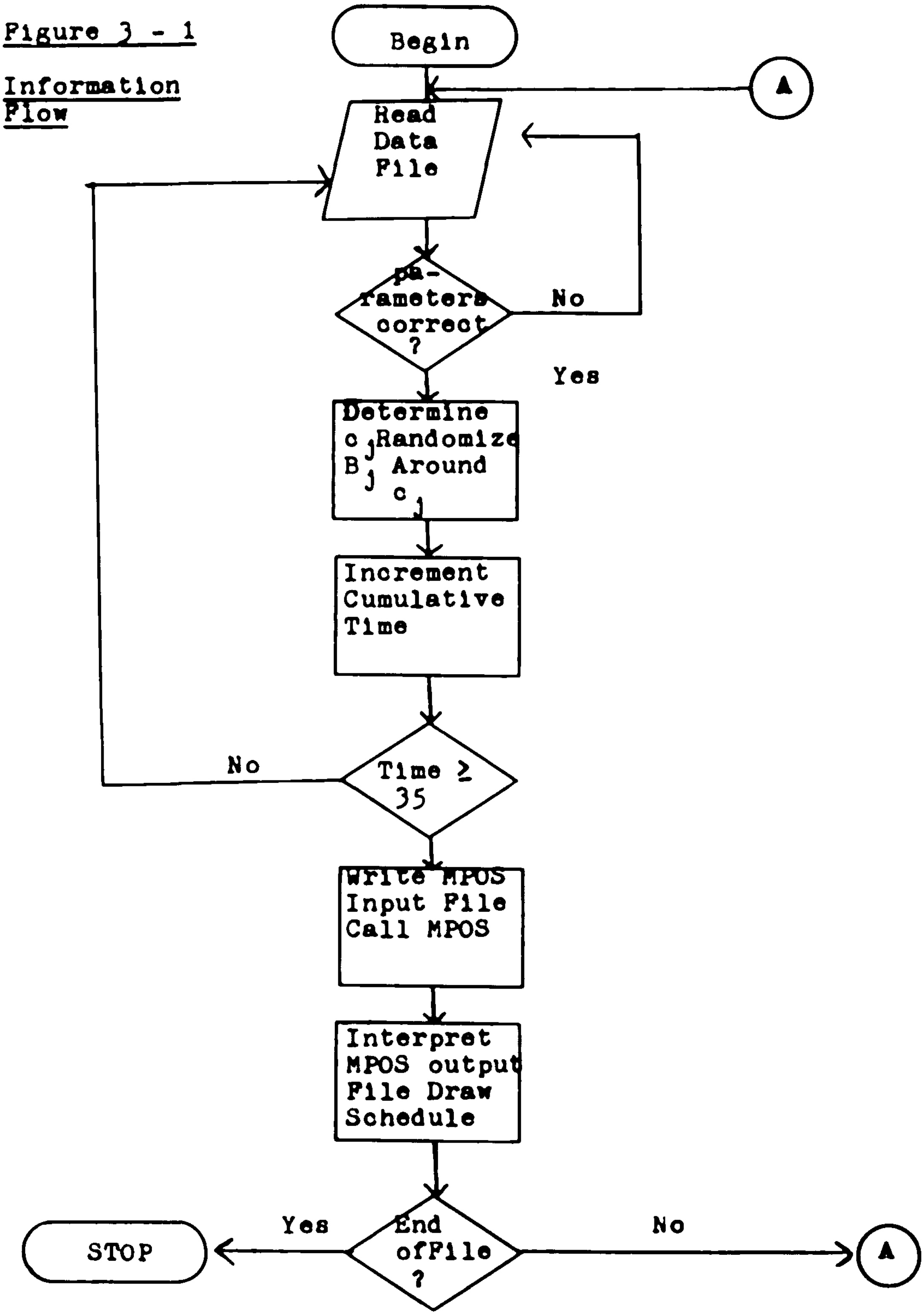
Actual schedules could be determined by further analysis of the MPOS output file. (see "Scheduling Decisions" (section 8) for information on how this was done.)

At the control period, every job in the wait queue is assigned a characteristic variable name.

The execution time limit request of the job and the Δt parameter are used by the FORTRAN program in the determination of the number of elements on the job's characteristic set. In an attempt to simplify the design of the experiments, the Δt parameter was chosen to be 1 system second. The program then determines the c_j and B_j parameters of the jobs present in the queue and finally writes the objective function of the zero-one problem. If after determination of c_j and B_j, of a job j, it is discovered that B_j is less

Figure 3 - 1

Information Flow



than c_j, then job j will be flagged as invalid and will not be considered in the zero-one formulation. In real life situations, an appropriate message could be written in the job's dayfile in order to make the user aware of the job's situation during that particular processing period. The user would then either increase the B_j parameter or resubmit the job in another processing period, or both.

The program finally writes resource constraints (1), (3) and (4) only. The reason for the omission of constraints of type 2 is not readily apparent and will be explained later in this chapter.

2. Variables Table Determination-Jobs' Characteristic Sets.

The MPOS package was used with the standard input. The standard input is the algebraic format where problems are stated in natural mathematical format. Each variable must have a distinct variable name. At the control period, every job in the queue is assigned a variable name A thru Z. Letters E and H are not used because of intrinsic restrictions of the MPOS package and of the Fortran Simulator.

A question frequently asked in the course of this experimental work was: "What is the optimum number of jobs that should be permitted to reside in the ready queue at the control period?". Too many ready jobs could produce internal conflicts and degrade capacity compared to a smaller number of jobs. Too few ready jobs may not achieve maximum capacity and maximum utilization of the system. It is evident that the number of jobs in the ready queue at the control period will depend on the workload imposed on the system.

Because of the excessive amount of storage required by the MPOS package, particularly when the Gomory algorithm is used, the number of concurrent ready and potentially active jobs was restricted to a maximum of 8 at each control period of the simulation study.

The number 8 is the result of different tests conducted prior to the simulations, in an effort to establish the threshold at which central memory space required by the simulation software package would exceed the maximum amount of core memory available for use on the CDC 6400. Careful analysis of the data provided by the Computing Center further showed that a maximum of 8 ready/potentially active jobs at the control period was approximately equivalent to a maximum cumulative time limit request of 35 system seconds above which the simulation software will request excess execution storage space. Consequently, the Fortran program stops scanning and reading the ready queue as soon as the cumulative execution time limit exceeds 35 system seconds, or when the number of jobs scanned is equal to 8, whichever condition prevails. The program then begins the coding of the zero-one problem in algebraic format (variables table only) for input to the MPOS package.

3. Determination of C_j's.

The Fortran program reads the characteristics of each test jobs in the attribute array, ATRIB (J, K), K = 1,9. The C_j associated to any job j is determined with the values of ATRIB (J, 1), ATRIB (J, 5), ATRIB (J, 6), ATRIB (J, 7) and ATRIB (J, 9).

The values of the attributes array for job j are the following:

ATRIB (j,1)	:	central memory request (in decimal)
ATRIB (j,2)	:	central processing time request
ATRIB (j,3)	:	channel time request
ATRIB (j,4)	:	peripheral processor time request
ATRIB (j,5)	:	system seconds
ATRIB (j,6)	:	job card priority
ATRIB (j,7)	:	processing period: 1. for Prime-Time 2. for Non-Prime Time 3. for Off-Hours
ATRIB (j,8)	:	Time of entry in the computer system (time is in seconds of the century)
ATRIB (j,9)	:	processing mode: 1. for Batch jobs 2. for Interactive jobs

The formula used for the computation of the C_j's is the following:

$$C_j = ATRIB\ (j,1) * ATRIB\ (j,5) * CMF * C' * PF$$

where: C' is the execution charge unit,
PF is the priority factor of the job
and CML is a central memory factor.

Replacing ATRIB (j,1) and ATRIB (j,5) by s_j and t_j respectively, yields the formula:

$$C_j = CMF * PF * C' * s_j * t_j$$

setting $C = C' * CMF * PF$, the expression for C_j

becomes: $C_j = C * s_j * t_j$

which was arrived at differently in chapter II. The CMF and PF factors are dimensionless whereas C' and therefore C are expressed in $/word/system second.

The priority factor (PF) of the job is established by the scheduler as a function of the job card priority of the job. The job card priority (JCP) /Priority factor (PF) function or (JCP)/(PF) function for the study is represented by the following table: (batch jobs only)

JCP	0	1	2	3	-----
PF	1	1.2	1.4	1.6	-----

(JCP/PF function for batch jobs only)

For interactive jobs, the PF is also a function of the processing period; interactive jobs have JCP equal to 0. The (JCP)/(PF) function for interactive jobs is the following:

Processing Period	Prime	Non-Prime	Off-Hours
PF	2.6	2.2	1.7

(JCP/PF function for interactive jobs)

The central memory factor or CMF of the expression depends on the level of central memory request for the processing period. It is established through the lookup of the following table:

Processing Period \ CM Request	30K 921	60K 384	100K 512	120 K 640	(octal units) (system units)
Prime Hours	.38	.50	.62	.80	
Non-Prime Hours	.36	.45	.52	.64	
Off-Hours	.33	.36	.38	.40	

CM Factor Table

(X system units $\equiv$ $(X_8/10)$K octal units)

(X_8 is the value of X expressed in octal)
K = 1024 words

For this simulation model, the short term fluctuations of the C parameter will not be considered.

The formula used for the generation of any process C_j

guarantees that the value of C_j is directly proportional to the process request for time and only "weakly" proportional to the process request for space. Time is the most critical and most limiting constraint of the scheduling process.

4. Determination of B_j's.

The B_j parameter represents the process maximum pay-will for space · time resource request. Processes entering the system with a B_j parameter less than the corresponding C_j parameter are automatically rejected by the scheduler. For this simulation model, every process's B_j is obtained by uniformly randomizing around the process's corresponding C_j. The randomization is performed in such a way that the B_j will always exceed the C_j; the expression of B_j generated by the simulator is as follows:

$B_j = C_j * (1 + XX)$, where XX is a uniform random variable between 0 and 1.

The model does not simulate real life situations in which, when the B_j is less than the corresponding C_j, the computer system scheduler would flag the job as unacceptable. The scheduler would then display a "rejection list" and would accept interactive or batch modification of the offending B_j's. Modification of the B_j's is difficult to model since the distribution of users' changes cannot be clearly characterized.

5. Objective Function.

The coefficient for every $e_{j,T}$ variable is the difference between the job's B_j and its corresponding C_j. The program writes the objective function according to that definition.

6. Constraints.

Constraints (1), (3) and (4) are written by the program exactly as it was explained in the previous chapter. Constraints (2), however, are of the form:

$$\sum_{j=1}^{q} (e_{j,t+t_j} - e_{j,t}) \geq 1 \text{ for } t = \Delta t, 2\Delta t, \ldots\ldots, T - \Delta t, T$$

and indicate that at least one job must be in execution at any time period. These constraints guarantee (in theory) that the system will be multiprogrammed. It was observed during the experimental tests that the zero-one machine will always attempt to schedule a job whenever possible because of the MAXIMIZE clause of the zero-one formulation

For that matter, constraints of type 2 do not have any effect on the basic solution of the algorithm when the operating interval length is fixed in advance. Their effect on the scheduling decisions is covered by the inherent structure of the zero-one algorithm. It was therefore decided to remove the constraints from the MPOS input file.

The simulator creates a few additional constraints in an attempt to reduce MPOS activity; MPOS will consider for scheduling only those variables of the zero-one formulation which are effectively active. (Chapter II, section "Simplification of the Zero-one Formulation.)

7. Optimum Length of the Operating Interval.

A question which arose often in the course of this experimental work was: "What is the optimum length of the control interval?". It is undeniable that the length of T is critical to the performance of the scheduler. A long control interval would facilitate the scheduling decisions but would tend to degrade storage capacity; a short operating interval would increase storage efficiency but would complicate scheduling decisions. Let us denote by PERF (Performance), the ratio of the cumulative time request for all jobs in the ready queue over the length chosen for the control interval. The larger PERF, the more complex but the more efficient scheduling becomes. For the simulation model presented here, the PERF factor has been chosen to be 5 because of the restraints of the MPOS package and the limited availability of executable central memory. Since the maximum cumulative time request acceptable in the simulation model is 35 systems seconds, the corresponding maximum length of T is therefore 7 systems seconds. It was not possible to study the fluctuations of the scheduling decisions for variations of T. This, we believe, would have been possible and enriching had we been able to have access to larger commercial industrial optimization packages. Figure 3 - 3 represents a typical MPOS input file generated by the Fortran program. The characteristics of the jobs considered in figure 3 - 3 are displayed in figure 3 - 2.

JOBS PARAMETERS (RUN NUMBER 1)

CM	CP	CH	PP	(SS/10)	JCP	PPH	MODE	PAYMIN	PAYWILL
160	1.0	.7	5.8	.4	0	1	1	45.	48.
129	.3	.2	2.2	.1	0	1	1	9.	13.
168	.6	.4	1.9	.3	0	1	1	35.	67.
132	.6	.5	2.5	.3	0	1	1	28.	33.
160	2.0	.2	2.2	.7	0	1	1	68.	122.
224	1.7	1.1	13.0	.6	0	1	2	245.	459.
160	1.4	.9	3.5	.6	0	1	1	67.	113.
224	1.3	.6	2.6	.6	0	1	1	94.	135.

FIGURE 3 - 2.

(Paymin and Paywill are rounded to the nearest unit.)

```
MPOS VERSION 3.2        NORTHWESTERN UNIVERSITY

     **************************************
     *                                    *
     *              M P O S               *
     *                                    *
     *            VERSION 3.2             *
     *                                    *
     * MULTI-PURPOSE OPTIMIZATION SYSTEM  *
     *                                    *
     **************************************

***** PROBLEM NUMBER  1 *****

       TITLE
       MUTEX
       *****  USE CUTTING PLANE ALGORITHM OF GOMORY  *****
       GOMORY

       *****  VARIABLE TABLE  *****
       INTEGER

       *****  CONTROL INTERVAL OF 7 SYSTEM SECONDS  *****

       A1001      TO      A1007
       B1001      TO      B1007
       C1001      TO      C1007
       D1001      TO      D1007
       F1001      TO      F1007
       G1001      TO      G1007
       I1001      TO      I1007
       J1001      TO      J1007

       *****  OBJECTIVE FUNCTION DEFINITION  *****
       MAXIMIZE

        +    3A1007 +    4B1007 +    32C1007 +    5D1007 +    54F1007 +  214G1007
        +   46I1007 +   41J1007 +

       *****  CONSTRAINTS DEFINITION  *****
       CONSTRAINTS

       *****  CONSTRAINTS OF TYPE 1: STEP FUNCTION  *****
    1.  A1001 -A1002                                 .LE. 0.
    2.  A1002 -A1003                                 .LE. 0.
    3.  A1003 -A1004                                 .LE. 0.
    4.  A1004 -A1005                                 .LE. 0.
    5.  A1005 -A1006                                 .LE. 0.
    6.  A1006 -A1007                                 .LE. 0.
    7.  B1001 -B1002                                 .LE. 0.
    8.  B1002 -B1003                                 .LE. 0.
    9.  B1003 -B1004                                 .LE. 0.
   10.  B1004 -B1005                                 .LE. 0.
   11.  B1005 -B1006                                 .LE. 0.
```

FIGURE 3 - 3

```
12.   B1006 -B1007                                    .LE. 0.
13.   C1001 -C1002                                    .LE. 0.
14.   C1002 -C1003                                    .LE. 0.
15.   C1003 -C1004                                    .LE. 0.
16.   C1004 -C1005                                    .LE. 0.
17.   C1005 -C1006                                    .LE. 0.
18.   C1006 -C1007                                    .LE. 0.
19.   D1001 -D1002                                    .LE. 0.
20.   D1002 -D1003                                    .LE. 0.
21.   D1003 -D1004                                    .LE. 0.
22.   D1004 -D1005                                    .LE. 0.
23.   D1005 -D1006                                    .LE. 0.
24.   D1006 -D1007                                    .LE. 0.
25.   F1001 -F1002                                    .LE. 0.

26.   F1002 -F1003                                    .LE. 0.
27.   F1003 -F1004                                    .LE. 0.
28.   F1004 -F1005                                    .LE. 0.
29.   F1005 -F1006                                    .LE. 0.
30.   F1006 -F1007                                    .LE. 0.
31.   G1001 -G1002                                    .LE. 0.
32.   G1002 -G1003                                    .LE. 0.
33.   G1003 -G1004                                    .LE. 0.
34.   G1004 -G1005                                    .LE. 0.
35.   G1005 -G1006                                    .LE. 0.
36.   G1006 -G1007                                    .LE. 0.
37.   I1001 -I1002                                    .LE. 0.
38.   I1002 -I1003                                    .LE. 0.
39.   I1003 -I1004                                    .LE. 0.
40.   I1004 -I1005                                    .LE. 0.
41.   I1005 -I1006                                    .LE. 0.
42.   I1006 -I1007                                    .LE. 0.
43.   J1001 -J1002                                    .LE. 0.
44.   J1002 -J1003                                    .LE. 0.
45.   J1003 -J1004                                    .LE. 0.
46.   J1004 -J1005                                    .LE. 0.
47.   J1005 -J1006                                    .LE. 0.
48.   J1006 -J1007                                    .LE. 0.
      $$$$$  END CONSTRAINTS TYPE 1  $$$$$

      $$$$$  BEGIN SIMPLIFICATION CONSTRAINTS   $$$$$
      $$$$$  THESE CONSTRAINTS FORCE MPOS       $$$$$
      $$$$$  TO CONCENTRATE ON EFFECTIVE CONSTRAINTS  $$$$$
49.   A1003                                              .EQ.  0
50.   C1002                                              .EQ.  0
51.   D1002                                              .EQ.  0
52.   F1005                                              .EQ.  0
53.   G1005                                              .EQ.  0
54.   I1005                                              .EQ.  0
55.   J1005                                              .EQ.  0
```

FIGURE 3 - 3 (continued)

```
*****  END CONSTRAINTS SIMPLIFICATION  *****

*****  BEGIN CONSTRAINTS TYPE 3: RESOURCE CONSTRAINTS  *****
*****  SPACE USED AT ANY TIME MUST BE LESS THAN 640 UNITS  *****

56.   +  160A1005 -  160A1001 +  129B1002 -  129B1001 +  168C1004 -  168C1001
      +  132D1004 -  132D1001 +  160F1007 -  160F1001 +  224G1007 -  224G1001
      +  160I1007 -  160I1001 +  224J1007 -  224J1001 +
                                               .LE.  640

57.   +  160A1006 -  160A1002 +  129B1003 -  129B1002 +  168C1005 -  168C1002
      +  132D1005 -  132D1002 +  160F1007 -  160F1002 +  224G1007 -  224G1002
      +  160I1007 -  160I1002 +  224J1007 -  224J1002 +
                                               .LE.  640

58.   +  160A1007 -  160A1003 +  129B1004 -  129B1003 +  168C1006 -  168C1003
      +  132D1006 -  132D1003 +  160F1007 -  160F1003 +  224G1007 -  224G1003
      +  160I1007 -  160I1003 +  224J1007 -  224J1003 +
                                               .LE.  640

59.   +  160A1007 -  160A1004 +  129B1005 -  129B1004 +  168C1007 -  168C1004
      +  132D1007 -  132D1004 +  160F1007 -  160F1004 +  224G1007 -  224G1004
      +  160I1007 -  160I1004 +  224J1007 -  224J1004 +
                                               .LE.  640

60.   +  160A1007 -  160A1005 +  129B1006 -  129B1005 +  168C1007 -  168C1005
      +  132D1007 -  132D1005 +  160F1007 -  160F1005 +  224G1007 -  224G1005
      +  160I1007 -  160I1005 +  224J1007 -  224J1005 +
                                               .LE.  640

61.   +  160A1007 -  160A1006 +  129B1007 -  129B1006 +  168C1007 -  168C1006
      +  132D1007 -  132D1006 +  160F1007 -  160F1006 +  224G1007 -  224G1006

      +  160I1007 -  160I1006 +  224J1007 -  224J1006 +
                                               .LE.  640
*****  END CONSTRAINTS TYPE 3  *****

*****  BEGIN CONSTRAINTS TYPE 4  *****
*****  MAXIMUM EFFICIENCY CONSTRAINT OF SHORE  *****

62.   +  640A1007 +  129B1007 +  504C1007 +  396D1007 +  960F1007 + 1344G1007
      +  960I1007 + 1344J1007 +
                                                  .LE.    4480
*****  END CONSTRAINT TYPE 4  *****

*****  CONSTRAINTS TYPE 5: ZERO-ONE CONSTRAINTS  *****
BNDALL   1
LIMIT   5000
OPTIMIZE
```

FIGURE 3 - 3 (continued)

3. MPOS Solution.

The objective of the MPOS package is to determine a zero-one integer solution from which a possible schedule could be derived. The zero-one integer solution implicitly denotes those processes that will not get access to central memory. The $e_{j,T}$ variable of such processes is always equal to 0. Let us consider a job j and its associated sequence of $e_{j,t}$, $e_{j,1}$, $e_{j,2}$, ..., e_{j,t_1}, e_{j,t_1+1}, ..., $e_{j,T}$

($\Delta t \equiv 1$ for this simulation study.)

Suppose e_{j,t_1} is the first of those $e_{j,t}$'s whose value is equal to 1 when reading from the left (e_{j,t_1} is the first non-zero variable). All other values to the right of e_{j,t_1} will be equal to 1, whereas all values to the left of e_{j,t_1} will be 0. e_{j,t_1} indicates the exact time at which process j completed execution and was released from the system; (the control point at which the job was executing is made available to another potential process). Since swapping is not allowed, the knowledge of job j execution time limit request permits determination of the execution period of the job and, therefore, the job's schedule. Job j started executing at time $t_1 - t_j$.

The MPOS output for the input file shown in figures 3 - 2 and 3 - 3 is displayed in figure 3 - 4. The appropriate schedule for the control interval considered is drawn in figure 3 - 5.

```
OBJECTIVE FUNCTION =          318 AT ITERATION        244
                                  TIME =     29.228 SECS.
```

SUMMARY OF RESULTS

A1001 =	0	A1002 =	0	A1003 =	0	A1004 =	0
A1005 =	0	A1006 =	0	A1007 =	0	B1001 =	0
B1002 =	0	B1003 =	0	B1004 =	0	B1005 =	0
B1006 =	0	B1007 =	1	C1001 =	0	C1002 =	0
C1003 =	0	C1004 =	0	C1005 =	0	C1006 =	0
C1007 =	0	D1001 =	0	D1002 =	0	D1003 =	0
D1004 =	0	D1005 =	0	D1006 =	0	D1007 =	0
F1001 =	0	F1002 =	0	F1003 =	0	F1004 =	0
F1005 =	0	F1006 =	1	F1007 =	1	G1001 =	0
G1002 =	0	G1003 =	0	G1004 =	0	G1005 =	0
G1006 =	1	G1007 =	1	I1001 =	0	I1002 =	0
I1003 =	0	I1004 =	0	I1005 =	0	I1006 =	1
I1007 =	1	J1001 =	0	J1002 =	0	J1003 =	0
J1004 =	0	J1005 =	0	J1006 =	0	J1007 =	0

FIGURE 3 - 4

Partial Output

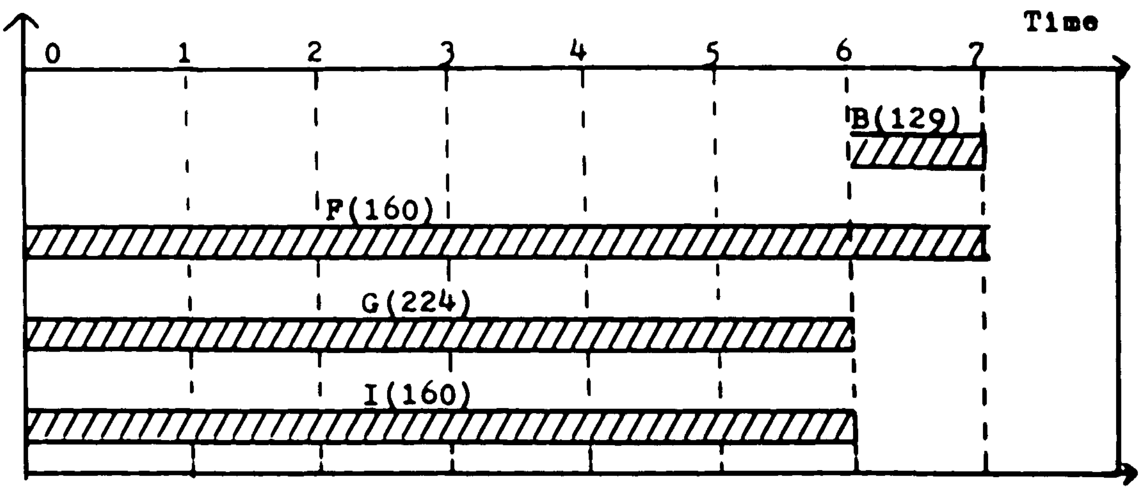


FIGURE 3 - 5
(the number in parentheses represents CM requests)

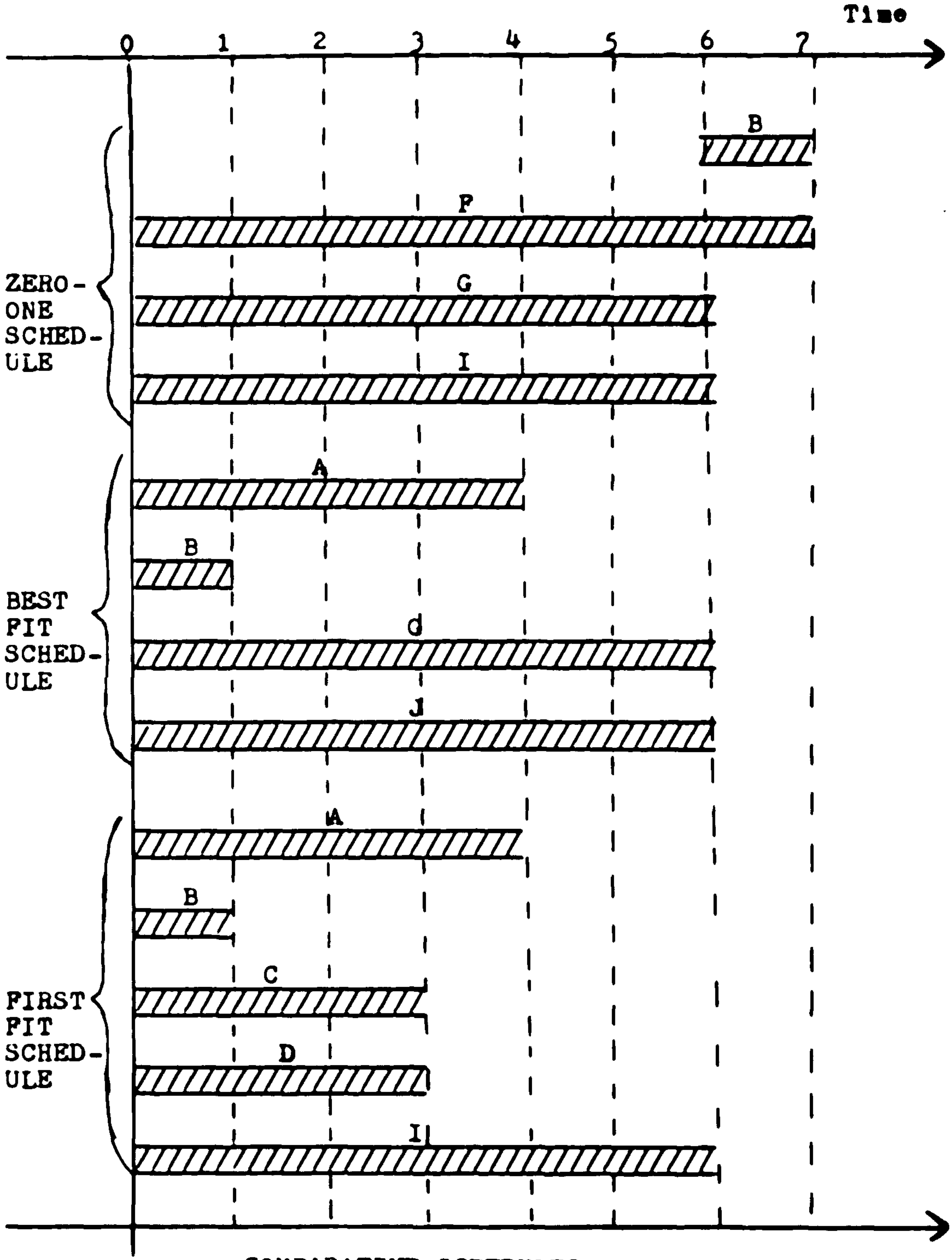
The value of the objective function is the optimum schedule's differential return to the computer system. By differential returns we mean the difference value between the B and C parameters of the scheduled jobs. The differential returns represent the net returns to the computer system. For the example shown in figures 3 - 3, 3 - 4, and 3 - 5, the objective function value is 318 dollars. The corresponding returns obtained in the same control interval with a Best Fit and First Fit strategies are 262 and 90 dollars respectively. Figures 3 - 6 and 3 - 7 show the comparative schedules obtained with the Best Fit and First Fit algorithms for the jobs of figures 3 - 3 and 3 - 4.

JOBS	CM	(SS/10)	PAYMIN	PAYWILL	BJ-CJ	ZERO-ONE	BEST FIT	FIRST FIT
A	160	.4	45.	48.	3.	0	1	1
B	129	.1	9.	13.	4.	1	1	1
C	168	.3	35.	67.	32.	0	0	1
D	132	.3	28.	33.	5.	0	0	1
F	160	.7	68.	122.	54.	1	0	0
G	224	.6	245.	459.	214.	1	1	0
I	160	.6	67.	113.	46.	1	0	1
J	224	.6	94.	135.	41.	0	1	0

FIGURE 3 - 6

COMPARATIVE ZERO-ONE, BEST FIT AND FIRST FIT SCHEDULES

FIGURE 3 - 7



COMPARATIVE SCHEDULES

Figure 3 - 8 represents partial results for 20 control interval decisions. It was found that in terms of differential dollar returns, the zero-one algorithm outperformed the Best Fit and First Fit algorithms by factors of .095 and .14 per control interval, respectively. The results mean that under normal conditions of control interval operations, one should expect the zero-one algorithm model presented in this thesis to generate substantially more dollar returns to the computer system than the Best Fit or First Fit algorithms for example.

RUN NUMBER	FIRST FIT	BEST FIT	ZERO-ONE
1	90.	262.	318.
2	66.27	123.18	140.25
3	8.86	80.95	105.14
4	42.62	82.09	178.02
5	133.26	133.26	154.38
6	137.47	183.61	272.06
7	183.03	128.81	223.02
8	52.73	47.98	322.34
9	58.29	58.29	116.06
10	61.62	56.87	105.88
11	0.00	49.71	78.95
12	51.21	46.18	84.95
13	15.77	70.61	86.96
14	62.93	81.69	147.63
15	20.87	75.36	105.57
16	7.12	19.83	124.96
17	30.58	42.99	191.29
18	2.36	14.77	162.81
19	18.36	19.96	49.58
20	23.57	23.57	105.84

FIGURE 3 - 8

DIFFERENTIAL RETURNS

The zero-one algorithm concept is especially applicable when central memory is overcrowded. The results of the simulation suggest that computer centers administrations have, with the zero-one algorithm, a tool to use memory overcrowding as a means to generate extra dollars returns. In addition, the zero-one algorithm guarantees optimum scheduling decisions for the control interval.

The control interval approach assumes that jobs arrive in the system by intervals and that memory is overcrowded at the control period*. Moreover, under the control interval approach, it is not, in general, good policy to schedule a job during the current interval if there is a possibility to schedule the same job during the following control interval when the presence of other jobs will have generated more competition for storage.

The control interval approach, therefore, generates competition between the jobs or the customers of the system before attempting any servicing. The example of figure 3 - 9 should help clarify this part of the concept.

---

* control period: is the time at which scheduling decisions for the next operating interval are made.

JOBS	CM	SS	$B_j - C_j$	ZERO-ONE
A	160	4	3.	0
B	129	2	4.	1
C	168	5	32.	0
D	132	5	5.	0
F	160	7	54.	1
G	224	5	214.	1
I	160	5	46.	1
J	224	5	41.	0

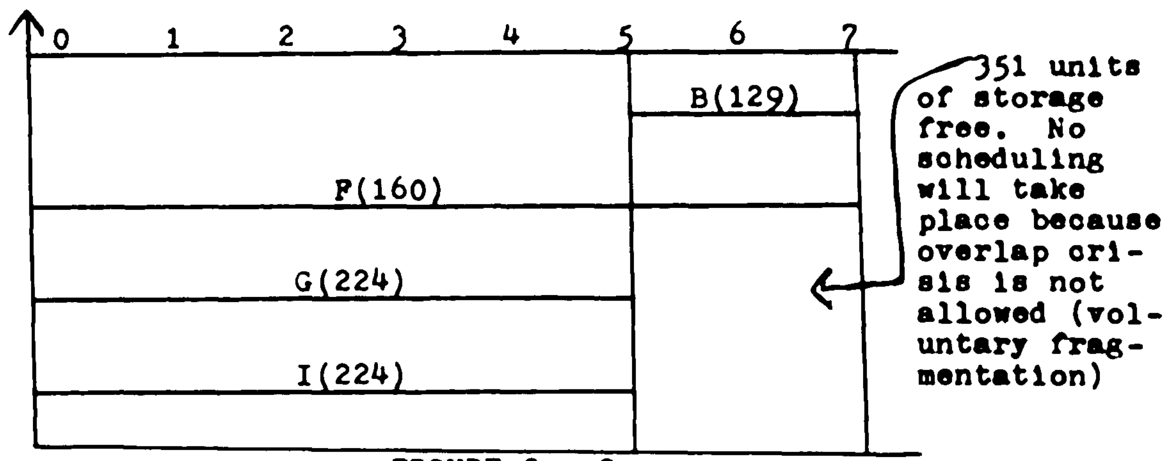


FIGURE 3 - 9

The control interval approach in a computer micro/macroscheduling environment also means that the computer system central authority as well as the computer system users:

- are fully aware of the preciousness of computer resources;
- recognize that precious resources are best used under competitive circumstances;
- and are willing to bring an honest contribution to achieving optimum utilization of the resources.

In addition, the approach assumes that the computer system functions as a completely closed system; that is users of the system will never balk.

When all the above conditions are fulfilled, the control interval approach is said to be operating under "normal conditions".

The control interval approach creates the phenomenon of voluntary fragmentation illustrated in figure 3 - 9. The fragmentation is voluntary in the sense that it is accepted or created by the scheduler as a means to generating additional dollars returns. Any cost incurred with such voluntary fragmentation should be paid off by the excess dollars returns. The fragmentation problem may, nonetheless, be alleviated, in practice, by switching from time to time and at the appropriate moment from the zero-one algorithm to some other passive algorithms. This technique of controlling fragmentation permits to effectively write "Generalized Zero-One Schedulers". This sort of scheduler will be the object of section 2 of chapter IV. This problem of fragmentation is in fact a consequence of the overlap crisis assumption mentioned in chapter II.

The control interval approach will occasionally delay the execution of some jobs entitled to access executable memory (jobs j with $B_j > C_j$). This should not be a problem since the scheduler modifies scheduling decisions by accepting input from the user community. The feedback loop: scheduler ⟶ users ⟶ scheduler enables every user to very easily change his or her jobs relative priority by increasing or decreasing the jobs' B parameters.

C. Conclusion.

The search for ways to monitor the performance of the "subjective zero-one" algorithm presented in chapter II has led to several simulation studies. Some important results of the simulations were presented in this chapter.

The studies provided visibility of the utilization of storage resources only and do not attempt to cover scheduling of other resources such as CPU, channels and devices. Analysis of the results showed that, in general, the zero-one scheduler will always outperform the passive schedulers described in chapter I, insofar as dollars returns to the system are concerned. The performance of the scheduler has been found to be dependent upon the user community maturity and awareness of the preciousness of the resources to be scheduled.

Conditions of optimum performance of the zero-one scheduler have been defined and investigated; many of those conditions exist in today's computer systems and centers.

The scheduler creates additional fragmentation; but that problem of fragmentation could be controlled with generalized schedulers that will be presented in chapter IV.

IV. CONCLUSIONS

The objective of this thesis was the determination of the feasibility of a non paged multiprogrammed memory allocator based on the idea of users' own pricing of storage resources. By constructing an abstract model for a zero-one linear scheduler, we have built a framework within which we have analyzed different schedules developed by a simulator program. It was found that a zero-one algorithm is perfectly feasible and under certain conditions of normality, will substantially outperform the best fit and first fit algorithm for example, insofar as dollars returns to the system are concerned.

We have had to define the concept of control interval approach which is the environment within which performance of the zero-one scheduler is maximum. Once we had accepted the idea that storage resources are best and most efficiently utilized under conditions of tight competition, we have been able to more clearly define performance measures.

It has become clear that fragmentation was not to be considered as a problem; as a matter of fact, fragmentation has been found sometimes necessary to guarantee maximum profitability of the overall scheduling process.

In this chapter, we intend to present some of the peripheral aspects of the concept of subjective scheduling.

We will also present and discuss certain areas of the work which appear to be disappointing.

1. Is the Zero-One Algorithm Fair?

The economic essence of sharing and multiprogramming can be captured in this sentence:

"by sharing resources, the users distribute the resources costs and each user pays less" [4]. Sharing benefits the system, too, for the system selects from a wide range of instantaneous requests those that are most likely to improve its efficiency. However, sharing creates the problem of priority rating.

The priority rating problem is very acute in a zero-one algorithm environment. Because users can directly influence their processes priority rating, a question naturally arises:

"how can we guarantee fairness in an environment whereby priority decisions are based on individual's monetary wealth?".

It takes little to realize that economic systems very often fail to be as fair as they ought to be. The systems protect themselves by the institution of laws and legislation.

We have wanted, in this thesis, to provide fairness in the computer system users community. We have based our work on the assumption that conditions of control interval approach exist in computer centers, or can be readily attained with minimum effort. In addition, we have implicitly hypothesized that pure competition exist in the scheduling environment. Pure competition is realized when:

- the economic product under investigation is homogeneous;
- each user is small relative to the market;
- all units possess complete knowledge of the economic environment;
- the system is completely closed.

We believe that memory, which is the economic product under study in this thesis, is perfectly homogeneous. Fairness of the system is guaranteed by the second clause of the definition which demands smallness of each buyer relative to the market. This thesis assumes that there cannot exist in the system a user with the largest differential parameter at all times. This is only an assumption and will probably not be true in many computing centers. It is the responsibility of the computing center administration to guarantee fairness in the environment should any clause of the pure competition model be violated.

2. Suggestions for Improvements and Further Studies.

In chapter II, we have presented constraints of type 2 as necessary constraints of the zero-one problem. Constraints (2) guarantee that at least one program is executing at any instant of the control interval. The removal of constraints (2) from the simulation model presented in chapter III, should not be taken as an indication of the superfluity of the constraints.

One of the most obvious flaws of the simulation model is the presetting of the length of the control interval. We had to adopt that attitude bacause of the limitations of the software utilized for the study. The presetting of the length of the operating interval to a value T, introduces the possibility that the whole core be left unused toward the end of the control interval, when the scheduler is waiting for the next control period. Scheduling decisions would nevertheless be optimum for the control interval as displayed in figure 4.1 and 4.2.a. Storage would, however, be better managed with the reduction of the control interval length as in figure 4.2.b. To the management of space, we have thus added the management of time. This is not surprising since time and space are inseparable physical entities.

JOBS	CM	SS	$B_j - C_j$
A	160	2	3
B	129	2	5
C	168	2	25
D	132	5	15
F	132	4	30
G	170	3	50
I	200	1	10
J	100	1	10

FIGURE 4 - 1

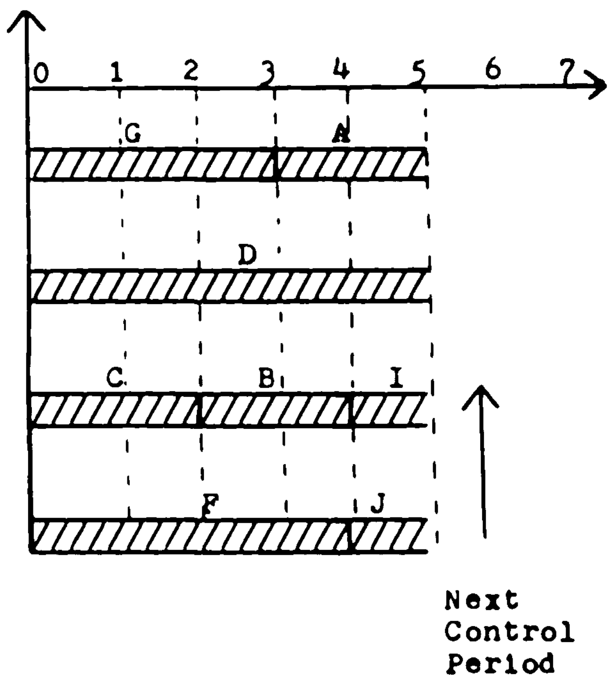


FIGURE 4 - 2 a)

whole core is left unused for 2 system seconds.

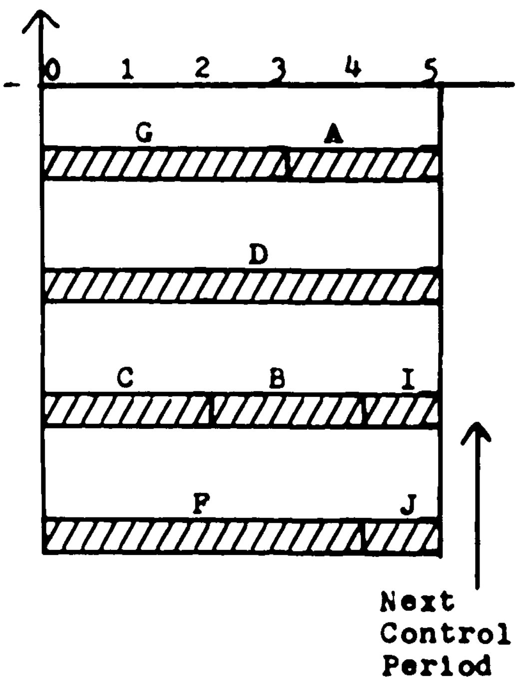


FIGURE 4 - 2 b)

adjustment of control interval length.

The control interval time becomes unpredictable and a random variable. Its distribution is a function of the system workload. The introduction of a dynamic management of time makes it difficult to study the behavior of the scheduler through simulation. We suggest that the following procedure be used in real scheduling:

1. define scheduling decisions variables with a preset value of T;

2. adjust T by reduction if possible.

We have introduced in chapter III the concept of voluntary fragmentation. Voluntary fragmentation is a sound policy under normal conditions of control interval approach. Yet it becomes undesirable if storage must be left unused for a long period of time. Again, management of time combines with management of space to remind us that space and time cannot be separated from each other. Let X be the amount of voluntary fragmentation created by the scheduler for a length of time t. The product X * t is then a random variable whose distribution will depend on the system workload. The longer the length of the control interval, the larger the expected value of X * t. Note that voluntary fragmentation only takes place toward the end of the control interval.

Two questions now arise:

1.) What is the optimum value of T which will guarantee maximum memory time product efficiency, and minimize the expected value of the product X * t?

2.) Will that optimum value of T guarantee that the system will not degenerate into a "thrashing"-state?

By thrashing-states, we want to describe situations whereby the length of the control intervals compels the scheduler to spend more time making and revising scheduling decisions, than effectively scheduling and allocating storage.

The answers to the questions we have raised in the above discussion will make it possible to design what we have previously referred to as generalized zero-one schedulers. Generalized zero-one schedulers would make full use of all constraints defined in chapter II. They would be able to conduct their own look ahead simulations for adjustment of T.

Generalized schedulers, as well as the scheduler presented in this thesis can be written in a higher procedural language such as Pascal.

We have just described and discussed a few of the many opportunities left opened for additional exploration of the idea of a Subjective Zero-One scheduler in a free enterprise system.

An area of the study which was somewhat disappointing is the apparent overhead created by the zero-one scheduler during the simulation experiments. It is not clear how much overhead is involved. Moreover, there is no possibility to estimate the speed at which scheduling decisions will be made. Processing speed is a very important factor in the design of any type of system program. The optimization software used for the simulation experiments was not designed for real time processing and was therefore very slow in arriving at useful scheduling decisions. The objective of this thesis was not the development of a procedure for real time zero-one optimization.

The author nonetheless believes that much attention should be directed to that effect before the storage allocation algorithm presented in this thesis becomes practical.

We have tried in this thesis to develop a procedure whereby users would have the ability to influence their jobs' priority rating through direct interaction with the job scheduler. The question is to know how much the user will be appreciative of the effort. The extent to which the user would want to be concerned with the scheduling process is not clearly understood. More work needs to be done in order to determine the limits of acceptable users' involvement in scheduling processes.

Multiprogramming, multiprocessing, and all other techniques are not solutions to the resources allocation problem; they are tools by which a solution may be implemented [4].

It was the purpose of this thesis to develop and present a relatively new approach to modelling the behavior of computational processes, to spark a different way of thinking about microscheduling, to evolve a philosophy about storage as an economic good of the computer environment.

We hope we have achieved that purpose. We also hope that some effort will be expended in the future to develop models of computer systems resources sharing and utilization similar to the model presented in this thesis.

BIBLIOGRAPHY

The following abbreviations are used:

CACM:	communications of the ACM
Proc. IEEE:	proceedings of the IEEE
NBS:	National Bureau of Standards

1. Brown, R. M., Browne, J. C., Chandy, K. H. Memory Management and Response Time, CACM, Vol. 20, N3, March 1977, pp. 153 - 165

2. Conti, M. D., A Case Study in Monitoring the CDC-6700-A Multiprogramming, Multiprocessing, Multi-Mode System. U.S. Department of Commerce, NBS Special Publication 401, Computer Performance Evaluation, September 1974. pp. 115 - 118

3. Demming, P., The Working Set Model for Program Behavior. CACM, Vol. 11, N5, May 1968, pp. 323 - 333

4. Demming, P., Resource Allocation in Multiprocess Computer Systems. Project MAC, Massachusetts Institute of Technology, Report No. MAC - TR - 50 (Ph.D Thesis)

5. Donovan, J. J., Systems Programming, Project MAC, Massachusetts Institute of Technology, McGraw - Hill (New York), 1972, pp. 366 - 383

6. Ellison, C. M., The Utah Tenex Scheduler (Utah University), Proc. IEEE, Vol. 63, June 1975, pp. 940 - 945.

7. Kleinrock, L., Optimum Bribing for Queue Position, The Journal of the Operations Research Society of America, Vol. 15, N2, March - April 1967, pp. 304 - 318

8. Kleinrock, L., Queuing Systems, Vol. II: Computer Applications, Wiley (New York) 1975

9. Knuth, D. E., The Art of Computer Programming, Vol. 1, Fundamental Algorithms, Addison Wesley, Reading (Massachusetts), 1968, pp. 435 - 452

10. Lewis, T. G., Smith M. Z., Applying Data Structures, Houghton Hill Company (Boston) 1976, pp. 163 - 181

11. MacMillan C., Jr., Mathematical Programming, 2nd edition, Wiley (New York), 1975, pp. 443 - 552

12. Marchand, M., Priority Pricing With Application to Time Shared Computers, AFIPS Conference Proceedings, Vol. 33, Part 1, 1968, Fall Joint Computer Conference, pp. 511 - 519

13. Nielsen, N. R., Flexible Pricing: An Approach to the Allocation of Computer Resources, AFIPS Conference Proceedings, Vol. 33, Part 1, 1968, Fall Joint Computer Conference, pp. 521 - 531

14. Randell, B. A., A Note on Storage Fragmentation and Program Segmentation, CACM, Vol. 12, N7, July 1969, pp. 365 - 372

15. Sahney, K. V., May L. J. Scheduling Computer Operations Publication N. 6, Computer and Information Systems Division, AIIE, Inc., 1972, No. AIIE - C IS - 72 - 6.

16. ____________________., Scope 3 - 4 Reference Manual, Control Data, Cyber 70 Series, Models 72, 73, 74, 6000 Series Computer Systems, Control Data Corporation, No. 60307200

17. Singer, N. M., Kanter, H., Moore, A., Prices and the Allocation of Computer Time, AFIPS Conference Proceedings, Vol. 33, Part 1, 1968, Fall Joint Computer Conference, pp. 493 - 498

18. Shaftel, J. L., Zmud, R. W., Allocation of Computer Resources Through Flexible Pricing. The Computer Journal, Vol. 17, N4, November 1974, pp. 306 - 312

19. Shore, J. E., On the External Storage Fragmentation Produced by First Fit and Best Fit Allocation Strategies, CACM, Vol. 18, N8, August 1975, pp. 433 - 440

20. Shiveley, W. M., The Micro-Scheduling of Computer Jobs. Ph.D Thesis, Industrial Engineering Dept., Lehigh University, 1971

21. Stein, J., Cohen, C., Multi-Purpose Optimization System (MPOS) User's Guide, Version 3, Manual No. 320, July 1976, Vogelback Computing Center, Northwestern University, Evanston, Illinois, 60201

22. Venesse, D. M., A Methodology for Performance Measurement. U.S. Department of Commerce, NBS Special Publication 401, Computer Performance Evaluation, September 1974, pp. 15 - 22

23. Program is on file in LUCC Library No. 0004G0, MUTEX.
Lehigh University Computing Center
Bethlehem, PA 18015

APPENDIX

APPENDIX 1

- CDC SCOPE Central Memory Usage and Allocation

- Operator/Scope communication.

Each job in process in the computer system occupies a contiguous block of words in central memory. References to all addresses within each block are made in relation to the reference address (RA) which is the first address in the block. The length of the block is the field length (FL) of the job. A reference to a location outside the job's field length causes an abnormal termination of processing. Thus, all other jobs and systems programs in central memory are fully protected against accidental overwriting.

Every job in central memory is related to a SCOPE control point. Each control point interrelates the following elements common to a particular job: the central memory field length allotted; other hardware and files used by the job; and a control point area in low core, that contains reference information about the job. Reference information are such information as the job name, processing time accumulated, related control statements and the job's exchange jump package.

Up to 15 control points are available; therefore, up to 15 system or user jobs may be active at control points simultaneously. Control point 0 (zero) is used to identify all hardware and software resources not presently allocated to user jobs or those used only by SCOPE.

The position of central memory storage allocated to each job is related to the control point number to which the job is assigned. The assignment is made and maintained in numerical order. The job at control point 2, for instance, always follows the job at control point 1, and the job at control point 3, will follow the job at control point 2. Figure A - 1 represents central memory allocation as maintained by SCOPE.

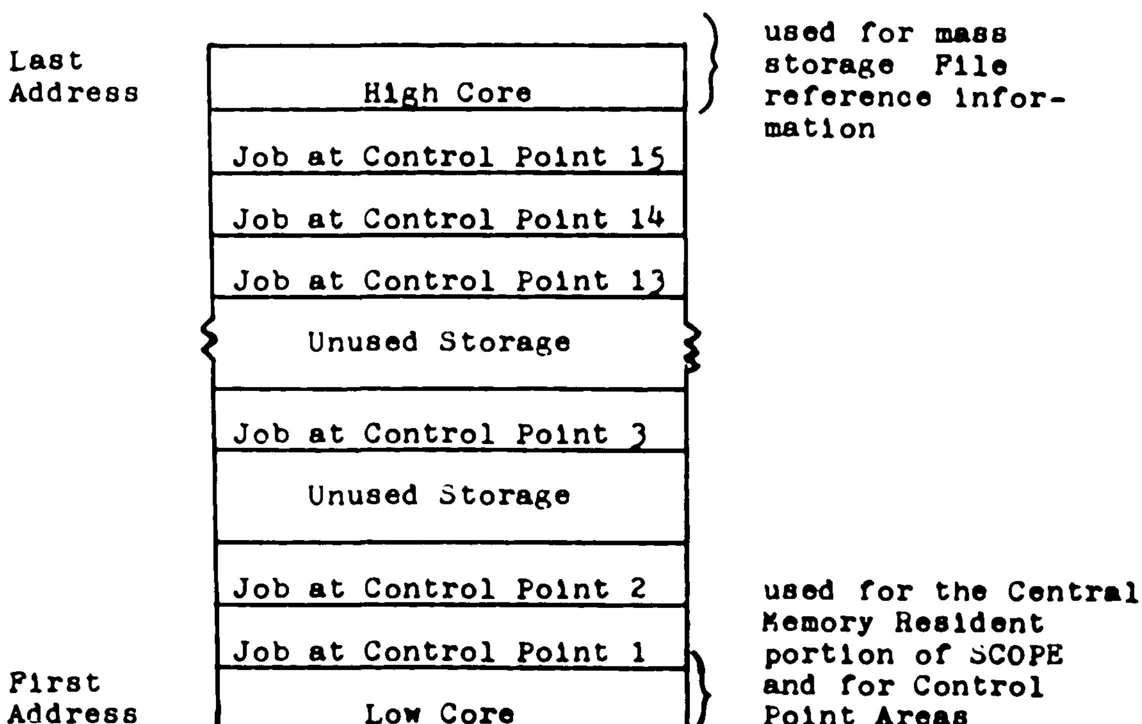


FIGURE A - 1 CENTRAL MEMORY ALLOCATION

CDC SCOPE 3.4 OPERATING SYSTEMS

Through a dynamic relocation process, jobs are moved up and down in storage to make room for new jobs assigned to control points. The process is continuous. If an arriving job is assigned at a free control point and if sufficient contiguous storage is not available for the new job, SCOPE will relocate other jobs as necessary to provide sufficient contiguous storage. Each job will be moved as a block, and only its reference address (RA) will be changed accordingly within the appropriate SCOPE reference tables. The order of the jobs within central memory remains the same. When the move is complete, the RA of the job or jobs are modified and jobs' activity is resumed.

A program gains or relinquishes the central processor through an exchange jump instruction. When this instruction is executed, the program using the central processor is interrupted. The control point area contains a 16-word exchange package which contains the information used directly in exchange jumps: the most recent contents of all processor registers, the RA and FL in central memory and ECS and the program address. The program address is the address of the next instruction to be executed.

SCOPE maintains in mass storage the job dayfile, a chronological accounting of each job run, which is automatically printed at job termination. It contains a copy of all control cards processed, equipment assignments, diagnostic messages, job accounting information, job statistics, and the date and time of day associated with each processing event relative to the job.

- Job Termination

a) Normal Termination

When a job is processed without error, normal termination activity begins upon reaching the first end of the record field or an EXIT or EXIT(S) control card. All hardware devices assigned to the job are assigned to control point 0 (zero), so they can be reassigned to other jobs.

b) Abnormal Termination

When an error occurs, SCOPE sets a flag indicating the error. If the error has not been previously identified in the job step by a call to the system program RECOVR, then SCOPE continues with error processing. Otherwise, control is returned to the user program for processing. A diagnostic message, reflecting the reason for abnormal termination, is written to the job dayfile. SCOPE then clears the error flag and searches the control cards record for an EXIT control card. If no EXIT statement is found, the job terminates as described under normal termination.

c) Termination by an Operator Command

When the operator types in a DROP command, the job terminates prematurely. End-of-job procedures are initiated as described under abnormal termination.

When the operator types in a KILL command, the job terminates prematurely. All files associated with the job are dropped regardless of name or disposition. The programmer does not receive a dayfile listing.

When the operator enters a RERUN command, the job is terminated and its input file is returned to the input queue, so that it can be run later. The output file is dropped, and a new output file is created.

APPENDIX 2

A P P E N D I X   S E C T I O N
(FIRST 500 JOBS PARAMETERS)

JOB NO	CM	SS	PRIORITY (JOB CARD)	PROCESSING PERIOD	MODE D-INTRCN	PAY-MIN	PAY-MILL
1	160	4.0	0	1	1	44.80	47.04
2	129	1.0	0	1	1	9.83	12.63
3	160	3.0	0	1	1	35.20	67.26
4	132	3.0	0	1	1	27.72	32.65
5	160	6.0	0	1	1	67.20	121.60
6	224	6.0	0	1	2	244.61	450.99
7	160	6.0	0	1	1	67.20	112.07
8	224	6.0	0	1	1	94.08	135.36
9	160	6.0	0	1	2	174.72	346.98
10	130	2.0	0	1	1	18.20	27.60
11	96	6.0	0	1	2	184.83	151.06
12	128	2.0	0	1	1	17.92	32.19
13	256	5.0	0	1	1	89.60	117.00
14	160	6.0	0	1	1	67.20	117.64
15	132	2.0	0	1	1	18.48	23.70
16	192	6.0	0	1	1	80.64	102.30
17	160	6.0	0	1	1	67.20	75.59
18	160	3.0	0	1	1	33.60	63.26
19	224	1.0	0	1	1	15.68	20.06
20	192	5.0	0	1	1	67.20	67.01
21	256	6.0	0	1	1	107.52	133.28
22	224	6.0	0	1	1	94.08	171.39
23	96	4.0	0	1	2	69.09	106.06
24	256	6.0	0	1	1	107.52	134.10
25	192	5.0	0	1	1	67.20	83.26
26	196	6.0	0	1	1	82.32	150.93
27	320	6.0	0	1	1	134.40	260.84
28	192	6.0	0	1	1	80.64	116.67
29	132	3.0	0	1	1	27.72	30.91
30	224	6.0	0	1	1	94.08	147.71
31	131	1.0	0	1	1	9.17	12.23
32	132	1.0	0	1	1	9.24	17.56
33	192	5.0	0	1	1	67.20	116.67
34	132	2.0	0	1	1	18.48	34.35
35	129	2.0	0	1	2	46.59	63.75
36	128	1.0	0	1	1	8.96	17.02
37	136	1.0	0	1	1	9.52	13.84
38	192	5.0	0	1	1	67.20	93.65
39	196	6.0	0	1	1	82.32	90.00
40	224	1.0	0	1	1	15.68	20.15
41	132	1.0	0	1	1	9.24	10.50
42	296	6.0	0	1	2	323.23	339.84
43	131	1.0	0	1	1	9.17	9.31
44	160	6.0	0	1	1	67.20	90.70
45	192	4.0	0	1	1	53.76	60.60
46	296	3.0	0	1	1	62.16	84.81
47	224	2.0	0	1	2	81.94	127.12
48	224	6.0	0	1	1	94.08	95.27
49	130	3.0	0	1	1	27.30	54.44
50	160	6.0	0	1	1	67.20	103.36

51	192	6.0	0	1	1	80.64	132.98
52	296	3.0	0	1	1	53.76	80.33
53	224	6.0	0	1	2	244.61	324.50
54	160	5.0	0	1	1	56.00	80.05
55	160	6.0	0	1	1	67.20	117.09
56	160	4.0	0	1	1	44.80	75.20
57	132	1.0	0	1	1	9.24	13.19
58	128	1.0	0	1	1	0.96	13.06
59	129	2.0	0	1	2	40.59	76.00
60	132	1.0	0	1	1	9.24	17.76
61	160	6.0	0	1	1	67.20	67.63
62	192	5.0	0	1	1	67.20	125.00
63	224	3.0	0	1	1	47.04	89.06
64	228	4.0	0	1	1	63.04	124.09
65	224	1.0	0	1	2	40.77	56.40
66	64	3.0	0	1	2	26.56	33.94
67	160	4.0	0	1	1	44.80	67.74
68	320	5.0	0	1	1	112.00	167.94
69	132	1.0	0	1	1	9.24	11.31
70	196	2.0	0	1	1	27.44	53.00
71	192	5.0	0	1	1	67.20	84.20
72	160	4.0	0	1	1	44.80	84.43
73	128	1.0	0	1	1	0.96	13.01
74	132	2.0	0	1	1	18.40	29.90
75	132	1.0	0	1	1	9.24	17.05
76	160	6.0	0	1	1	67.20	111.63
77	192	6.0	0	1	1	80.64	100.27
78	160	5.0	0	1	1	56.00	103.45
79	160	6.0	0	1	1	67.20	69.94
80	160	2.0	0	1	1	23.52	40.63
81	192	5.0	0	1	1	67.20	99.16
82	160	5.0	0	1	2	145.60	202.17
83	132	2.0	0	1	1	18.40	20.11
84	192	5.0	0	1	1	67.20	90.10
85	160	1.0	0	1	1	11.20	19.14
86	320	5.0	0	1	1	112.00	173.66
87	128	2.0	0	1	1	17.92	33.42
88	224	4.0	0	1	2	163.07	197.63
89	192	6.0	0	1	1	80.64	132.09
90	132	1.0	0	1	1	9.24	17.50
91	196	4.0	0	1	1	54.00	100.90
92	192	6.0	0	1	1	80.64	91.36
93	224	6.0	0	1	1	94.08	114.24
94	144	4.0	0	1	1	40.32	54.27
95	160	6.0	0	1	1	67.20	119.56
96	152	5.0	0	1	1	53.20	62.07
97	192	4.0	0	1	1	53.76	95.44
98	223	2.0	0	1	1	31.22	44.32
99	132	1.0	0	1	1	9.24	10.82
100	192	4.0	0	1	1	53.76	92.20

101	288	6.0	0	1	1	99.78	172.40
102	256	4.0	0	1	1	71.68	130.63
103	196	2.0	0	1	1	27.44	46.66
104	132	1.0	0	1	1	9.24	17.19
105	224	6.0	0	1	1	94.08	101.63
106	224	5.0	0	1	1	78.40	81.29
107	160	6.0	0	1	1	67.20	99.09
108	196	4.0	0	1	1	54.32	80.40
109	160	6.0	0	1	1	67.20	105.55
110	160	4.0	0	1	1	44.80	70.00
111	192	6.0	0	1	1	80.64	97.15
112	160	5.0	0	1	1	56.00	79.61
113	296	3.0	0	1	1	62.16	101.90
114	128	2.0	0	1	1	17.92	30.97
115	56	1.0	0	1	2	7.75	14.24
116	192	5.0	0	1	1	67.20	109.72
117	160	6.0	0	1	1	67.20	80.36
118	160	1.0	0	1	1	11.20	21.44
119	192	5.0	0	1	1	67.20	92.17
120	192	5.0	0	1	1	67.20	87.30
121	224	6.0	0	1	1	94.08	194.15
122	68	1.0	0	1	1	3.62	5.32
123	130	1.0	0	1	1	9.10	11.95
124	160	6.0	0	1	1	67.20	60.42
125	192	6.0	0	1	1	80.64	94.21
126	160	6.0	0	1	1	67.20	100.67
127	135	3.0	0	1	1	20.35	40.41
128	256	3.0	0	1	1	53.76	60.63
129	224	6.0	0	1	1	94.08	96.01
130	192	5.0	0	1	1	67.20	75.15
131	224	6.0	0	1	1	94.08	120.04
132	192	4.0	0	1	1	53.76	60.20
133	192	6.0	0	1	1	80.64	155.14
134	160	4.0	0	1	1	44.80	46.92
135	224	6.0	0	1	1	94.08	142.30
136	224	6.0	0	1	1	94.08	151.93
137	224	4.0	0	1	2	183.07	177.94
138	320	4.0	0	1	2	232.96	320.03
139	256	6.0	0	1	1	107.52	119.63
140	192	6.0	0	1	1	80.64	80.74
141	224	5.0	0	1	1	78.40	142.91
142	192	5.0	0	1	1	67.20	73.67
143	224	4.0	0	1	2	163.07	192.10
144	192	5.0	0	1	1	67.20	73.94
145	128	1.0	0	1	1	8.96	10.06
146	128	2.0	0	1	1	17.92	24.32
147	192	5.0	0	1	1	67.20	91.04
148	160	4.0	0	1	1	44.80	82.43
149	160	4.0	0	1	1	44.80	82.93
150	160	6.0	0	1	1	67.20	83.69

151	194	4.0	0	1	1	54.32	109.66
152	192	6.0	0	1	1	80.64	122.20
153	128	1.0	0	1	1	8.96	11.09
154	192	4.0	0	1	1	53.76	82.35
155	256	2.0	0	1	1	53.76	64.50
156	300	5.0	0	1	1	133.00	147.51
157	96	4.0	0	1	2	69.09	74.60
158	324	6.0	0	1	1	136.00	107.69
159	224	6.0	0	1	1	96.00	149.37
160	224	6.0	0	1	1	96.00	131.10
161	224	6.0	0	1	1	96.00	132.69
162	224	5.0	0	1	1	70.40	106.30
163	192	4.0	0	1	1	53.76	90.96
164	160	5.0	0	1	1	56.00	80.30
165	128	1.0	0	1	1	8.96	19.70
166	304	5.0	0	1	1	134.40	194.51
167	192	6.0	0	1	1	82.32	101.19
168	224	6.0	0	1	1	96.00	107.74
169	160	6.0	0	1	1	67.20	129.00
170	56	1.0	0	1	2	7.75	12.66
171	224	4.0	0	1	2	163.07	270.16
172	192	6.0	0	1	1	80.64	141.90
173	160	6.0	0	1	1	67.20	67.61
174	74	1.0	0	1	2	10.24	11.60
175	96	1.0	0	1	2	17.47	20.09
176	160	5.0	0	1	1	56.00	109.69
177	224	3.0	0	1	2	122.30	166.31
178	160	6.0	0	1	1	67.20	122.61
179	223	2.0	0	1	1	31.22	43.13
180	160	6.0	0	1	1	67.20	129.67
181	224	6.0	0	1	1	96.00	121.69
182	160	6.0	0	1	1	67.20	122.01
183	224	6.0	0	1	1	96.00	129.39
184	225	6.0	0	1	1	96.50	179.21
185	128	1.0	0	1	1	8.96	9.20
186	128	2.0	0	1	1	17.92	26.36
187	292	6.0	0	3	1	77.41	149.66
188	132	1.0	0	1	1	9.24	11.46
189	164	3.0	0	1	1	34.44	66.26
190	224	6.0	0	1	1	96.00	150.60
191	292	3.0	0	1	1	62.16	100.13
192	192	6.0	0	1	1	80.64	112.02
193	160	6.0	0	1	1	67.20	92.33
194	160	4.0	0	1	1	44.00	92.99
195	224	4.0	0	1	1	62.72	69.12
196	132	2.0	0	1	1	10.40	20.99
197	192	4.0	0	1	1	53.76	109.13
198	224	5.0	0	1	2	203.04	340.09
199	96	5.0	0	1	2	87.36	100.55
200	160	4.0	0	1	1	44.00	61.09

201	192	4.0	0	1	1	53.76	103.09
202	220	6.0	0	1	1	95.76	113.02
203	224	1.0	0	1	2	40.77	79.41
204	160	5.0	0	1	1	56.00	72.03
205	224	5.0	0	1	1	78.40	102.18
206	160	5.0	0	1	1	56.00	74.54
207	132	2.0	0	1	1	10.40	39.99
208	160	6.0	0	1	1	67.20	120.25
209	132	2.0	0	1	1	10.40	36.00
210	160	6.0	0	1	1	67.20	100.69
211	128	2.0	0	1	1	17.92	26.67
212	192	5.0	0	1	1	67.20	110.93
213	224	6.0	0	1	1	95.76	97.92
214	220	6.0	0	1	1	95.76	137.33
215	160	5.0	0	1	1	56.00	89.70
216	132	2.0	0	1	1	10.40	32.10
217	224	6.0	0	1	1	94.00	170.55
218	224	6.0	0	1	1	94.00	145.01
219	224	5.0	0	1	1	78.40	131.33
220	132	1.0	0	1	1	9.24	12.14
221	128	1.0	0	1	1	8.96	10.27
222	60	1.0	0	1	1	3.62	7.01
223	128	2.0	0	1	1	17.92	23.41
224	224	4.0	0	1	1	62.72	77.92
225	160	4.0	0	1	1	44.00	84.12
226	160	5.0	0	1	1	56.00	70.31
227	132	1.0	0	1	1	9.24	17.60
228	160	4.0	0	1	1	44.00	64.41
229	160	6.0	0	1	1	67.20	110.94
230	224	6.0	0	1	1	94.00	174.71
231	224	6.0	0	1	1	94.00	99.37
232	112	1.0	0	1	1	7.04	11.46
233	160	4.0	0	1	1	44.00	92.07
234	192	6.0	0	1	1	80.64	102.41
235	224	4.0	0	1	2	163.07	222.09
236	192	5.0	0	1	1	67.20	74.61
237	224	6.0	0	1	1	94.00	100.73
238	224	6.0	0	1	1	94.00	121.42
239	132	2.0	0	1	1	10.40	33.57
240	132	2.0	0	1	1	10.40	32.46
241	192	5.0	0	1	1	67.20	92.49
242	132	3.0	0	1	1	27.72	40.61
243	224	6.0	0	1	1	94.00	101.96
244	160	4.0	0	1	1	44.00	51.09
245	224	6.0	0	1	1	94.00	105.31
246	160	4.0	0	1	1	44.00	83.08
247	128	1.0	0	1	1	8.96	10.89
248	224	5.0	0	1	1	78.40	146.22
249	160	6.0	0	1	1	67.20	119.77
250	160	6.0	0	1	1	67.20	127.30

251	130	1.0	0	1	1	9.10	12.22
252	132	2.0	0	1	1	18.40	20.78
253	224	4.0	0	1	2	163.07	273.79
254	232	5.0	0	1	1	81.20	140.47
255	224	5.0	0	1	1	78.40	115.98
256	192	6.0	0	1	1	80.64	100.99
257	224	4.0	0	1	2	163.07	319.93
258	224	6.0	0	1	1	94.08	114.03
259	320	6.0	0	1	1	134.40	140.41
260	128	2.0	0	1	1	17.92	23.01
261	224	6.0	0	1	1	94.08	104.09
262	128	1.0	0	1	1	8.96	16.29
263	20	1.0	0	1	2	2.77	3.96
264	130	1.0	0	1	1	9.10	13.29
265	160	6.0	0	1	1	67.20	109.07
266	232	5.0	0	1	1	81.20	108.56
267	192	5.0	0	1	1	67.20	120.30
268	320	5.0	0	1	1	112.00	196.42
269	160	4.0	0	1	1	44.80	61.71
270	230	1.0	0	1	1	16.52	26.40
271	320	6.0	0	1	1	134.40	175.49
272	160	6.0	0	1	1	67.20	131.36
273	224	6.0	0	1	1	94.08	106.10
274	224	6.0	0	1	1	94.08	118.10
275	192	6.0	0	1	1	80.64	115.67
276	160	3.0	0	1	2	87.36	109.33
277	224	6.0	0	1	1	94.08	160.32
278	224	5.0	0	1	1	78.40	145.69
279	160	5.0	0	1	1	56.00	109.59
280	64	1.0	0	1	1	3.48	4.44
281	320	6.0	0	1	1	134.40	200.01
282	232	1.0	0	1	1	16.24	31.00
283	160	6.0	0	1	1	67.20	80.40
284	128	1.0	0	1	1	8.96	9.32
285	250	5.0	0	1	1	89.60	143.03
286	224	6.0	0	1	1	94.08	101.30
287	224	6.0	0	1	1	94.08	95.10
288	320	6.0	0	1	1	134.40	240.00
289	320	5.0	0	1	1	112.00	170.61
290	132	1.0	0	1	1	9.24	14.24
291	224	5.0	0	1	1	78.40	106.06
292	160	4.0	0	1	1	44.80	87.91
293	192	6.0	0	1	1	80.64	97.70
294	192	5.0	0	1	1	67.20	95.06
295	320	5.0	0	1	1	112.00	130.22
296	224	5.0	0	1	1	78.40	123.41
297	160	5.0	0	1	1	56.00	60.01
298	128	1.0	0	1	1	8.96	12.69
299	224	6.0	0	1	1	94.08	166.90
300	164	6.0	0	1	1	68.88	83.50

301	224	6.0	0	1	1	94.08	122.00
302	192	6.0	0	1	1	80.64	157.21
303	128	1.0	0	1	1	8.96	14.05
304	228	1.0	0	1	1	15.96	30.15
305	160	4.0	0	1	1	44.80	92.15
306	132	2.0	0	1	1	18.48	25.93
307	132	1.0	0	1	1	9.24	13.49
308	160	4.0	0	1	1	44.80	49.61
309	192	6.0	0	1	1	80.64	109.21
310	128	3.0	0	1	1	26.88	45.00
311	296	2.0	0	1	1	41.44	85.23
312	164	6.0	0	1	1	68.88	102.10
313	224	3.0	0	1	1	47.04	74.23
314	128	2.0	0	1	1	17.92	20.10
315	228	3.0	0	1	1	47.88	86.31
316	160	6.0	0	1	1	67.20	100.13
317	97	1.0	0	1	2	17.65	20.47
318	320	6.0	0	1	1	134.40	244.29
319	192	5.0	0	1	1	67.20	100.90
320	160	5.0	0	1	1	56.00	84.00
321	129	1.0	0	1	1	9.03	16.00
322	160	4.0	0	1	1	44.80	97.09
323	160	3.0	0	1	1	33.60	40.97
324	128	2.0	0	1	2	46.59	80.72
325	192	6.0	0	1	1	80.64	161.22
326	160	2.0	0	1	1	22.40	24.56
327	128	2.0	0	1	1	17.92	21.70
328	132	1.0	0	1	1	9.24	16.23
329	160	5.0	0	1	1	56.00	80.90
330	224	5.0	0	1	1	78.40	92.30
331	160	4.0	0	1	1	44.80	79.13
332	192	5.0	0	1	1	67.20	105.49
333	192	6.0	0	1	1	80.64	142.91
334	192	5.0	0	1	1	67.20	90.97
335	160	6.0	0	1	1	67.20	87.09
336	132	1.0	0	1	1	9.24	17.64
337	224	6.0	0	1	1	94.08	179.71
338	160	5.0	0	1	1	56.00	101.96
339	224	5.0	0	1	1	78.40	83.09
340	132	3.0	0	1	1	27.72	51.05
341	228	5.0	0	1	1	79.80	81.16
342	160	4.0	0	1	1	44.80	50.13
343	192	4.0	0	1	1	53.76	74.03
344	160	5.0	0	1	1	56.00	101.39
345	160	4.0	0	1	1	44.80	57.03
346	192	6.0	0	1	1	80.64	144.59
347	135	2.0	0	1	1	18.90	33.14
348	224	6.0	0	1	1	94.08	114.51
349	128	2.0	0	1	1	17.92	10.96
350	192	6.0	0	1	1	80.64	89.07

351	160	5.0	0	1	1	56.00	79.97
352	160	6.0	0	1	1	67.20	100.49
353	160	3.0	0	1	1	33.60	65.16
354	224	6.0	0	1	1	94.92	124.51
355	192	6.0	0	1	1	80.64	90.95
356	160	6.0	0	1	1	67.20	128.29
357	232	5.0	0	1	1	81.20	149.07
358	160	6.0	0	1	1	67.20	132.52
359	132	2.0	0	1	1	10.40	32.12
360	138	2.0	0	1	1	19.04	34.39
361	198	4.0	0	1	1	54.00	102.06
362	132	1.0	0	1	1	9.24	13.26
363	224	6.0	0	1	1	94.00	110.96
364	192	5.0	0	1	1	67.20	122.13
365	132	1.0	0	1	1	9.24	10.30
366	192	6.0	0	1	1	80.64	113.70
367	160	3.0	0	1	1	33.60	51.70
368	192	6.0	0	1	1	80.64	129.19
369	192	6.0	0	1	2	209.66	310.32
370	160	4.0	0	1	1	44.00	45.64
371	132	2.0	0	1	1	10.40	22.34
372	132	2.0	0	1	1	10.40	33.76
373	160	6.0	0	1	1	67.20	72.07
374	160	2.0	0	1	1	22.40	29.60
375	184	6.0	0	1	2	113.57	104.51
376	120	4.0	0	1	2	93.10	104.51
377	64	2.0	0	1	1	6.01	11.71
378	224	5.0	0	1	1	70.40	134.00
379	132	1.0	0	1	1	9.24	14.27
380	160	3.0	0	1	1	33.60	43.72
381	320	5.0	0	1	1	112.00	196.13
382	192	5.0	0	1	1	67.20	132.95
383	192	6.0	0	1	1	80.64	117.95
384	160	5.0	0	1	1	56.00	100.33
385	160	4.0	0	1	1	44.00	69.10
386	224	5.0	0	1	1	70.40	112.70
387	224	6.0	0	1	1	94.00	109.75
388	132	1.0	0	1	1	9.24	17.52
389	140	4.0	0	1	1	41.44	59.20
390	120	2.0	0	1	1	17.92	30.93
391	192	5.0	0	1	1	67.20	77.62
392	132	3.0	0	1	1	27.72	52.97
393	133	1.0	0	1	1	9.31	15.69
394	120	5.0	0	1	2	116.40	150.64
395	132	1.0	0	1	1	9.24	13.09
396	160	4.0	0	1	1	44.00	83.94
397	120	1.0	0	1	1	8.96	11.55
398	96	1.0	0	1	2	17.47	29.12
399	224	5.0	0	1	1	70.40	140.36
400	160	4.0	0	1	1	44.00	50.70

401	128	1.0	0	1	1	8.96	14.70
402	128	2.0	0	1	1	17.92	20.63
403	128	2.0	0	1	2	46.59	78.62
404	224	6.0	0	1	1	94.08	178.10
405	132	2.0	0	1	1	18.48	36.27
406	97	2.0	0	1	2	35.31	49.20
407	160	3.0	0	1	2	67.36	136.11
408	131	1.0	0	1	1	9.17	12.06
409	192	6.0	0	1	1	80.64	148.27
410	160	5.0	0	1	1	56.00	93.30
411	192	6.0	0	1	1	80.64	127.94
412	146	2.0	0	1	1	20.44	40.10
413	160	4.0	0	1	1	44.80	70.32
414	56	1.0	0	1	2	7.75	7.03
415	128	1.0	0	1	1	8.96	9.46
416	160	2.0	0	1	1	22.40	27.29
417	133	1.0	0	1	1	9.31	9.55
418	138	1.0	0	1	1	9.66	9.95
419	160	5.0	0	1	1	56.00	92.05
420	132	1.0	0	1	1	9.24	9.64
421	224	6.0	0	1	1	94.08	173.98
422	160	5.0	0	1	1	56.00	108.93
423	228	5.0	0	1	1	79.80	138.71
424	160	5.0	0	1	1	56.00	91.97
425	192	5.0	0	1	1	67.20	134.18
426	192	6.0	0	1	1	80.64	122.30
427	256	4.0	0	1	1	71.68	103.69
428	192	6.0	0	1	1	80.64	145.05
429	224	6.0	0	1	1	94.08	163.38
430	128	2.0	0	1	1	17.92	18.95
431	160	3.0	0	1	1	33.60	59.60
432	160	3.0	0	1	1	33.60	51.90
433	133	2.0	0	1	1	18.62	29.22
434	160	6.0	0	1	1	67.20	123.03
435	192	4.0	0	1	1	53.76	86.35
436	192	6.0	0	1	1	80.64	85.70
437	160	2.0	0	1	1	22.40	38.20
438	132	1.0	0	1	1	9.24	17.16
439	160	4.0	0	1	1	44.80	69.01
440	192	6.0	0	1	1	80.64	107.02
441	160	6.0	0	1	1	67.20	87.20
442	160	4.0	0	1	1	44.80	67.29
443	184	4.0	0	1	2	75.71	108.69
444	224	5.0	0	1	1	78.40	80.77
445	380	4.0	0	1	1	106.40	201.73
446	160	5.0	0	1	1	56.00	97.05
447	131	2.0	0	1	1	18.34	20.44
448	160	4.0	0	1	1	44.80	69.39
449	160	5.0	0	1	1	56.00	109.45
450	192	6.0	0	1	1	80.64	127.46

451	160	9.0	0	1	1	56.00	103.20
452	224	6.0	0	1	1	94.00	101.39
453	144	1.0	0	1	1	10.22	17.41
454	192	6.0	0	1	1	80.64	191.09
455	164	9.0	0	1	1	57.40	114.21
456	192	5.0	0	1	1	67.20	94.40
457	132	1.0	0	1	1	9.24	10.74
458	192	9.0	0	1	1	67.20	84.09
459	160	3.0	0	1	1	33.60	39.10
460	232	4.0	0	1	1	64.96	82.50
461	192	4.0	0	1	1	53.76	95.01
462	160	3.0	0	1	1	33.60	47.94
463	232	2.0	0	1	1	32.48	61.72
464	160	4.0	0	1	1	44.80	44.00
465	304	6.0	3	1	1	193.54	346.79
466	129	1.0	0	1	1	9.03	17.40
467	144	1.0	0	1	1	10.08	10.60
468	224	6.0	0	1	1	95.76	169.20
469	160	3.0	0	1	1	33.60	45.74
470	320	6.0	0	1	1	134.40	190.10
471	160	4.0	0	1	1	44.80	61.19
472	130	1.0	0	1	1	9.66	10.06
473	320	5.0	0	1	1	112.00	222.42
474	224	6.0	0	1	1	94.08	130.04
475	192	6.0	0	1	1	80.64	129.51
476	139	1.0	0	1	1	9.73	12.07
477	132	1.0	0	1	1	9.24	13.27
478	224	3.0	0	1	1	47.04	90.94
479	160	3.0	0	1	2	87.36	112.39
480	132	4.0	0	1	2	96.10	110.09
481	144	1.0	0	1	1	10.22	15.19
482	160	4.0	0	1	1	44.80	47.03
483	160	3.0	0	1	1	33.60	30.33
484	224	4.0	0	1	1	62.72	73.00
485	160	6.0	0	1	1	67.20	92.93
486	132	3.0	0	1	1	27.72	36.01
487	130	1.0	0	1	1	9.10	15.65
488	132	1.0	0	1	1	9.24	12.39
489	304	6.0	3	1	1	193.54	379.65
490	160	6.0	0	1	1	67.20	182.31
491	96	1.0	0	1	2	17.47	26.94
492	164	4.0	0	1	1	45.92	51.40
493	224	2.0	0	1	1	31.36	45.63
494	192	6.0	0	1	1	80.64	85.06
495	130	1.0	0	1	1	9.10	15.00
496	139	1.0	0	1	1	9.73	19.44
497	320	6.0	0	1	1	134.40	263.99
498	224	4.0	0	1	1	62.72	63.20
499	120	6.0	0	1	2	139.70	273.94
500	304	6.0	3	1	1	193.54	369.46

BIOGRAPHIC NOTE

Paul Kodjo Niamkey was born May 17, 1954 in Abidjan (Republic of The Ivory Coast). He resided in Abidjan until 1963, then in Bordeaux (France) until 1970.

He graduated in 1972 from the Lycee Classique d'Abidjan; from the Université d'Abidjan in 1975 with a Bachelor of Mathematics degree.

He has been a student at Lehigh University since January 1976 when he began graduate studies in Computer Sciences.