W. F. King III
S. E. Smith
I. Wladawsky

Effects of Serial Programs in Multiprocessing Systems

Abstract: A model of a multiprocessing, multiprogramming computer system with serially reusable programs was developed to study the effect of serial programs on system performance. Two strategies for implementing serially reusable programs were investigated, a *wait* strategy in which the processor waits until the serial program is available, and a *switch* strategy, in which the processor is freed to do other work. Relative performances and asymptotic conditions as functions of the number of processors, processes, serially reusable programs, and the fraction of time each process executes serially reusable programs were obtained. Quantitative results are presented showing that the switch strategy is superior. The wait strategy causes quick saturation when the number of processes is increased.

Introduction

In any multiprocessing, multiprogramming system [1] there are parts of the operating system code which cannot be executed concurrently by more than one processor, e.g., the mechanism that assigns processors to tasks and the mechanism that assigns page frames to tasks. This paper studies the effect of these serially reusable sections of code on the performance of multiprocessor systems.

To maximize the amount of concurrency possible within the operating system, the entire system can be partitioned into a collection of component programs. Each portion of the system which must be executed serially is made a separate program. We denote these as *serial programs* (SP). The paging algorithm and the dispatcher or scheduler are examples of serial programs found in typical multiprocessor operating systems. Programs that can be executed concurrently by more than one processor are denoted as *concurrent programs* (CP).

Two fundamentally different strategies are available to enforce the serial execution (one processor at a time) of the serial programs. First, we assume that a *lock* or *semaphore* is associated with each serial program.

1. A process [2] attempting to execute an SP will test the lock. If the lock is not set, then the process will set it, execute the SP, and finally reset the lock. If the lock was set, the process and its associated processor will wait for the lock to be reset. Because the processor waits with the process, this strategy is called the *wait* strategy.
2. Same as 1 except that if the lock was set, the process waits for the lock but the processor is freed to do other available work. Since the processor is switched upon encountering a set lock, this strategy is called the *switch* strategy.

It is clear that in the wait strategy, the time a process is waiting for an SP is wasted time for the associated processor. With the switch strategy, this wasted time can be avoided by switching the processor to another process, if there is another process that is ready to continue. The extent to which the switch strategy is superior (in terms of throughput) to the wait strategy depends on the availability of ready processes and the overhead in switching processors among processes. The availability of ready processes depends on the level of multiprogramming and the characteristics of the processes, e.g., frequency of I/O operations and mean time between page faults. The process-switching overhead is very much machine dependent, and we assume an idealized switch strategy in which switching can be accomplished in zero time. This is taken into account when we compare the performance of the two strategies.

In the remainder of this paper we investigate and compare the throughput of systems using either the wait or the switch strategy. A simplified model of multiprocessing, multiprogramming systems is developed and analyzed using standard Markov analysis techniques. Parameters of the model include the number of processors, the number of SP's, the level of multiprogramming, and the

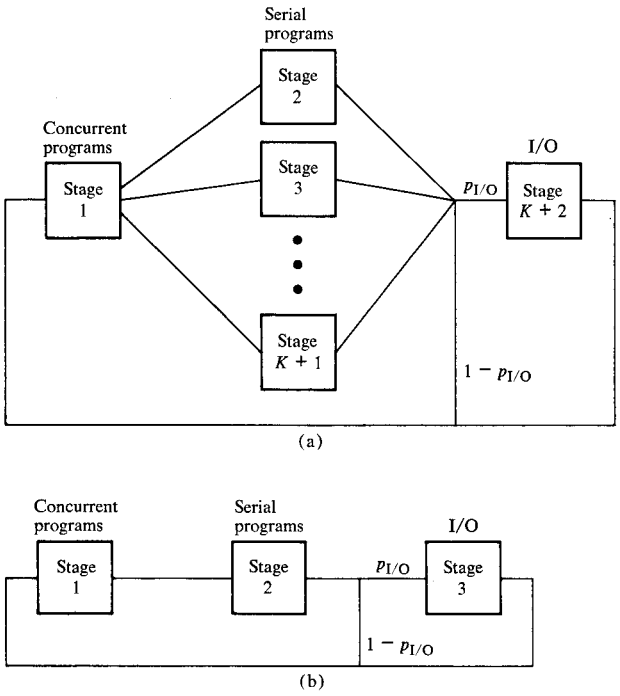


Figure 1 (a) Model of a multiprocessing system with serial programs; (b) approximate model of the same system.

times processes spend in different parts of the system. The throughputs with the two strategies are compared as functions of these parameters. A more detailed description of this work can be found in an earlier internal report [3].

Model

Consider an operating system with K SP's. Processes in the system can be in one of five types of activities:

1. Executing an SP.
2. Executing a CP.
3. Waiting for an SP.
4. Waiting for an available processor.
5. Performing an I/O operation.

We assume there are a fixed number of processes and processors in the system. When a process completes its action, we assume that another takes its place immediately. Figure 1 depicts the flow of processes among the activities. After a process completes a CP, it selects an SP to execute. All SP's are equally likely to be selected, i.e., the probability of selecting a given SP is $1/K$. If the selected SP is busy, then the process must wait until it becomes available. After completing the SP, the process either attempts to execute another CP or initiates an I/O operation (with some fixed probability). After completing an I/O operation, a process attempts to execute a CP. Any number of CP's can be executed simultaneously if processors are available.

The allocation of processors depends on the particular strategy. For the wait strategy, a processor accompanies a process from a CP to an SP and remains with the process until completion of the SP. It is then freed and can be assigned to another process to execute a CP.

For the switch strategy, if the SP is busy, the processor is freed and does not wait with the process. If the SP is not busy, the processor remains with the process and begins executing the SP. After completion of an SP, the processor is assigned to the next process waiting for that SP, if any; otherwise, the processor is available to be assigned to another process to execute a CP. This implies that any SP which has waiting processes also has an assigned processor. The assumption underlying this allocation strategy is that the SP's are potential system bottlenecks and hence should have higher priorities than CP's.

The following parameters are used to characterize the model:

N = number of processes
M = number of processors
K = number of SP's
$A \cdot M$ = average time to execute a CP
$B \cdot M$ = average time to execute an SP
C = average time to perform an I/O operation
$p_{I/O}$ = probability that an I/O operation follows an SP.

The execution times contain the factor M in order to have the total processing power remain constant as M increases. We assume that each processor in an M-processor system operates at $1/M$ times the speed of a single processor in a uni-processor system. As a result, the execution times in an M-processor system are M times what they would be in a system with one fast processor.

Markov analysis

To obtain analytical solutions for the behavior of the model and in order to make the analysis tractable we found it necessary to make the usual assumption that the times to execute a CP, an SP, and an I/O operation are each independent, identically distributed, random variables with exponential distributions with mean values $A \cdot M$, $B \cdot M$, and C, respectively. The model of the previous section then becomes a closed queuing network with $K+2$ stages (and queues). The queuing network is as shown in Fig. 1(a). The state of the system is a $(K+2)$-tuple $(n_1', n_2', \cdots, n_{K+2}')$, where n_i' is the number of processes in stage i, either waiting or in service. This queuing network differs from others which have been investigated [4] in that the number of processors at each stage is not fixed. One could write the set of equilibrium rate equa-

tions and solve them for the stationary state probabilities [5]. However, the large number of states involved makes this approach impractical except for very small values of K and N.

To reduce the number of states we approximate the queuing network of Fig. 1(a) by the three-state network shown in Fig. 1(b). The first and third stages are the same as in the original network. The second stage approximates the behavior of the K SP's in the original model. It is a multiserver stage in which the number of processes in service is given by a function of K, M, and the number of processes in the second stage. Let $n_i, i = 1, 2, 3$, represent the number of processes in stage i of the approximate network. The number of processes in service in the second stage is given by

$$\min [S(K, n_2), M], \tag{1}$$

where

$$S(K, n_2) = K\left[1 - \left(\frac{K-1}{K}\right)^{n_2}\right]. \tag{2}$$

The function $S(K, n_2)$ describes the contention caused by n_2 processes competing for K SP's. Each of the n_2 processes within the second stage is considered to have requested one of the K SP's with equal probability; $S(K, n_2)$ is the expected number of busy SP's which have been requested by at least one process. We originally derived $S(K, n_2)$ by first finding an expression $P(K, n_2, m)$ for the probability that exactly m SP's are busy. We then found the expected number of busy SP's from the sum $\Sigma_m\ m\ P(K, n_2, m)$. The following argument yields the same result for $S(K, n_2)$ in a simpler and intuitive manner.

Assume that n_2 processes are in the K SP stages in the original model. For a given stage, the probability that the stage is idle is

$$\left(\frac{K-1}{K}\right)^{n_2}. \tag{3}$$

Hence, with probability

$$1 - \left(\frac{K-1}{K}\right)^{n_2} \tag{4}$$

the stage is busy and, therefore, the expected number of busy stages is

$$K\left[1 - \left(\frac{K-1}{K}\right)^{n_2}\right]. \tag{5}$$

For a given N, M, and K and a given strategy, the state definition for the approximate network is the pair (n_1, n_2). The remaining $N - n_1 - n_2$ processes are in the third stage (I/O activity). The allocation of processors to stages 1 and 2 and the number of processes in service in each stage is determined by (n_1, n_2) and the strategy. The service times of a process in stages 1, 2, and 3 are still assumed to be independent, identically distributed, exponential random variables with means $A \cdot M$, $B \cdot M$, and C, respectively.

Because of the greatly reduced number of states, it is possible to analyze the behavior of this approximate model using standard Markov techniques. The equilibrium equations can be constructed using the rates of transition from one state to another. The Appendix contains the state transition rates for the two strategies, and the equilibrium equations can be solved for the stationary state probabilities.

The numerical studies which follow compare the wait and switch strategies in terms of throughput (processes completed per unit time) of the respective systems. The performance measure is defined as the ratio of the observed throughput to the maximum throughput obtained when all processors are 100 percent utilized, and this is taken to be the ratio of the expected number of processors doing useful work to the total number of processors in the system (M). These values are easily computed from the probabilities of the states and the number of processors doing useful work in each state.

Let $p_{n_1 n_2}$ be the stationary probability that the system is in state (n_1, n_2). For the switch strategy, if the system is in state (n_1, n_2), then $\min[M, S(K, n_2)]$ processors are busy in stage 2 and $\min[n_1, M - \min[M, S(K, n_2)]]$ are busy in stage 1. Thus, the throughput for the switch strategy is

$$T_s = \frac{1}{M} \sum_{(n_1, n_2)} p_{n_1 n_2} \{\min[M, S(K, n_2)] + \min[n_1, M - \min[M, S(K, n_2)]]\}. \tag{6}$$

For the wait strategy, although n_2 processors are in stage 2, only $S(K, n_2)$ are doing useful work. Of the $M - n_2$ processors in stage 1, $\min[n_1, M - n_2]$ are busy. Hence, the throughput for the wait strategy is

$$T_w = \frac{1}{M} \sum_{(n_1, n_2)} p_{n_1 n_2} \{S(K, n_2) + \min[n_1, M - n_2]\}. \tag{7}$$

Results

• *Comparison of throughputs*

For $N \leq M$, processes do not have to wait for processors in either strategy and we therefore expect T_s and T_w to be identical. On the other hand, if $N > M$, the switch strategy should yield a higher throughput because processors encountering locks can be reassigned to other processes.

An APL program was written to compute the stationary state probabilities and the throughput for each strategy. A number of runs were made with different combinations of the parameters K, N, M, A, B, C, and $p_{I/O}$. Figure 2

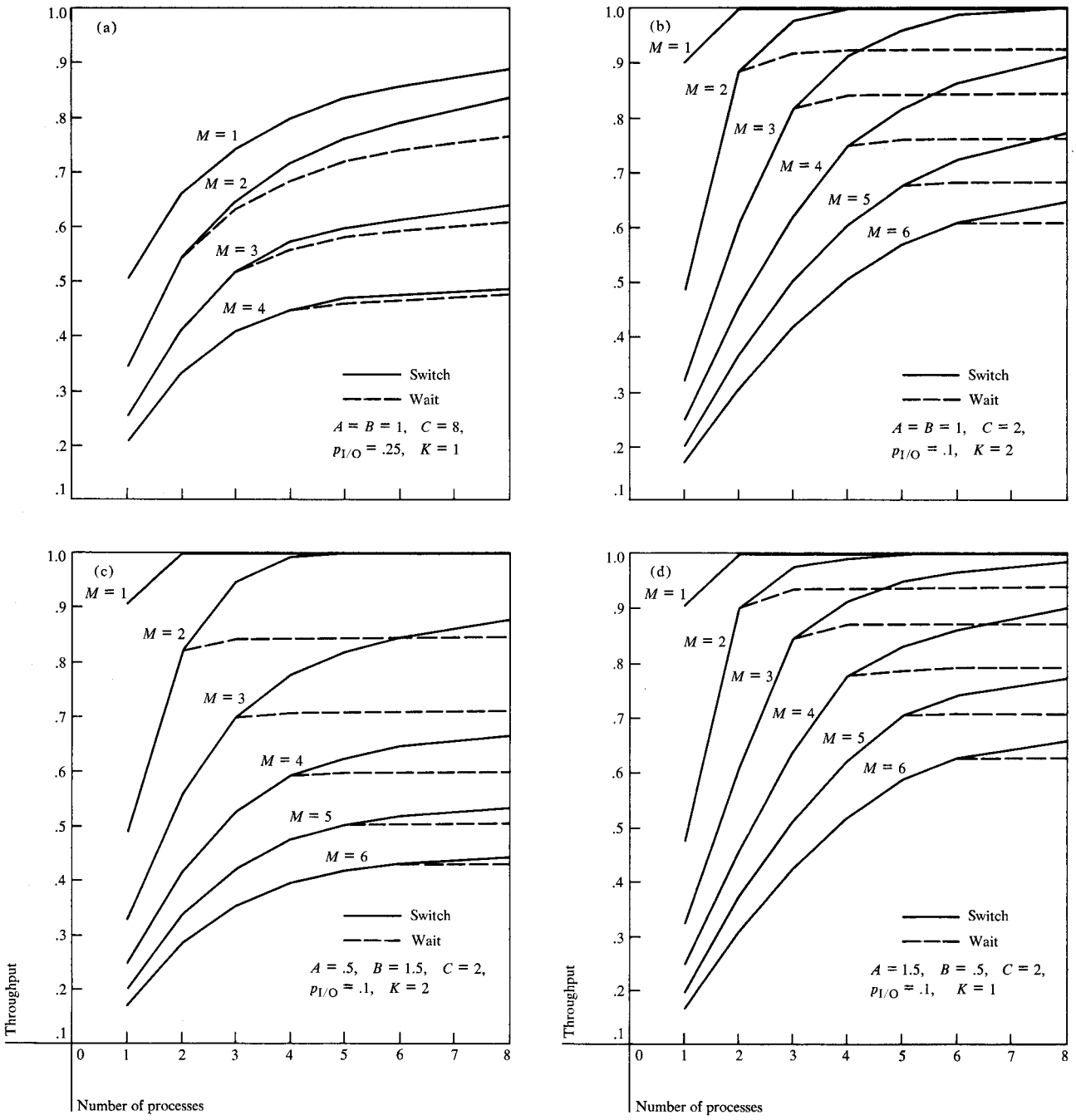


Figure 2 Wait- and switch-strategy throughputs as functions of several parameters.

shows the variation of throughput with the number of processes for various combinations of the other parameters. The general shape of the curves is as predicted above.

• *Asymptotic behavior of the wait strategy*

After a certain point the throughput of the wait strategy becomes independent of the level of multiprogramming, N, and approaches a constant value, $T_{w\infty}$. This is because with the wait strategy, once N is large enough so that $n_1 + n_2$ is always greater than M, increasing N has no effect because all processors are already allocated to processes.

We now derive an expression for $T_{w\infty}$ for those situations in which the I/O stage is not the system bottleneck. As N becomes large, the second stage of the model

can be analyzed as a single server whose rate of arrival and rate of service are functions of the number of processes waiting or in service. The stationary state probabilities for such systems can be computed [6] and these give us the probabilities $p(n_2)$ that we have exactly n_2 processes in the second stage [3]. Using these probabilities one can form the following expression for the asymptotic throughput of the wait strategy:

$$T_{w\infty} = \frac{\bar{S}}{M}\left(1+\frac{A}{B}\right) = \frac{K}{M}\left(1+\frac{A}{B}\right)\left[1-\left(\frac{K-1}{K}\right)^{n_2}\right] \quad (8)$$

$$= \frac{K}{M}\left(1+\frac{A}{B}\right)\left[1-\sum_{n_2=0}^{M}\left(\frac{K-1}{K}\right)^{n_2} p(n_2)\right]. \quad (9)$$

If the I/O stage is the system bottleneck, the throughput will be much more sensitive to the service time of the I/O stage than to the scheduling strategy.

• *Asymptotic behavior of the switch strategy*

One would expect that the switch strategy should be capable of taking advantage of additional processes since processors encountering locks can be switched to other processes. However, we verify in this section that there are cases in which the throughput of the switch strategy saturates at a value less than one. An example of circumstances leading to this property is the case when $K < M$ and all processes are waiting or executing SP's.

The following simple argument yields an upper bound on the steady state throughput. Let n_1^* and n_2^* be the average numbers of busy processors doing useful work in stages 1 and 2, respectively. From the definition of throughput given above, the average throughput is

$$T = \frac{n_1^* + n_2^*}{M}. \quad (10)$$

For either strategy in the steady state, the average rate of processes completing the first stage must equal the average rate of processes completing the second stage; i.e.,

$$\frac{n_1^*}{A \cdot M} = \frac{n_2^*}{B \cdot M}. \quad (11)$$

This makes it possible to express the throughput in terms of only n_1^* or n_2^*:

$$T = \frac{n_1^*}{M}\left(1+\frac{B}{A}\right) = \frac{n_2^*}{M}\left(1+\frac{A}{B}\right). \quad (12)$$

For both strategies,

$$n_2^* \le \bar{S}(K, n_2) = K\left[1-\left(\frac{K-1}{K}\right)^{n_2}\right] \le K. \quad (13)$$

Thus, we have the following upper bound on T:

$$T \le \frac{K}{M}\left(1+\frac{A}{B}\right). \quad (14)$$

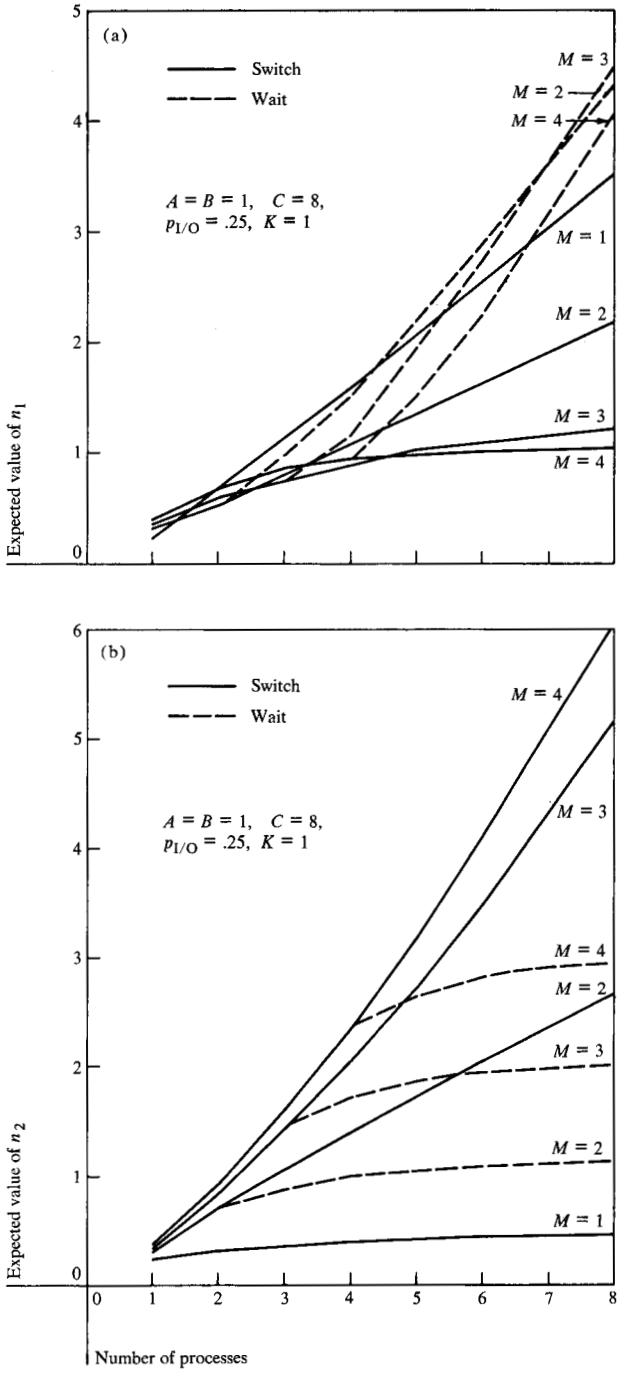


Figure 3 Number of processes in (a) stage 1 and (b) stage 2 as functions of the total number of processes.

Our numerical results (Fig. 2) indeed show that as N becomes large, T_s asymptotically approaches

$$\min\left[1, \frac{K}{M}\left(1+\frac{A}{B}\right)\right], \quad (15)$$

as predicted above.

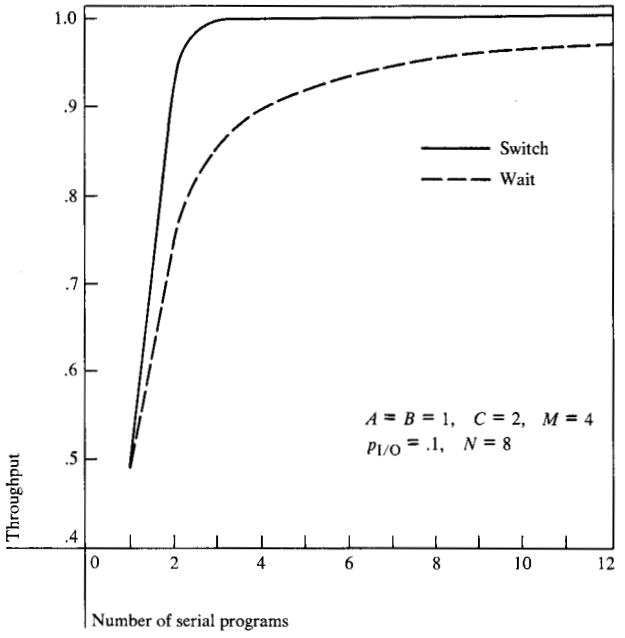
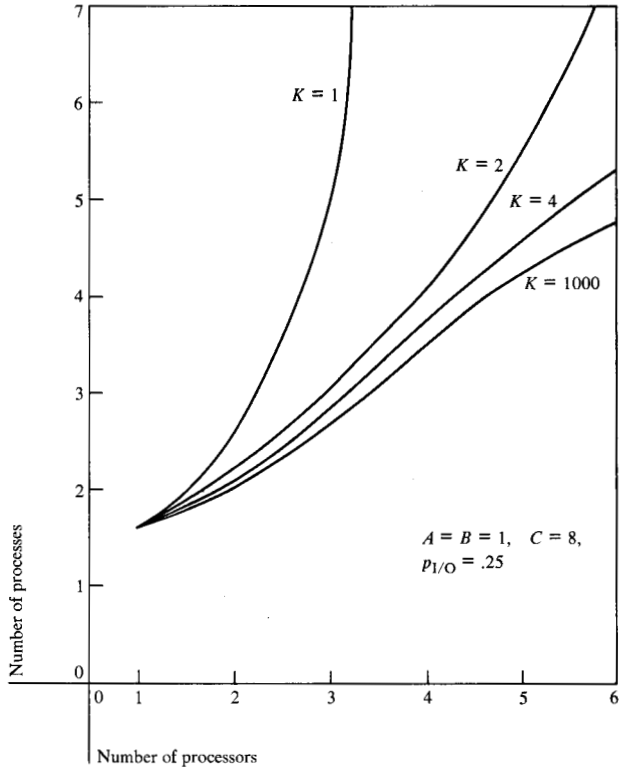


Figure 4 Effect of the number of serial programs on the throughput.

Figure 5 Interdependence of the numbers of processes, processors, and serial programs for a constant throughput.



Comparing equations (9) and (15), one notes that

$$T_{w\infty} < \frac{K}{M}\left(1 + \frac{A}{B}\right), \tag{15}$$

and therefore $T_{w\infty}$ is strictly less than the general throughput upper bound derived in this section.

Discussion

The internal behavior of the two strategies is shown in Fig. 3 as the expected number of processes in stages 1 and 2 as functions of N. The curves for $M = 3$ and 4 depict the behavior when the SP's limit the throughput of the system. In these cases, with the switch strategy, n_1 remains small and n_2 increases as N is increased. This further explains the behavior of the asymptotic throughput of the switch strategy. As N increases, all the additional processes end up in stage 2, waiting for access to SP's, and n_1 remains too small to provide sufficient work for the available processors. With the wait strategy, the behavior is just the opposite; n_1 increases with N while n_2 remains small. It is interesting that the observed throughputs with the two strategies are similar, even though the internal behavior is markedly different.

Figure 4 plots the throughputs of the two strategies as functions of the number of SP's, with the other parameters fixed. The throughputs of both strategies increase rapidly as K is increased, and the difference between the wait and the idealized switch strategies decreases as K is increased. When comparing the relative performances of the two strategies, it must be remembered that we are considering an idealized switch strategy with no switching overhead. In a situation with a certain amount of switching overhead, the switch strategy would not be better unless the superiority of the idealized switch strategy is more than enough to compensate for the overhead of switching.

It has been observed in multiprocessor systems that as one increases the number of processors while keeping the total processing power constant, it is also necessary to increase the level of multiprogramming in order to keep the overall throughput constant. Figure 5 shows how this effect was present in the behavior of our model. It plots combinations of N and M which yield a throughput of 0.6 for various values of K. Only the switch strategy is considered. The curve for $K = 1000$ is effectively the case where there is no contention for SP's because the probability of two processes concurrently trying to execute the same SP is almost zero. As K becomes smaller, it takes a higher level of multiprogramming (larger N) to achieve the same throughput because additional processes are needed to compensate for those waiting for access to SP's. For $K = 1$, there exist values of M for which 0.6 throughput is unattainable. This occurs because for these values of M and K, the upper bound on the throughput is less than 0.6.

Appendix

The following derivation gives the rates of transition (probabilities of a transition per unit time) between the states of the model shown in Fig. 1(b). These are the rates used to solve for the stationary state probabilities for the model for the wait and switch strategies.

In general, when a transition can occur from stage i to j when the model is in state k, the rate at which this transition occurs is given by

$$\begin{bmatrix}\textit{probability that a process}\\ \textit{leaving } i \textit{ goes to } j\end{bmatrix} \times \begin{bmatrix}\textit{number of processes in service}\\ \textit{in stage } i \textit{ in state } k\end{bmatrix} \div \begin{bmatrix}\textit{mean service time}\\ \textit{of stage } i\end{bmatrix}. \quad \text{(A1)}$$

There are four types of transitions that can occur. The rates at which these transitions occur differ for the two strategies. The switch strategy is considered first.

Transition 1: (n_1, n_2) *to* $(n_1 + 1, n_2)$
This transition occurs when a process completes an I/O operation and moves from stage 3 to stage 1. It can occur whenever $n_1 + n_2 < N$. The rate at which it occurs is constant,

$$r_1 = \frac{1}{C}. \quad \text{(A2)}$$

Transition 2: (n_1, n_2) *to* $(n_1, n_2 - 1)$
This occurs when a process completes stage 2 and moves to stage 3, i.e., chooses with fixed probability $p_{I/O}$ to perform an I/O operation. It can occur whenever $n_2 > 0$. Since there are $\min[M, S(K, n_2)]$ processes executing in stage 2, the rate at which this transition occurs is

$$r_2 = \frac{p_{I/O} \min [M, S(K, n_2)]}{BM}. \quad \text{(A3)}$$

Transition 3: (n_1, n_2) *to* $(n_1 + 1, n_2 - 1)$
This occurs when a process completes stage 2 and moves to stage 1; i.e., it chooses not to do an I/O operation. It can occur whenever $n_2 > 0$ and the rate is

$$r_3 = \frac{(1 - p_{I/O}) \min [M, S(K, n_2)]}{BM}. \quad \text{(A4)}$$

Transition 4: (n_1, n_2) *to* $(n_1 - 1, n_2 + 1)$ This occurs when a process completes stage 1 and moves to stage 2. In order for this transition to occur, there must be processes in stage 1, $n_1 > 0$, and processors available, $M - \min[M, S(K, n_2)] > 0$. The number of executing processes in stage 1 is the minimum of the number of processes and processors, which determines the rate of this transition as

$$r_4 = \frac{\min [n_1, M - \min [M, S(K, n_2)]]}{AM}. \quad \text{(A5)}$$

This defines the set of possible state transitions which can occur with the switch strategy, and the rates at which they occur.

For the wait strategy, the transitions are very similar but have the restriction that in every case $n_2 \leq M$. This follows from the fact that processors and processes wait together for SP's in the second stage. The number of executing processes in stage 2 is $S(K, n_2)$, and since $S(K, n_2) \leq n_2 \leq M$, $\min[M, S(K, n_2)] = S(K, n_2)$. Thus the rates for transitions 1, 2, and 3 with the wait strategy are the same as with the switch strategy. For transition 4 from (n_1, n_2) to $(n_1 - 1, n_2 + 1)$, the rates are different. With the wait strategy the number of processors available is $M - n_2$, and this makes the rate

$$r_4 = \frac{\min [n_1, M - n_2]}{AM}. \quad \text{(A6)}$$

References

1. P. J. Denning, "Third Generation Computer Systems," *ACM Computing Surveys* **3**, 175 (1971).
2. E. G. Coffman and P. J. Denning, *Operating Systems Theory*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1973.
3. W. F. King III, S. E. Smith and I. Wladawsky, "On the Effects of Serial Programs in Multiprocessing Systems," Research Report RC 4136, IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598, November 1972.
4. W. J. Gordon and G. F. Newell, "Closed Queuing Systems With Exponential Servers," *Operations Research* **15**, 254 (1967).
5. R. A. Howard, *Dynamic Programming and Markov Processes*, M.I.T. Press, Cambridge, MA, 1960.
6. D. R. Cox and W. L. Smith, *Queues*, Methuen & Co., Ltd., London, 1961.

Received March 8, 1974

W. F. King's current address is the IBM Research Laboratory, Monterey and Cottle Roads, San Jose, California 95114; S. E. Smith and I. Wladawsky are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.