Multiprogramming under a Page on Demand Strategy

JOHN L. SMITH
The University of Michigan, Ann Arbor, Michigan*

A model of multiprogramming for a particular computer system using a page on demand strategy is developed. Analysis of this model is used to predict performance (measured by the average usage of the CPU) when user programs are typical of those arising from an interactive time sharing environment. The effect of several hardware modifications is also analyzed. A parameter, readily calculated from the hardware characteristics and the program statistics, is proposed for gauging the effect of multiprogramming.

1. Introduction

A point of contention currently exists on the best strategy for handling the loading of programs and the sharing of the high speed memory in large time sharing computer systems. The problem is most critical and most difficult to resolve when the system is processing a heavy load of programs which execute for relatively short intervals. As this situation can be expected to arise frequently in systems interactively serving a large number of users, an analysis of the performance of a particular computer subsystem has been made.

The techniques of multiprogramming and paging have been proposed as means for efficiently adapting a computer system to an interactive type of load. The implementation of these techniques requires a considerable hardware investment for the handling of dynamic relocation [1], and the best control policy remains to be determined. One mode of paging [2] has been termed the single page loading strategy, the aim being to obtain a conservation of the high speed memory by loading pages from secondary memories only on demand. Another mode of paging is to load only entire segments of programs. Both of these loading strategies are generally proposed in conjunction with multiprogramming, so that execution may be switched to another user program when one program requires a page (or segment) to be transferred from a secondary memory. The question which has been raised is whether the characteristics of programs and the necessary system overhead will permit sufficient overlapping of fetching and execution for an improvement in the system performance to be obtained by these techniques.

In a recent paper by Fine, Jackson, and McIsaac [3], some relevant statistics on the dynamic behavior of a particular set of programs were reported. These programs were typical of those run in interactive mode on the Q-32 time sharing system at the System Development Corporation. On the basis of these statistics, the authors concluded that there was considerable doubt about the worth of the page on demand strategy. Certainly any suggestion that it would be useful to multiprogram a few pages from many programs in the high speed memory was negated by these statistics.

The work reported here partly consists of a description and analysis of a stochastic model of the important operations in multiprogramming with a page on demand strategy. The model is used to investigate the multiprogramming of a particular system under full load conditions. One aspect of the results concerns the application of the statistics of Fine et al., to the model in an effort to obtain further substantiation of their conclusions. The performance of the system when loaded by programs having markedly different statistics is also analyzed. In each case the effect of several hardware modifications is demonstrated.

2. The Model

While there are many additional hardware components in a large time sharing computer system, a typical hardware configuration for the subsystem relevant to this study is shown in Figure 1. A schematic description of the operation of this computer subsystem is given in Figure 2. There are two service functions represented in the model: (1) execution of programs; (2) transfer of pages between the

*Systems Engineering Laboratory, Department of Electrical Engineering, College of Engineering. This research was supported by contract AF 30 (602)-3953 with the US Air Force, Rome Air Development Center, Rome N. Y.

drum memory and the high speed memory. The queueing and service disciplines associated with these functions, and the simplifying assumptions and approximations used, are now described.

It is assumed that programs are loaded into the high speed memory under a page on demand strategy, with a standard initial three page allocation. Two classes of user programs (class 1 and class 2), which differ in the statistics of their execution intervals and page demands, are identified. The model allows for a maximum of three programs to be multiprogrammed in the high speed memory.

It is well to explain the significance of an execution interval. This pertains to the period of execution allocated to a particular program in effecting the time sharing of the CPU. Because of random interactions with the user, only a fraction of all program execution intervals are terminated by a time interrupt (see Figure 3). As a result of the page on demand strategy, each execution interval is divided into a number of phases. An execution phase occurs between successive calls for new pages from the drum, and between the last page call and the termination of the execution interval.

Eligible programs queue for execution by the CPU. For each class of programs there are two parameters of the model, μ and $\lambda(x)$, which describe the length of an execution phase. A subscript will be used to identify the parameters belonging to each class of programs. The parameter μ defines a negative exponential distribution function describing the execution interval for a program. It is assumed that each execution interval is an independent random variable. Hence,

$$\text{probability that an execution interval} \leq t = 1 - e^{-\mu t} \quad (1)$$

where $1/\mu$ is the average execution interval. The function $\lambda(x)$ defines a nonhomogeneous Poisson process, which describes the occurrence of demands for new pages by a program during its execution interval. For a program which has already activated y pages, the average interval between successive page requests is $1/\lambda(y)$. Hence

$$\text{probability that the interval between successive page requests} \leq t = 1 - e^{-\lambda(x)t}. \quad (2)$$

Information on the drum memory is organized into recording fields, consisting of a number of tracks, which can be read or written in parallel by fixed heads. In general each field is composed of m sectors, each sector having a storage capacity of one page of information. Upon the generation of a page transfer request, the request is assigned to a subqueue associated with the sector containing the required page. Unit time for the model is the time taken for the transfer of one page from the drum ($1/m$ revolution time), and time is synchronized to the passage of page headers under the read heads. One can envisage a rotation of the drum as effecting periodic service, in the order 1, 2, $\cdots$, m, to each subqueue. Each service interval is of one time unit and, if the corresponding subqueue is not empty, a page transfer will occur; otherwise there will be no transfer during that time unit.

For the purpose of modeling the operation, two important assumptions have been made. Firstly, each page requested for transfer from the drum is assumed to be located with equal probability ($1/m$) in any drum sector. Secondly, it is assumed that the output of pages to the drum from the high speed memory produces no delay in the input of pages to the high speed memory from the drum. The latter assumption is valid in cases in which a vacant page position is maintained in the high speed memory and the usage factor for the drum is slightly below 100%, so that there is usually a vacant page position in each drum sector. Hence a useful model need only treat the input of pages from the drum to the high speed memory.

In the Appendix, the following approximation to the page transfer process, in which the average transfer capacity of the drum I/O channel combination is exactly reproduced, is derived. Every page transfer request, present at the beginning of a time unit, has some probability of being serviced during that time unit. In particular the

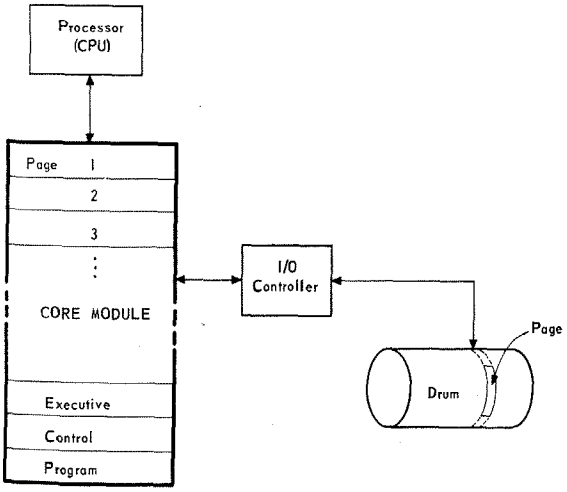

FIG. 1. A computer subsystem

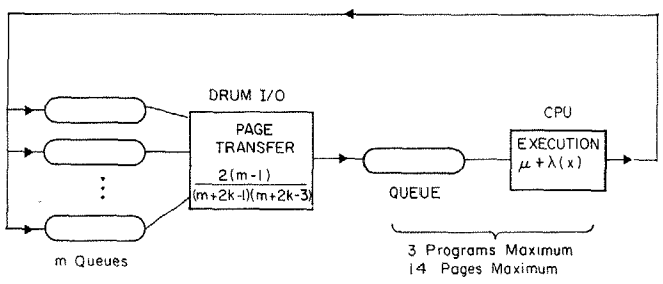

FIG. 2. The model of multiprogramming and paging

kth request to occur (out of all those present) has probability

$$\frac{2(m-1)}{(m+2k-1)(m+2k-3)}$$

of transfer during the time unit.

It was the intention in developing this model to identify class 2 programs with those jobs having longer execution intervals and greater high speed storage demands. The following priorities were also implemented in the model. In choosing the next program to use the CPU, an eligible (able to execute) class 2 program is given priority over any eligible class 1 program. Likewise the program with the larger high speed memory commitment is given priority over another program of the same class, when both are eligible to execute. The same priority structure was also implemented in the operation of the drum I/O channel.

The admission of preemption in the drum I/O subqueues does not change the rate of completion of page transfer requests from that resulting from a first-in first-out (FIFO) queue discipline. In the real system, however, the expected additional waiting time of preempted requests will be dependent upon their elapsed waiting time. Thus there is a slight discrepancy in the representation described above if we equate the variable k to the priority of a request, when this differs from the request's order of arrival. This point may be clarified by referring to the derivation in the Appendix. No compensation for this feature was made in the model, but calculations of the expected completion times of requests, under all possible preemptions which can occur in the particular system, show the model to give close approximations. As the most important property to be represented for this analysis is the drum channel transfer capacity, the approximation is justifiable.

The main limitations to the scope of the model are: (1) there is no representation of the CPU time or the I/O channel usage devoted to system overhead, (2) there is no representation of the competition which might arise between the CPU and the I/O channel for access to the high speed memory, (3) the number of programs and the number of pages that can be handled are limited by the analysis method, (4) a condition of system overload is assumed; that is, class 1 and class 2 programs are assumed to be always awaiting processing when the high speed storage is available.

3. Relating the Model to an Actual System

The mathematical descriptions which have been incorporated in the model ensure that it is tractable for numerical analysis and optimization procedures. The results quoted in this paper were obtained using the analysis program RQA-1, developed by Wallace and Rosenberg [4]. Some representative values for the parameters of the model are now derived by referring to the statistics taken by Fine et al. [3]. These statistics are the result of detailed measurements taken on 182 sample execution intervals of five programs.[1] Their adaptation to the model serves as a test of the fitness of the probability distributions assumed for the program characteristics.

In Figure 3 the cumulative frequency function of the number of instructions executed by a program during one scheduled execution interval is shown. The discontinuity at 80,000 instructions is due to the system imposed quantum of 400msec of execution time, which was equivalent to the average execution time for 80,000 instructions. Also plotted in Figure 3 is a negative exponential probability distribution having the same mean (20,400 instructions) as the statistics. It is obvious that the negative exponential is a very poor fit to the cumulative frequency function of these statistics. Therefore in view of the characteristics of the model, it is useful to divide these statistics into two classes:

Class 1: Samples in which a program executes 1000 instructions or less

Class 2: Samples in which a program executes more than 1000 instructions.

There are 91 samples in each class. Figure 4 shows the cumulative frequency functions of the number of instructions executed per execution interval for each class, to-

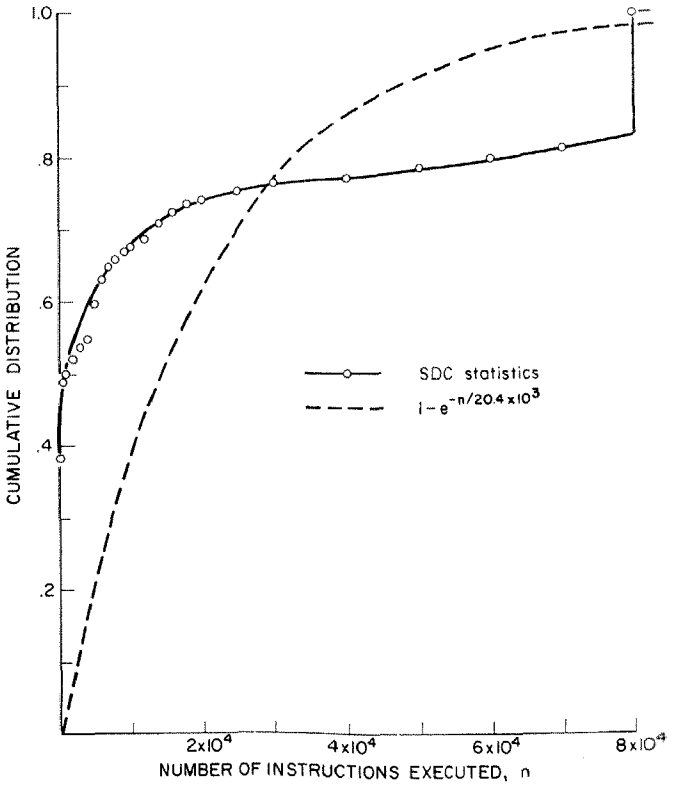

FIG. 3. Cumulative distribution of the number of instructions executed per execution interval

[1] It should be remembered, however, that these statistics were not taken from a paged system.

gether with a negative exponential probability distribution having the same mean. The mean for class 1 is 88 instructions and for class 2 the mean is 40,700 instructions.

It is proposed that by using the above classifications the negative exponential distribution is a tolerable approximation for the particular statistics.[2] In appraising this approximation the following feature must be considered in addition to the fit of the curves. It is implied by the model that all program execution intervals are independent random variables described by the negative exponential distribution. It is apparent from additional statistics, not quoted here, that consecutive execution intervals for the same program are not completely independent. On this point it can be argued that if the system is processing several different programs the combined sequence of execution intervals should be almost independent, despite some correlation between execution intervals for a particular program.

A second statistic of the five programs is displayed in Figure 5. Here the average number of instructions executed between requests for a new page is plotted against the number of pages already activated during the execution interval. From the available statistics little can be said on whether the process of new page requests approximates a Poisson process or not. An important feature of the model, however, is that the parameter $1/\lambda(x)$ can be adjusted so that the model conforms exactly to the functional behavior depicted in Figure 5. This should be an overriding factor in achieving correspondence between the model and the physical system.

At this stage we must take into account one restriction which exists in any computer system—the capacity of the high speed memory. Here we need only consider the capacity available to user programs, which enforces an upper bound on the total number of active pages which can be held in the high speed memory. It is highly probable that in a multiprogrammed environment, individual programs will be forced to swap pages between the drum and the high speed memory because the total number of active pages of all the programs in the high speed memory has exceeded this bound. (In the model the bound is 14 pages and Figure 5 shows that swapping is likely to occur even when there is only one program using this size of memory. We shall see, however, that the effect of larger high speed memories can be analyzed with the model.) In modeling a program's demand rate for page transfers after swapping first occurs it will be assumed that the rate remains constant at the value in effect when the first swap occurred—unless more pages of the high speed memory become available and the program is allowed to expand into these.

While it is not appropriate in this paper to present the details of the analysis methods, a brief comment on the limitations is necessary. The complexity of the model proposed is considerable from an analysis viewpoint; never-

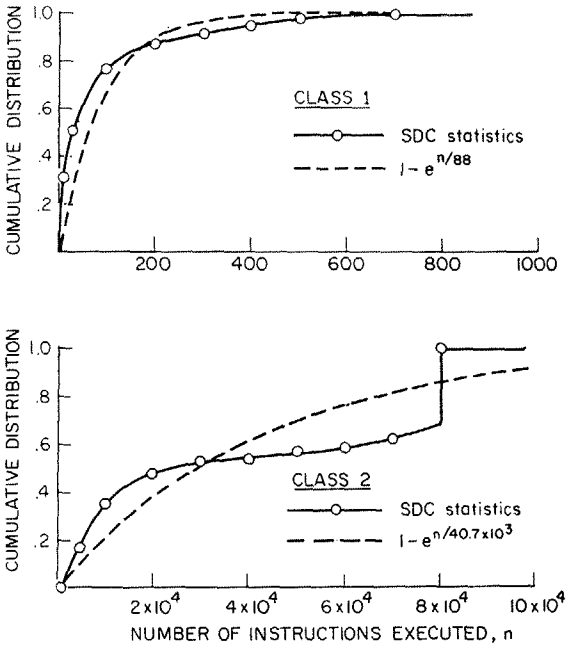

FIG. 4. Cumulative distributions for each class of execution interval

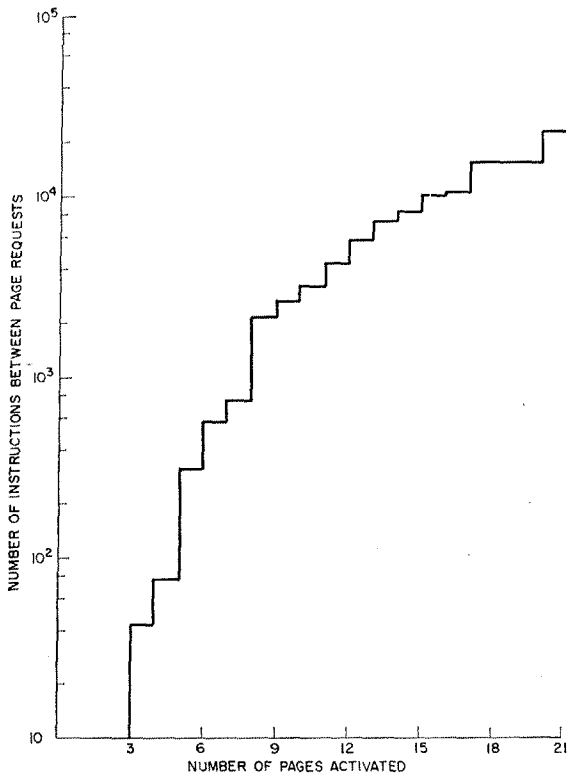

FIG. 5. Average page demand statistics

[2] This also suggests that a two-phase hyperexponential distribution could be used for all execution intervals.

theless, all the results to be presented were obtained by precise numerical solution of the equations of the mathematical model. The size of the high speed memory available for user programs in the model was limited to 14 pages in order to restrict the computation required in the analysis. Larger high speed memories were represented by redefining the physical page size and then appropriately modifying the value of the abscissa used to determine the page demand rate from Figure 5, and also modifying the page transfer time. This representation implied that any restrictions arising from the location of particular instruction sequences and data in different pages were ignored. Another limitation is that the attainable solutions describe the stationary behavior of the model; that is, the analysis method is restricted to measuring performance parameters such as the long run average CPU usage by programs. The executive control policies which can be considered are therefore stationary, for example, those that maintain a constant mix of class 1 and class 2 programs in the high speed memory. Because of the manner in which class 1 and class 2 programs have been identified with the particular statistics, stationary mixes of these classes would not occur in the real system. However, the performance measures which we obtain by analyzing various multiprogramming mixes should be a good indication of the performance obtained in practice.

4. A Gauge for Multiprogramming

By reference to the system model proposed in Section 2 and the type of program statistics quoted in Section 3 it is possible to identify the critical mechanism of multiprogramming (under the particular operating conditions defined). Programs using the high speed memory act as a source of page transfer requests. These requests queue for service from the drum I/O channel. Therefore in an analogous manner to a single queue–single server system, a utilization factor ρ (see [6, p. 17]) for the drum I/O channel can be defined:

$$\rho = \frac{\text{rate of occurrence of page transfer requests}}{\text{rate of page transfer completions}}.$$

With few exceptions, the minimum number of page transfer requests which can occur over each execution interval is equal to the number of pages activated during the interval. This number will be augmented, however, if the high speed memory limitations necessitate swapping. Therefore the rate of occurrence of page transfer requests is a function of the following program and system variables:

(i) the number of instructions executed per execution interval,
(ii) the cycle time of the high speed memory,
(iii) the page demand function (cf. Figure 5),
(iv) the maximum number of pages of the high speed memory available to each program.

The rate of page transfer completions is dependent on the hardware characteristics:

(i) the drum revolution time,
(ii) the number of pages per drum field, m.

It follows that ρ is directly related to the ratio of the speed of the high speed memory to the speed of the drum. In expressing ρ as a function of the parameters of the model this ratio is incorporated in the time normalization implied in the definition of μ, $\lambda(x)$, and the page transfer process. The derivation is given below.

Using eqs. (1) and (2) and defining y to be the maximum number of pages of high speed memory which will be allocated to each program, the average number of page transfers per execution interval is given by:

$$\begin{aligned} N &= 1 + \frac{\lambda(1)}{\lambda(1)+\mu} + \cdots + \prod_{x=1}^{y-1} \frac{\lambda(x)}{\lambda(x)+\mu} \\ &\quad + \prod_{x=1}^{y} \frac{\lambda(x)}{\lambda(x)+\mu} \cdot \left\{1 - \frac{\lambda(y)}{\lambda(y)+\mu}\right\}^{-1} \\ &= 1 + \sum_{z=1}^{y-1} \prod_{x=1}^{z} \frac{\lambda(x)}{\lambda(x)+\mu} + \prod_{x=1}^{y-1} \frac{\lambda(x)}{\lambda(x)+\mu} \cdot \frac{\lambda(y)}{\mu}. \end{aligned}$$

The rate of occurrence of page transfer requests is then given by $N\mu$. In the Appendix it is demonstrated that the rate of page transfer completions is dependent on the number of requests present. It follows from eq. (6) that for k requests this rate is

$$\frac{2k}{m + 2k - 1}$$

and therefore

$$\rho \approx N\mu \frac{(m + 2k - 1)}{2k}.$$

For efficient multiprogramming ρ should not be greatly in excess of 1, or else frequent queueing delays in the I/O channel will occur. A small utilization factor would not necessarily indicate poor performance in terms of CPU usage, it merely indicates low I/O channel usage, which at worst is idle equipment.

In Section 5, ρ is evaluated for several situations to indicate its correlation with multiprogramming performance. Because the parameter k varies during the operation of a system, the convention of using its maximum value (the number of programs being multiprogrammed) in the evaluation of ρ is adopted. Also, some approximation will be involved in specifying the value of the parameter y in each situation.

In order to illustrate the sensitivity of ρ to the value of k, consider a typical case in which $m = 4$ and three programs are being multiprogrammed. The rate of page transfer completions may vary between 2.5 for $k = 1$ and 1.5 for $k = 3$. Accounting for this dynamic variation would not be significant in the results to be quoted here, but some applications may call for a more accurate formula for ρ. Appendix G of [5] indicates some approaches to this problem.

5. Analysis of the Model Using the SDC Statistics

In order to complete the parameter specifications for the model, the operating speeds of the drum and the high speed memories must be defined. The following figures were selected as being representative of current hardware performance.

Mean instruction execution time	2.5μsec
Drum speed	3,000 rpm
Page size	4 pages/rev.

It follows that the transfer time for one page (the time unit) is 5msec, and $m = 4$. If it is assumed that the average number of instructions executed during class 1 and class 2 execution intervals is 100 and 40,000, respectively, then

$$\frac{1}{\mu_1} = \frac{100 \times 2.5 \times 10^{-6}}{5 \times 10^{-3}} = 0.05 \text{ time units}$$

$$\frac{1}{\mu_2} = \frac{40 \times 10^{3} \times 2.5 \times 10^{-6}}{5 \times 10^{-3}} = 20 \text{ time units.}$$

Here the subscript denotes the class of programs which the parameter describes. Also, with reference to Figure 5, the average number of instructions executed between page calls, when a program has already activated eight pages, is 2,160 instructions. Therefore

$$\frac{1}{\lambda_1(8)} = \frac{1}{\lambda_2(8)} = \frac{2160}{2000} = 1.08 \text{ time units.}$$

Similarly all the values in the sequence $\lambda_1(3), \lambda_1(4), \cdots, \lambda_2(3), \cdots$, can be evaluated.

The control policies for which the model was analyzed are defined in Table I. The first two policies do not involve multiprogramming, the remainder do. All the policies impose upper bounds on the number of physical pages of the high speed memory which each program may use. It is implicit that the class of each program is identifiable in advance.

In Table II three different system configurations are defined, and in Table III the results of the model analysis for each configuration and for each control policy are given. In all cases the performance parameter is the average CPU usage for each class of programs. The model parameters derived at the beginning of this section correspond to the first case listed in Table II. Case II involves a change in the page transfer rate corresponding to ten times the original drum speed. While the same capacity drum running at 30,000 rpm may not be feasible it is of interest to analyze this case. Case III corresponds to doubling the size of the high speed memory and this change is modeled by doubling the physical page size in both the high speed memory and the drum. The results given are for $m = 4$. This implies a larger drum in terms of the number of bits per track. It should be noted that the maximum page allocation limits given in Table I also refer to double size pages in this case.

Since the same type of high speed memory has been assumed in each case, the average CPU usage can be used for direct comparison of the rate at which execution intervals are completed. In all cases, the multiprogramming of programs having class 1 execution intervals (policies 2A and 3A) produces a significant improvement in the CPU usage, with most of the improvement possible by maintaining just two programs in the high speed memory. If the system is only processing programs with class 1 execution intervals, the elapsed time for each execution is solely determined by the drum speed. In case II where the drum speed is 10 times that of case I, the CPU usage is also increased by a factor of 10.

TABLE I. STATIONARY MULTIPROGRAMMING POLICIES

Policy	Program mix		Maximum page allocation	
	Class 1	Class 2	Class 1	Class 2
1A	1	···	14	···
1B	···	1	···	14
2A	2	···	7, 7	···
2B	1	1	7	11
2C	···	2	···	7, 7
2D	···	2	···	7, 11
3A	3	···	5, 5, 6	···
3B	2	1	5, 5	6
3C	2	1	5, 5	8

TABLE II. HARDWARE CONFIGURATIONS

	High speed memory	Drum	Average execution time/instruction
I	14 pages	3,000 rpm, $m = 4$	2.5μsec
II	14 pages	30,000 rpm, $m = 4$	2.5μsec
III	28 pages	3,000 rpm, $m = 4$	2.5μsec

TABLE III. AVERAGE CPU USAGE FOR STATIONARY MULTIPROGRAMMING

Policy	I			II			III		
	Class 1	Class 2	ρ	Class 1	Class 2	ρ	Class 1	Class 2	ρ
1A	.005	···	210	.049	···	21.0	.008	···	159
1B	···	.314	1.88	···	.822	.188	···	.467	1.08
2A	.008	···	148	.077	···	14.8	.011	···	111
2B	.003	.261	3.44	.011	.778	.344	.004	.433	1.51
2C	···	.158	5.51	···	.751	.551	···	.609	0.89
2D	···	.220	3.50	···	.819	.350	···	.607	0.94
3A	.009	···	124	.087	···	12.4	.012	···	95.7
3B	.005	.070	11.7	.049	.449	1.17	.008	.353	2.69
3C	.004	.181	3.79	.025	.698	.379	.007	.405	2.13

Multiprogramming programs having class 2 execution intervals seriously degrades performance in case I but produces a significant improvement in performance for case III (compare policies 2C and 2D with policy 1B). In case II there is a slight degradation in performance. A detailed examination of the performance figures suggests that, overall, in case I multiprogramming would lead to poorer performance, and there is little advantage to be gained by using multiprogramming in case II. In case III, however. there seems to be considerable gain possible. This last conclusion is based on the following estimates.

The statistics show that although class 1 and 2 execution intervals will occur randomly, they occur with equal frequency in the SDC system. Therefore the execution times devoted to the two classes of execution intervals are in the proportion 0.25msec to 100msec (these times being the average execution intervals for the two classes). Now if no multiprogramming were employed we have:

average elapsed time for 100msec execution, class 2	= 100/0.467
	= 214msec
average elapsed time for 0.25msec execution, class 1	= 0.25/0.008
	= 31.3msec
average elapsed time for one class 1 and one class 2 execution interval is then	= 214 + 31.3
	≈ 245msec

Now considering the continual multiprogramming of two programs, it has been indicated at the conclusion of Section 3 that in the real situation the composite of programs in the high speed memory will fluctuate between two class 1 execution intervals, one class 1 and one class 2 execution interval, and two class 2 execution intervals. Using the performance figures for stationary mixes we can make the following observations:

If two class 1 execution intervals occur together,

average elapsed time for 0.25msec execution, class 1	= 0.25/0.011
	= 22.7msec

If two class 2 execution intervals occur together,

average elapsed time for 100msec execution, class 2	= 100/0.609
	= 164msec

If one class 1 and one class 2 execution intervals occur together,

average elapsed time for 0.25msec execution, class 1	= 0.25/0.004
	= 62.5msec
average class 2 execution time during an elapsed time of 62.5msec	= 62.5 × 0.433
	= 27.1msec

Combining these elapsed times in any desired manner we find that the average elapsed time for one class 1 and one class 2 execution interval is in the range 182msec to 187msec. This is approximately a 25% reduction on the time calculated for the same amount of processing with no multiprogramming.

Because of the inherent approximations in using the closed form expression for ρ, the scale of values obtained for each case in Table III do not correspond exactly. Nevertheless the results indicate that this parameter, which is easily estimated from program statistics and hardware parameters, is a useful gauge for multiprogramming.

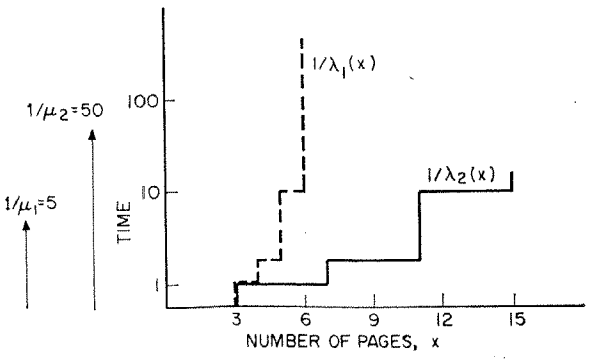

FIG. 6. Parameter set A

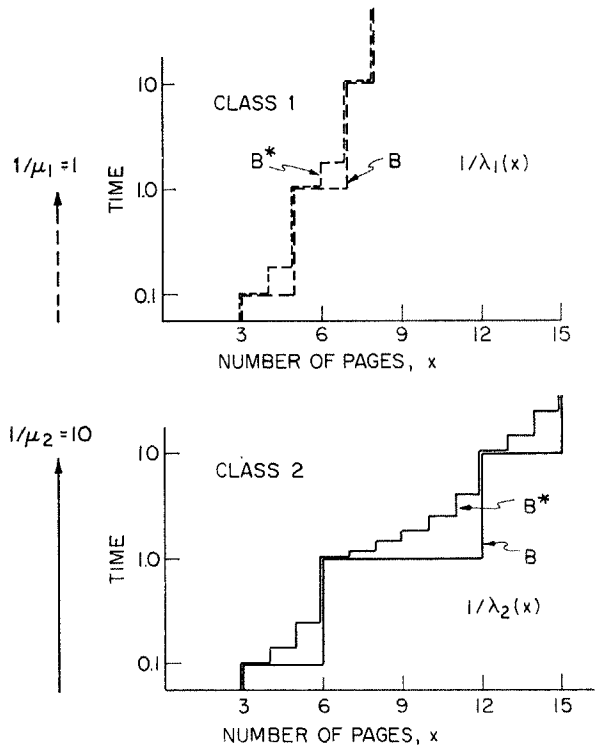

FIG. 7. Parameter set B

6. Further Analysis

The program statistics obtained by Fine et al., are representative of a class of programs run on a particular time sharing system. The statistics of the complete population of programs run on any system may differ markedly from these. In this section, in order to demonstrate the potential of multiprogramming and the effect of some hardware variations, some conditions more favorable to multiprogramming will be assumed, without firm justification for the accompanying program statistics. However, if program codes and data were organized for optimum performance in a paging environment, program behavior might be more closely represented by these statistics.

The results to be discussed can be reproduced, in a sense, for a multitude of model parameter values. It is sufficient for presentation of the significant points to use just two basic sets of parameter values. These sets of parameters labeled A and B are described in Figures 6 and 7, respectively. It is not necessary to interpret the parameter values in terms of absolute core and drum speeds. There is a continuum of absolute values corresponding to each parameter set, and this adaption of the model has already been demonstrated.

Both parameter sets have the characteristics postulated in Section 2 concerning the two classes of programs (or execution intervals) which employ the CPU. The average execution intervals are in the ratio 10:1 for the two classes, and the class 1 programs exhibit a much more rapid degradation in page demand rates. The page demand rates are higher and the average execution intervals shorter for parameter set B than for parameter set A. The drum organization assumed is four pages per field. We shall see that, in the system described by the model, it is generally suitable to multiprogram two or three programs defined by these parameter sets. Presumably, if page demand rates were considerably higher, or execution intervals much shorter, than in these parameter sets, the conditions would not be suitable for multiprogramming.

In Figure 7 an alternate set of page demand functions has been plotted and labeled B*. These functions correspond to a linear interpolation between the discontinuities in $\lambda_1(x)$ and $\lambda_2(x)$ for the parameter set B. This illustrates the flexibility of the model concerning the page demand functions which can be represented.

The performance of the system under parameter set A is described in Figures 8 and 9. Also plotted in these figures is the performance measure for the following six modifications to the basic drum configuration implied by parameter set A: (1) twice the drum speed; (2) half the drum speed; (3) twice the number of read heads per drum field ($m=8$); (4) half the number of read heads per drum field ($m=2$); (5) modifications (1) and (4); (6) modifications (2) and (3).

The multiprogramming of up to three class 1 programs has been analyzed and it seems obvious that this situation is very favorable for CPU usage (see Figure 8). In

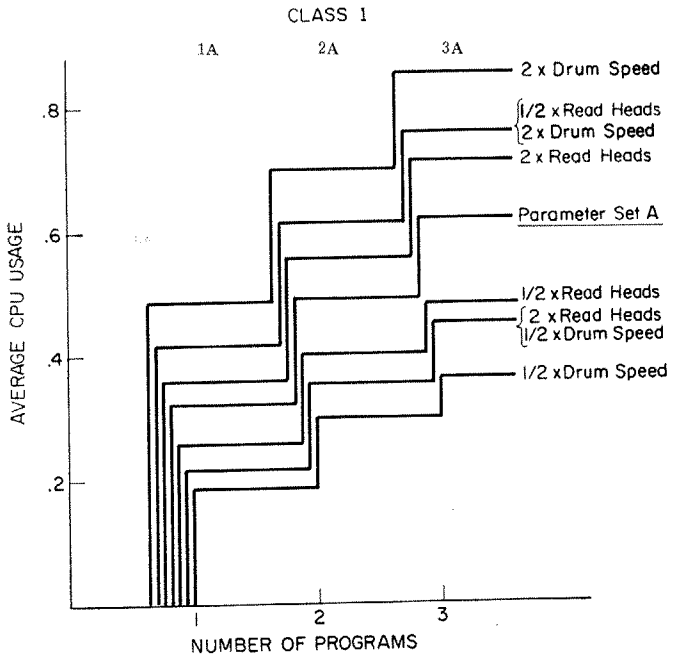

FIG. 8. Performance for the multiprogramming of class 1 programs from parameter set A

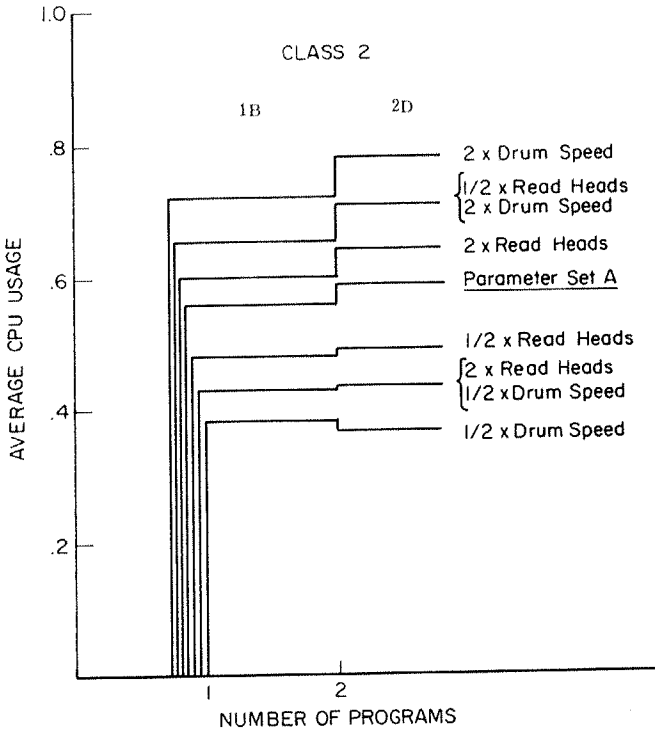

FIG. 9. Performance for the multiprogramming of class 2 programs from parameter set A

the ranges considered there is an almost linear increase in performance with drum speed and with the number of read heads per drum field. Performance is most sensitive to the drum speed, which determines the delay experienced by each program, on its making a page transfer request, until it becomes eligible to execute again. The effect of increasing the number of read heads is to allow more subqueues for page transfer requests, thereby increasing the expected number of page transfers per drum revolution for a given number of requests present (that is, assuming uniform page distribution). When there is only one program using the high speed memory the only effect of varying the number of pages per drum revolution is to modify the latency delays, and this is a less significant factor.

It is worthwhile to point out at this stage that modification of drum speed can be interpreted as a modification (in the opposite sense) of the average execution time per instruction, which in turn can be interpreted, with some approximation, as a modification to the cycle time of the high speed memory. Thus halving the drum speed may be considered equivalent to halving the cycle time of the high speed memory, in its effect on the average CPU usage. The point which must then be borne in mind is that, if the cycle time is halved, programs will require only half the execution time. Therefore, while our performance parameter is a gauge for measuring the effectiveness of multiprogramming, it cannot be used as a comparison of the rate of completion of program execution intervals in this case.

From Figure 9 we see that the conditions are not as favorable for multiprogramming class 2 programs from parameter set A as for multiprogramming class 1 programs. If the drum speed is halved, performance actually decreases with two programs in core memory. In the light of the comments in the previous paragraph, it follows that if the cycle time were halved it would also be disadvantageous to multiprogram two class 2 programs.

Referring to the corresponding results for parameter set B (Figures 10 and 11), similar comments can be made for variations in the drum speed. The importance of page demand rates is also illustrated. The effect of the modification to page demands, described by the parameter set B*, is readily anticipated. These parameters have a somewhat lower average demand rate than parameter set B. The difference is most noticeable in the multiprogramming of two class 2 programs. For example, under policy 2D, one program is allowed a maximum of 11 pages and the demand rates $\lambda_2(7), \cdots, \lambda_2(11)$ differ significantly for the two parameter sets.

Similar effects could be observed if the model were analyzed for stationary mixes of class 1 and class 2 programs; however, all the major results for the multiprogramming of a fixed number of programs have been illustrated. In [5] the case where the number of programs using the high speed memory changes according to certain loading strategies is examined. This can give an advantage over multiprogramming a fixed number of programs, but under stationary conditions the advantage over multiprogramming the optimal fixed number of programs is not great. The need for nonstationary control policies can also be demonstrated; this need and also some models subject to closed form solution are treated in [5].

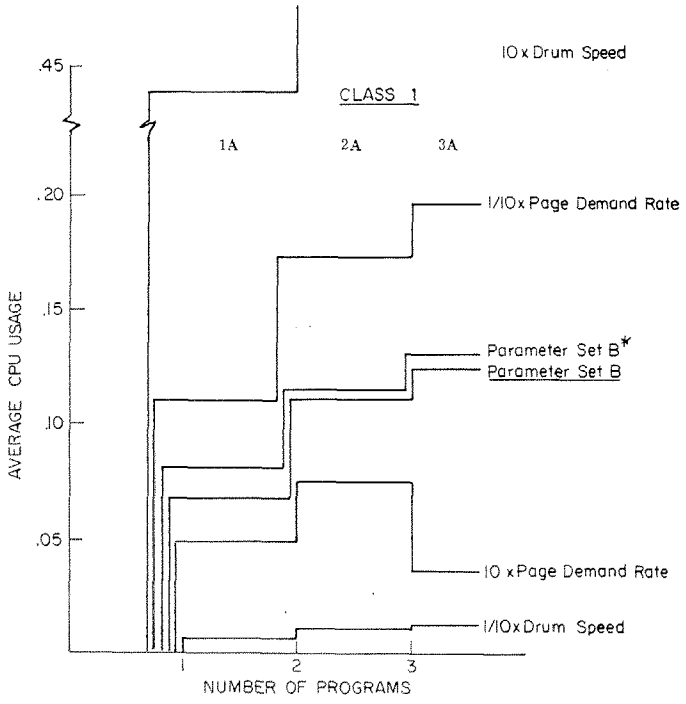

FIG. 10. Performance for the multiprogramming of class 1 programs from parameter set B

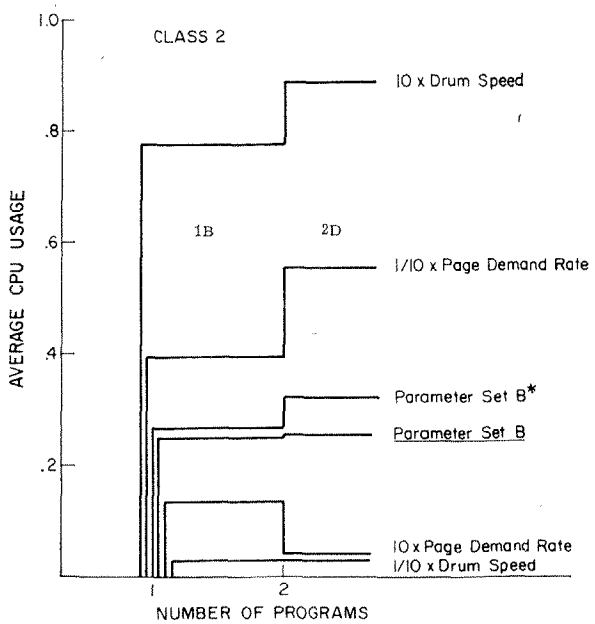

FIG. 11. Performance for the multiprogramming of class 2 programs from parameter set B

7. Conclusion

The most significant feature, not dealt with in this investigation, is the system overhead involved in both the executive control program and the implementation of paging and multiprogramming. Present indications are that this overhead is considerable and its effect would be to moderate any improvement attributed here to multiprogramming. Nevertheless, analysis of the model has provided useful insight into the potential of multiprogramming and paging, and the results could be modified to take account of system overhead.

Categoric conclusions cannot be made from the limited set of statistics described in Section 3, but as they have been recorded from a time sharing interactive environment they are appropriate. Analysis of the model for the range of parameters corresponding to these statistics indicates that a conservative outlook for multiprogramming using a page on demand strategy must be maintained. If a large high speed memory is available for user programs, however, it does seem that there is some advantage (dependent on system overhead) to be gained from multiprogramming.

Specifically, the SDC statistics have described paging for a fixed page size of 1024 words of 48 bits. The model has demonstrated that it is fortuitous to attempt multiprogramming with only 14 pages available for the user programs. With 28 pages available to user programs, a 25% margin of improvement has been calculated for the multiprogramming of two programs. It is likely that the method we have used to obtain the results for a 28 page memory is a little favorable to the page on demand strategy, because doubling the page size will not halve the number of page calls. Therefore a slightly larger memory may be necessary to obtain the improvement cited.

The necessary operating conditions for the success of multiprogramming have been identified. The reason for the failure of multiprogramming, in those cases analyzed, was that programs were delayed too frequently and for too long a period, either because of their own short execution intervals or because they were constrained to execute in a small high speed memory area. The most important characteristic of the hardware is the *relative* speed of the high speed memory and the drum accessing process. All these factors are reflected in the channel utilization factor, ρ, which has been proposed as a gauge for multiprogramming under a page on demand strategy.

Acknowledgments. The author is indebted to Professor B. Arden of the University of Michigan for helpful discussions on this work, and to G. Fine, C. Jackson, and P. McIsaac for making available their measurements taken at the System Development Corporation. Appreciation is also due to the referee for his very helpful criticism.

RECEIVED APRIL 1967; REVISED JUNE 1967

REFERENCES

1. GIBSON, C. T. Time sharing on the IBM System/360: Model 67. Proc. AFIPS 1966 Spring Joint Comput. Conf., Vol. 28, pp. 61–78.
2. ARDEN, B. W., GALLER, B. A., O'BRIEN, T. C., AND WESTERVELT, F. H. Program and addressing structure in a time-sharing environment. *J. ACM 13*, 1 (Jan. 1966), 1–16.
3. FINE, G. H., JACKSON, C. W., AND MCISAAC, P. V. Dynamic program behavior under paging. Proc. 21st ACM National Conf., Washington, D. C., 1966, ACM Publ. P-66, pp. 223–228.
4. WALLACE, V. L., AND ROSENBERG, R. S. R.Q.A. −1, the Recursive Queue Analyzer. Technical Report No. 2, Sys. Eng. Lab., U. of Michigan, Ann Arbor, Mich., Feb. 1966.
5. SMITH, J. L. Markov decisions on a partitioned state space, and the control of multiprogramming. Technical Report No. 9, Sys. Eng. Lab., U. of Michigan, Ann Arbor, Mich., April 1967.
6. MORSE, P. M. *Queues, Inventories and Maintenance.* John Wiley & Sons, Inc., New York, 1958.

APPENDIX

The details of the model of the drum and I/O channel operation are given below.

Let us consider a request which arrives to find n prior requests, labeled 1, 2, $\cdots$, n, waiting for service. In order to include those cases in which some additional information is known about the locations of the pages associated with these n requests, let p_j^k denote the probability that the kth request concerns a page displaced j sectors from the current origin of the read heads.

Hence

$$\sum_{j=0}^{m-1} p_j^k = 1, \qquad k = 1, 2, \cdots, n. \quad (3)$$

Then measuring time from the beginning of the first time unit after the $(n+1)$ request arrives, and assuming each subqueue is serviced according to a first-in first-out discipline, the expected time until the completion of this most recent request can be expressed as

$$\begin{aligned} E(t_{n+1}) &= \sum_{j=0}^{m-1} \frac{1}{m} \left\{ \left(\sum_{k=1}^{n} p_j^k m \right) + j + 1 \right\} \\ &= (m + 2n + 1)/2. \end{aligned} \quad (4)$$

It is of interest to observe that this time is independent of all p_j^k.

The Markov chain approximation to the behavior of the drum I/O channel now proposed has a state description which is based on the number of page transfer requests queued at the beginning of a time unit. The requests are assigned priority according to their order of arrival, and the probability that the jth priority request will have its page transferred in the ensuing time unit is denoted by p_j. Therefore the completion time for the highest priority request is described by a geometric distribution, with mean $1/p_1$, since during each time interval it has probability p_1 of having its page transferred. If the jth priority request completes, all lower priority requests k $(k > j)$ assume priority $k - 1$ in the ensuing time unit. In order

to satisfy (4) we have

$$p_1 = 2/(m + 1).$$

The distributions for the completion time of the lower priority requests are not as simple. However it is an easy matter to show (see [5, Appendix E]) that

$$E(t_{n+1}) = \frac{n}{\sum_{j=1}^{n} p_j} \tag{5}$$

and so in order to satisfy (5) we have

$$\sum_{j=1}^{n} p_j = \frac{2n}{m + 2n - 1}. \tag{6}$$

Solving (6) in order for $p_1, p_2, \cdots$, yields

$$p_k = \frac{2(m - 1)}{(m + 2k - 3)(m + 2k - 1)} \tag{7}$$

thus defining the model parameters for any number of requests. From (6) it follows that

$$\lim_{n\to\infty} \sum_{j=1}^{n} p_j = 1$$

which is in accordance with the maximum capacity of the I/O channel—one page transfer per unit time interval.

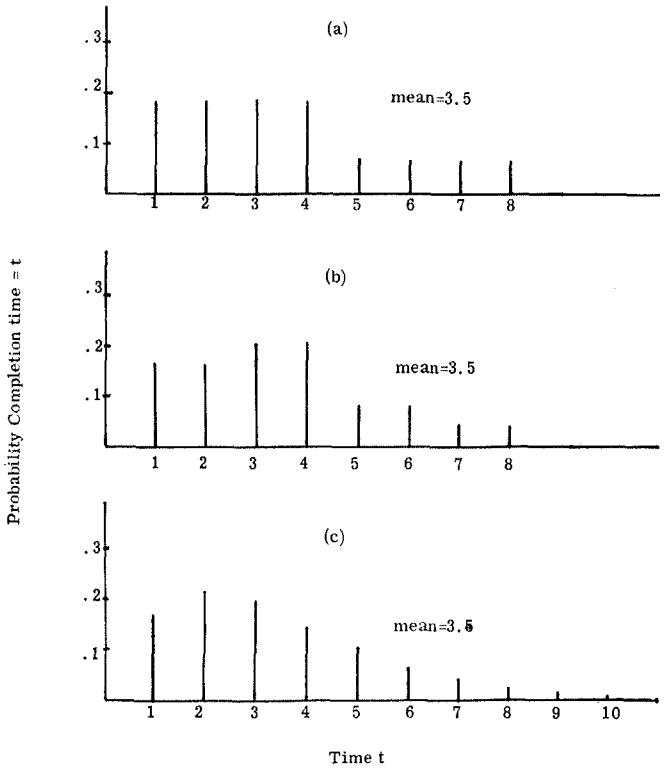

FIG. 12. Probability density functions for page transfer completion time.

It can also be seen from (5) that

$$E(t_{n+1}) = n, \qquad \text{for } n \text{ large.}$$

The representation described above is a very tractable Markov model, because of its simple state description. It should be noted that this model only describes the completion time (waiting time + transfer time) of a request from the beginning of the first time unit after the request is generated. The additional latency delay, experienced by each request, is accounted for exactly in the discrete time model of Section 2.

When there is only one program using the high speed memory, only one page transfer request can occur, and so in the Markov model the completion time of a page transfer request is described by a geometric distribution with mean $(m + 1)/2$. Of course, in the actual system this time is a random variable, uniformly distributed over the range $1, 2, \cdots, m$. However, the average CPU usage derived from the model of Section 2 is only dependent on the first moment of the distribution, and hence it is the same for either distribution.

When more than one page transfer request can occur the higher order moments of the distribution affect the average CPU usage. If the use of the drum I/O channel is heavy, however, so that each new page transfer request arrives to find other requests already waiting, then the distribution resulting from the Markov chain approximation is close to reality. This is illustrated by the probability density functions plotted in Figure 12, which correspond to the case of a request which arrives when one other request is waiting, and $m = 4$. Figure 12(a) is the density function for the completion time of the second request if no additional information is known about the location of the page associated with the first request. However, the location of the page associated with the first request is dependent on the history of previous page transfers and the elapsed waiting time of the request. Figure 12(b) is the density function for the completion time of the second request, a priori that there is twice the probability of the page associated with the first request being located in the next two sectors to rotate under the read heads. Figure 12(c) is the density function yielded by the Markov chain approximation derived in this Appendix.

From Figure 12(c) it can be seen that one approximation, implicit in the Markov model, is the assignment of a small probability to completion times longer than those which can possibly occur in the real system. In general, the variance of the resulting distribution is larger but the mean is always the same as for the real system. The effect of this decrease in the "orderliness" (see Morse [6]) of the drum I/O operation is to increase the likelihood of queueing delays, which in turn would reduce the mean CPU usage derived from the model of Section 2. But we have indicated that the model becomes close to reality as the use of the drum I/O channel increases, and therefore, it should provide reliable results.