M. Z. Ghanem

Study of Memory Partitioning for Multiprogramming Systems with Virtual Memory

Abstract: In this paper, we investigate the effect that the shape of the lifetime function has on the optimal partition of the main memory of a computer among N programs, where the criterion of optimality is maximization of CPU utilization. We used a simple queuing model as a base for understanding this interrelationship. The lifetime function is the average of the execution intervals of a program as a function of the amount of memory allocated. When the lifetime function is convex and is proportional to m^{α}, where m is the size of memory, then the optimal partition is obtained by dividing the main memory equally among q of the N programs (q is the optimal degree of multiprogramming). Thus, the best partition is always one of two policies: allocate all memory equally among the q programs or allocate all memory to one program. When the lifetime function has a degenerate S shape (is proportional to m^{α} when $m \leq m_0$ and remains constant beyond m_0), then there exists a memory size m such that no program can have a memory size other than m or m_0; if any program has a memory size greater than m_0, each other program should have a memory size that is equal to or greater than m_0.

Introduction

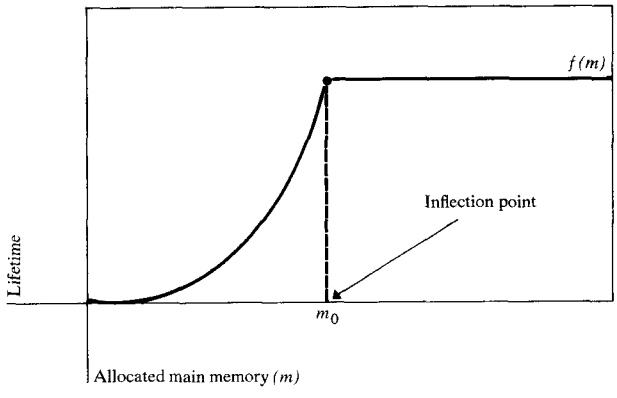
One of the most important purposes of multiprogramming in a virtual memory computer system is to increase utilization of the central processor. The amount of main memory that must be allocated to each program is an important factor in determining this utilization. Belady [1] has shown experimentally that biasing the partition of main memory increases CPU utilization, and he accounted for this result by the convexity of the lifetime function. Denning and Spirn [2] explained Belady's observation analytically using the fact that if the lifetime function f is convex, then $[f(M+\Delta)+f(M-\Delta)]$ is always greater than $2f(M)$, where M is the size of memory. But that is not enough to ensure better CPU utilization due to biasing. Actually, if we follow that argument, it would lead us to allocate all the memory to one and only one program, because if f is convex, then $[f(M+2\Delta)+f(M-2\Delta)]$ is always greater than $[f(M+\Delta)+f(M-\Delta)]$.

This argument assumes that CPU utilization is proportional to the sum of the two lifetime functions. In fact, this neglects the overlapping effect due to multiprogramming (i.e., when some programs are involved in resolving page faults, others can utilize the CPU).

In a previous paper [3], the author included the effect of this overlapping in an analysis through a simple queuing model designed to aid in understanding this aspect of system behavior. In that model it was assumed that only two programs could use the main memory. The author found that when the lifetime function is strictly convex in the region of the available memory and the degree of convexity is less than a certain threshold value, then the optimal solution is a balanced partition (allocate to each program the same amount of memory). If the degree of convexity exceeds the threshold value, then memory should be allocated to only one program (the extreme partition). In other words, the best partition is always one of the two policies; namely the balanced partition or the extreme partition. The previous paper also discussed the case in which the lifetime function has the degenerate S shape shown in Fig. 1.

In this paper, we extend the analysis presented in the previous paper by considering that N programs can use the main memory (where N can be any number less than infinity). We first consider a lifetime function having a

Figure 1 Lifetime function.



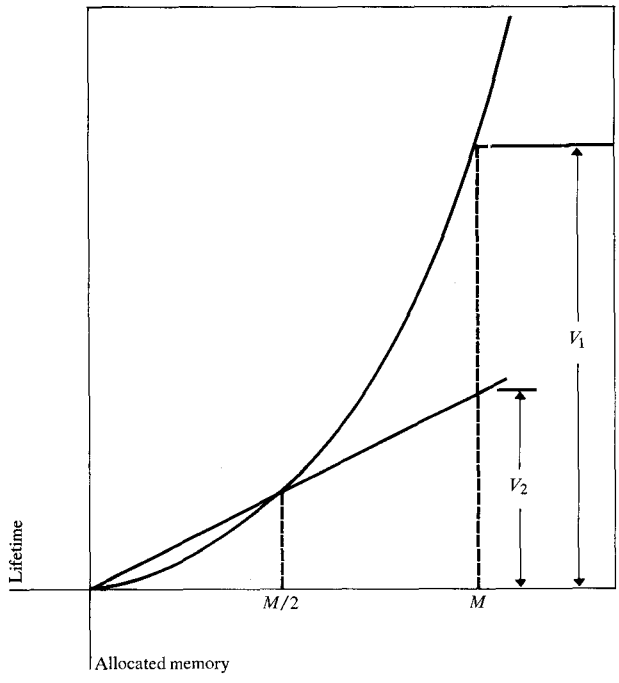


Figure 2 Very convex lifetime function.

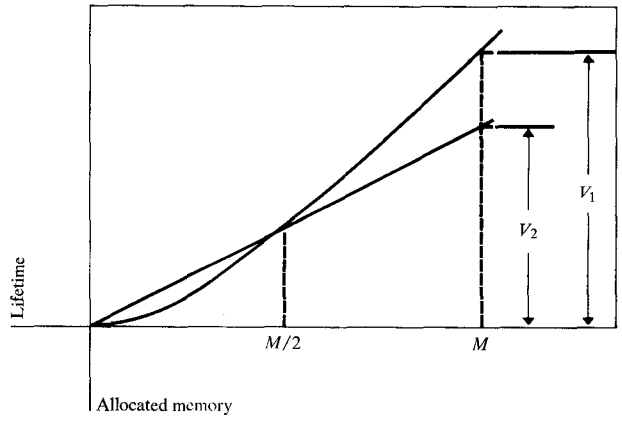


Figure 3 Lifetime function that is not very convex.

convex shape (i.e., can be expressed as Rm^{α}, where R is the average execution interval of a program for an allocated memory size of unity and m is the size of memory). We then show that the optimal partition is to divide the main memory equally among q of the N programs (q is the optimal degree of multiprogramming). We then consider a lifetime function having a degenerate S shape (i.e., it can be expressed as Rm^{α} when m is less than m_0 and as Rm_0^{α} elsewhere). In this case, we found that there exists a memory size m such that no program can have a memory size other then m or m_0, where m_0 is the inflection point. If any program has a memory size that is strictly greater than m_0, each of the other programs should have a memory size that is greater than or equal to m_0.

This result can be explained by assuming that we have two programs I and J and that the available memory size is M. Let V_1 be the average execution interval of program I when we allocate all the available memory, M, to program I; let V_2 be the sum of the average execution interval of program I and the average execution interval of program J when we allocate an amount of memory of $M/2$ to each program. Since the lifetime function is convex, V_1 is always equal to or greater than V_2, as is shown in Figs. 2 and 3. Thus if we neglect the overlapping effect, it is always better to allocate all the memory to one program, but the overlapping effect changes the situation. If we have a very convex lifetime function (α is large), the difference between V_1 and V_2 can dominate the overlapping effect, and it is better to allocate all memory to one program (Fig. 2). If α is not that large (e.g., results in a straight line), the overlapping effect can dominate the difference between V_1 and V_2 and it is better to divide the memory between the two programs equally (Fig. 3).

In the next section, we formulate the problem as a closed queuing network problem and obtain a formula for CPU utilization. We then simplify this formula to be able to analyze the relation between two programs at the optimal partition. Next we analyze the case in which the lifetime function is an exponential function of memory size. Finally, we analyze the case in which the lifetime function has a degenerate S shape, which approximates the actual behavior of programs.

Formulation of the problem

For purposes of investigating this particular problem, we assume that we have N programs residing in main memory. Furthermore, we assume that the time required for bringing a page from the secondary memory to the main memory has the mean value $1/\mu$ and is governed by a probability distribution having a rational Laplace transform. We also assume that the time between page faults for program i has the mean $1/\lambda_i$ (where λ_i is a function of the amount of main memory allocated to program i) and is governed by a proability distribution having a rational Laplace transform. The service discipline of the CPU is assumed to be processor sharing, i.e., the CPU cycles are divided equally among the programs that have their required pages in main memory. The service discipline of the paging I/O device is assumed to be such that the number of servers in the service center is greater than or equal to N.

The assumption that a probability distribution has a rational Laplace transform is a very general assumption. The processor sharing discipline approximates the round-robin scheduling mechanism, which should be used in allocating the CPU because the distribution of the execu-

tion intervals has a long tail [4]. The assumption that the number of I/O devices is greater than N is a simplification to make the problem mathematically tractable.

Based on some recent results by Baskett, Chandy, Muntz, and Palacios [5] on a general queuing network modeling technique, it can be demonstrated (see Appendix) that CPU utilization is given by $(1-\Pi_0)$, where

$$G = (\Pi_0)^{-1} = \sum_{d_1=1,0} \sum_{d_2=1,0} \cdots \sum_{d_N=1,0} \left[\left(\sum_{i=1}^{N} d_i \right)! \prod_{j=1}^{N} x_j^{d_j} \right] \quad (1)$$

where $x_i = \mu/\lambda_i$, $i = 1, 2, \cdots, N$.

Analysis of the objective function

The expression for the CPU utilization given in the previous section is rather complex. To be able to analyze the relation between any two programs at the optimal partition, we derive the objective function as a function of two variables only (x_i and x_j). The form that is derived is less complicated than the form of Eq. (1).

Theorem 1 For given i and j, G can be written as

$$G(x_1, x_2, \cdots, x_N) = B_0(i,j) + B_1(i,j)[x_i + x_j] + B_{12}(i,j)x_i x_j,$$

where $B_0(i,j)$, $B_1(i,j)$, and $B_{12}(i,j)$ are functions of $x_1, x_2, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{j-1}, x_{j+1}, \cdots, x_N$; i.e., they do not depend on x_i or x_j.

Proof By rearranging the order of summation in Eq. (1), G can be rewritten as

$$\begin{aligned} G = & \sum_{d_1=0,1} \sum_{d_2=0,1} \cdots \sum_{d_N=0,1} \left[\sum_{d_i=0,1} \sum_{d_j=0,1} \left(\sum_{k=1}^{N} d_k \right)! \prod_{k=1}^{N} x_k^{d_k} \right] \\ = & \sum_{d_1=0,1} \sum_{d_2=0,1} \cdots \sum_{d_N=0,1} \left(\left\{ \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right) + 2 \right]! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right\} x_i x_j \right. \\ & + \left\{ \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right) + 1 \right]! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right\} x_i \\ & + \left\{ \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right) + 1 \right]! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right\} x_j \\ & \left. + \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right)! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right] \right) \\ = & B_0 + B_1(x_i + x_j) + B_{12} x_i x_j, \end{aligned}$$

where

$$B_0 = \sum_{d_1=0,1} \sum_{d_2=0,1} \cdots \sum_{d_N=0,1} \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right)! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right]$$

$$B_1 = \sum_{d_1=0,1} \sum_{d_2=0,1} \cdots \sum_{d_N=0,1} \left\{ \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right) + 1 \right]! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right\}$$

$$B_{12} = \sum_{d_1=0,1} \sum_{d_2=0,1} \cdots \sum_{d_N=0,1} \left\{ \left[\left(\sum_{\substack{k=1 \\ k\neq i,j}}^{N} d_k \right) + 2 \right]! \prod_{\substack{k=1 \\ k\neq i,j}}^{N} x_k^{d_k} \right\}.$$

And it is clear that the B_0, B_1, and B_{12} do not depend on x_i or x_j.

Convex lifetime function

In this section we analyze the case in which the lifetime function is an exponential function of memory size (i.e., is of the form Rm^{α}. We analyze the case in which $\alpha > 1$ for the following reasons:

1. When $0 \leq \alpha \leq 1$ (i.e., the lifetime function is concave), it is easy to prove that the optimal policy is to divide main memory equally among the N programs.
2. The case in which $\alpha < 0$ (i.e., the lifetime function is monotonically decreasing) is not important because it is the nature of the lifetime function to be monotonically nondecreasing.

In this section we prove that when $\alpha > 1$, any optimal partition has to be in the form $(M/q, M/q, \cdots, M/q, 0, 0, 0)$, where q is the optimal degree of multiprogramming. In particular when $N = 2$, we prove that there exists an α_c such that when $\alpha > \alpha_c$, the optimal partition is to allocate all of memory to one of these programs and if $\alpha \leq \alpha_c$, the optimal partition is to divide the memory equally between two programs.

We first show the maximization of the function

$$B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2 \quad (2)$$

subject to the constraints

$$x_1 = \mu R(m_1)^{\alpha}, \quad (3)$$

$$x_2 = \mu R(m_2)^{\alpha}, \text{ and} \quad (4)$$

$$m_1 + m_2 = M_{12}, \quad (5)$$

is achieved at either $(m_1 = m_2 = M_{12}/2)$ or $(m_1, m_2 = 0, M_{12})$.

Let $\ell = (2m_1/M_{12}) - 1$ and (6)

$$A = \mu R(M_{12}/2)^{\alpha}. \tag{7}$$

Then $x_1 = A(1+\ell)^{\alpha}$, (8)

$$x_2 = A(1-\ell)^{\alpha}, \tag{9}$$

and the maximization problem becomes:

$$\text{Maximize } f(\ell) \equiv B_1A(1+\ell)^{\alpha} + B_1A(1-\ell)^{\alpha} + B_{12}A^2(1-\ell^2)^{\alpha} \text{ in the range } -1 \le \ell \le 1.$$

The function $f(\ell)$ is an even function of ℓ, i.e., $f(\ell) = f(-\ell)$. Thus we will concentrate on examining $f(\ell)$ for positive values of ℓ, i.e., for $0 \le \ell \le 1$.

Because

$$\frac{\partial f}{\partial \ell} = AB_1\alpha[(1+\ell)^{\alpha-1} - (1-\ell)^{\alpha-1}] - 2A^2\alpha B_{12}(1-\ell^2)^{\alpha-1}\ell, \tag{10}$$

we have a stationary point at $\ell = 0$.

Because

$$\frac{\partial^2 f}{\partial \ell^2} = AB_1\alpha(\alpha-1)[(1+\ell)^{\alpha-2} + (1-\ell)^{\alpha-2}] - 2A^2\alpha B_{12}[(1-\ell^2)^{\alpha-1} - (\alpha-1)(2\ell^2)(1-\ell^2)^{\alpha-2}], \tag{11}$$

then at $\ell = 0$

$$\frac{\partial^2 f}{\partial \ell^2}\Big|_{\ell=0} = 2AB_1\alpha(\alpha-1) - 2A^2\alpha B_{12}. \tag{12}$$

From Eq. (12), we can conclude that if $\alpha > 1 + (AB_{12}/B_1)$, then f will have a local minimum at the point $\ell = 0$.

Proposition 1 If $\alpha > 1 + (AB_{12}/B_1)$, then $\partial f/\partial \ell$ is greater than zero for all values of $\ell \in (0, 1)$.

Proof Because

$$\frac{\partial f}{\partial \ell} = A\alpha[B_1(1+\ell)^{\alpha-1} - B_1(1-\ell)^{\alpha-1} - 2AB_{12}\ell(1-\ell^2)^{\alpha-1}]$$

$$= A\alpha(1-\ell^2)^{\alpha-1} \times \left[\frac{B_1}{(1-\ell)^{\alpha-1}} - \frac{B_1}{(1+\ell)^{\alpha-1}} - 2AB_{12}\ell\right] \tag{13}$$

and $A\alpha(1-\ell^2)^{\alpha-1} > 0$ for all values of $\ell \in (0, 1)$, all that we have to prove is that

$$c(\ell) = B_1(1-\ell)^{1-\alpha} - B_1(1+\ell)^{1-\alpha} - 2AB_{12}\ell > 0 \tag{14}$$

for all values of $\ell \in (0, 1)$. But

$$\frac{\partial c(\ell)}{\partial \ell} = -(1-\alpha)B_1(1-\ell)^{-\alpha} - (1-\alpha)B_1(1+\ell)^{-\alpha} - 2AB_{12}, \tag{15}$$

which implies that

$$\frac{\partial c(\ell)}{\partial \ell}\Big|_{\ell=0} = -2B_1(1-\alpha) - 2AB_{12}. \tag{16}$$

But because $\alpha - 1 > (AB_{12}/B_1)$, it implies that

$$\frac{\partial c}{\partial \ell}\Big|_{\ell=0} > 0. \tag{17}$$

On the other hand $\partial^2 c/\partial \ell^2$ can be written as

$$\frac{\partial^2 c}{\partial \ell^2} = -\alpha(1-\alpha)B_1(1-\ell)^{-\alpha-1} + \alpha(1-\alpha)B_1(1+\ell)^{-\alpha-1}$$

$$= \alpha(\alpha-1)B_1[(1-\ell)^{-\alpha-1} - (1+\ell)^{-\alpha-1}]$$

$$\ge 0. \tag{18}$$

From Eqs. (17) and (18) and from the fact that $c = 0$ at $\ell = 0$, we can conclude that $(\partial f/\partial \ell) > 0$ for all values of $\ell \in (0, 1)$. Q.E.D.

Theorem 2 If $\alpha > 1 + (AB_{12}/B_1)$, then f achieves its minimum at $\ell = 0$. Moreover, it achieves its maximum at $\ell = 1$.

Proof We proved that if $\alpha > 1 + (AB_{12}/B_1)$ then f has a local minimum at $\ell = 0$; from Proposition 1, the slope of f is greater than zero for all values of ℓ. Thus f, in this case, is a monotonically increasing function of ℓ, which implies that the local minimum at $\ell = 0$ is actually a global minimum. Because f is a monotonically increasing function of ℓ and we consider the range $0 \le \ell \le 1$, it is clear that the maximum of f is at $\ell = 1$.

Theorem 3 If $1 \le \alpha \le 1 + (AB_{12}/B_1)$, then the maximum of f is achieved at either $\ell = 0$ or $\ell = 1$.

Proof If $\alpha \le 1 + (AB_{12}/B_1)$, we can conclude from Eqs. (16) and (18) that

$$\frac{\partial c}{\partial \ell} \le 0 \text{ at } \ell = 0 \text{ and} \tag{19}$$

$$\frac{\partial^2 c}{\partial \ell^2} \ge 0 \text{ for every } \ell. \tag{20}$$

Equations (19) and (20) and the fact that $c(0) = 0$ imply either that c is always negative or that if c is not always negative it may become zero once at $\ell = \ell^*$ (for some ℓ^*) and then stay nonnegative in the interval $(\ell^*, 1]$. The first case means that $f(\ell)$ possesses its global maximum at $\ell = 0$. The second case means that the global maximum should be either at $\ell = 0$ or at $\ell = 1$, depending on whether $f(0) > f(1)$ or $f(1) > f(0)$, respectively.

Corollary 1 For any $\ell_0 \in (0, 1)$, if $f(\ell_0) > f(0)$, then then $f(\ell_0) \leq f(\ell)$ for all $\ell \in (\ell_0, 1]$.

Proof If $f(\ell_0) > f(0)$, then from Theorem 3 ℓ_0 lies in the interval $(\ell^*, 1]$. Because c is nonnegative in the interval $(\ell^*, 1]$, then $\partial f / \partial \ell$ is nonnegative in that interval. This implies that $f(\ell_0) \leq f(\ell)$ for all $\ell \in (\ell_0, 1]$. Q.E.D.

From Theorems 2 and 3 we can conclude that $f(\ell)$ has its maximum value either at $\ell = 0$ or at $\ell = 1$. Now, we can conclude the following theorem.

Theorem 4 The maximization of the function $B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2$, subject to the constraints $x_1 = \mu R(m_1)^\alpha$, $x_2 = \mu R(m_2)^\alpha$ $(\alpha \geq 0)$, and $m_1 + m_2 = M_{12}$ is achieved at either $(m_1 = m_2 = M_{12}/2)$ or $(m_1, m_2 = 0, M_{12})$.

Now we are ready to prove the main theorem.

Theorem 5 Let $P = (m_1, m_2, \cdots, m_N)$ be a memory partition. Assume that all the programs have the same lifetime function, which is of the form $Rm^\alpha (\alpha \geq 0)$. If there exist i and j such that $m_i \neq m_j$, $m_i > 0$ and $m_j > 0$, then P is not an optimal partition.

Proof Without loss of generality let $m_1 \neq m_2$, $m_1 > 0$ and $m_2 > 0$. Because

$$G(x_1, x_2, \cdots, x_N) = B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2,$$

using Theorem 4 we can redistribute $M_{12} = (m_1 + m_2)$ between programs 1 and 2 and find another partition $(m_1^* = m_2^* = (m_1 + m_2)/2$ or $m_1^* = 0$ and $m_2^* = M_{12})$ that achieves a larger value of G, i.e., a higher CPU utilization. Thus P is not an optimal partition.

Theorem 5 can be restated as follows:

Theorem 5' Let $P = (m_1, m_2, \cdots, m_N)$ be a memory partition. Assume all the programs have the same lifetime function, which is convex and of the form Rm^α. If P is an optimal partition, then P must be in the following form:

$P = (m_1, m_2, \cdots, m_N)$ with

$m_i = M/j$ for $i \leq j$

$= 0$ for $i > j$.

Corollary 2 If $N = 2$, then there is an α_c such that if $\alpha > \alpha_c$ the optimal partition of memory is extreme (i.e., all memory is allocated to one program). And if $\alpha \leq \alpha_c$, the best memory partition is unbiased (i.e., main memory is divided equally between the two programs).

Proof The proof is obvious from the above theorems, and a different proof is given in [6].

Degenerate S shape lifetime function

In practice, the lifetime function is not convex over the entire region $m \geq 0$. Actually, if the amount of memory allocated to a program exceeds a certain amount m_0, the lifetime function increases little by additional memory allocation [7]. Thus the lifetime function can be approximated by a function $y_1(m)$ of the form:

$$y_1(m) = \begin{cases} Rm^\alpha & 0 \leq m \leq m_0 \\ Rm_0^\alpha & m \geq m_0 \end{cases} \tag{21}$$

In this section, we develop results similar to those that were developed in the previous section. We analyze the case where $\alpha \geq 1$ for the same reasons that were mentioned previously. The main result is that there exists a memory size m such that no programs can have a memory size other than m or m_0.

Now we can conclude the following theorem:

Theorem 6 Maximization of the function

$$B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2,$$

subject to the constraints

$$x_1 = \mu y_1(m_1),$$

$$x_2 = \mu y_1(m_2), \text{ and}$$

$$m_1 + m_2 = M_{12} < 2m_0,$$

is achieved at either $m_1 = m_2 = M_{12}/2$ or $(m_1, m_2) = (m_0, M_{12} - m_0)$.

Proof Let $y_1(m)$ be the lifetime function, and assume that $H(y_1, m_1, m_2)$ denotes the value of the function

$$B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2$$

when x_1 is equal to $\mu y_1(m_1)$ and x_2 is equal to $\mu y_1(m_2)$. If $m_1 < m_0 < m_2$, then for the function $x = \mu y_1(m)$ defined above,

$$H(y_1, m_1, m_2) = B_0 + B_1\mu R(m_1)^\alpha + B_1\mu R(m_0)^\alpha + B_{12}\mu^2R^2(m_1m_0)\alpha.$$

But $H(y_1, M_{12} - m_0, m_0) = B_0 + B_1\mu R(M_{12} - m_0)^\alpha + B_1\mu R(m_0)^\alpha + B_{12}\mu^2R^2[m_0(M_{12} - m_0)]$. Because $M_{12} - m_0 > m_1$, we can conclude that $H(y_1, M_{12} - m_0, m_0) \geq H(y_1, m_1, m_2)$ for all m_1 and m_2 such that

$$m_1 < m_0 < m_2. \tag{22}$$

On the other hand, if $m_1 < m_0$ and $m_2 < m_0$, the following conclusion can be made:

$$\text{If } H(y_1, m_1, m_2) > H(y_1, M_{12}/2, M_{12}/2,) \tag{23}$$

then by Corollary 1 to Theorem 3, we obtain

$$H(y_1, m_0, M_{12} - m_0) \geq H(y_1, m_1, m_2). \tag{24}$$

From Eqs. (23) and (24), we can conclude that

$$H(y_1, m_1, m_2) \leq \max [H(y_1, m_0, M_{12} - m_0) \quad H(y_1, M_{12}/2, M_{12}/2)] \tag{25}$$

for any m_1 and m_2 such that $m_1 < m_0$ and $m_2 < m_0$. From (22) and (25), it is clear that the maximum value of $H(y_1, m_1, m_2)$ should be at either the point $(M_{12}/2, M_{12}/2)$ or $(m_0, M_{12} - m_0)$. Q.E.D.

Theorem 7 Maximization of the function

$$B_0 + B_1(x_1 + x_2) + B_{12}x_1x_2,$$

subject to the constraints

$$x_1 = \mu y_1(m_1),$$

$$x_2 = \mu y_1(m_2), \text{ and}$$

$$m_1 + m_2 = M_{12} > 2m_0,$$

is achieved at all the points of the set O, where

$$O = \{(m_1^*, m_2^*): m_1^* = (M_{12}/2) - \gamma, \quad m_2^* = (M_{12}/2) + \gamma, \gamma \leq (M_{12}/2) - m_0\}.$$

Proof At any point (m_1^*, m_2^*) that is in the set O, we have

$$x_1 = \mu R(m_0)^\alpha, \text{ and}$$

$$x_2 = \mu R(m_0)^\alpha.$$

Then $H(y_1, m_1^*, m_2^*)$ can be written as

$$H(y_1, m_1^*, m_2^*) = B_0 + 2B_1\mu R(m_0)^\alpha + B_{12}\mu^2R^2(m_0)^{2\alpha}. \quad (26)$$

Let (m_1, m_2) be a point that is not in the set O. Then either $m_1 < m_0$ or $m_2 < m_0$. Without loss of generality, assume $m_1 < m_0$; then H can be written as

$$H(y_1, m_1, m_2) = B_0 + B_1\mu R(m_1)^\alpha + B_1\mu R(m_0)^\alpha + B_{12}\mu^2R^2(m_1)^\alpha(m_0)^\alpha. \quad (27)$$

From (26) and (27) we conclude that $H(y_1, m_1^*, m_2^*) \geq H(y_1, m_1, m_2)$, where (m_1^*, m_2^*) is a point in the set O and (m_1, m_2) is a point that is not in the set O. Q.E.D.

But it can be shown that, at an optimal partition, if any program has an amount of memory that is strictly greater than m_0, then it can be proved that $m_i \geq m_0$ for all i. This case can never happen except when the total size of main memory is greater than Nm_0 (and the problem becomes trivial). But if the total size of main memory is less than or equal to Nm_0, we can prove the following theorem.

Theorem 8 Let $P = (m_1, m_2, \cdots, m_N)$ be a memory partition. Assume that all the programs have the same degenerate S-shape lifetime function, with inflection point m_0, of the form given by Eq. (21). Furthermore, assume that the total size of main memory is less than, or equal to, Nm_0. If there exist i and j such that $m_i \neq m_j$ ($m_i \neq 0$ or m_0) and ($m_j \neq 0$ or m_0), then P is not an optimal partition.

Proof: The proof is similar to that for Theorem 3.

Conclusion

The main conclusion of this paper is that the best memory allocation tends to be unbiased among q programs (where q is the optimal degree of multiprogramming). This step decreases the difficulty of the problem of finding the optimal degree of multiprogramming. We have presented here the case in which every program has the same lifetime function. We conjecture that in the case of different programs the meaning of unbiased memory allocation must be changed to reflect the characteristic of each program.

Appendix

Assume that we have two stations, station 1 representing the CPU and station 2 the paging I/O device. Consider the state of station 1 to be represented by

$$y_1 = (n_{11}, n_{12}, \cdots, n_{1N}),$$

where

$n_{1r} = 1$ if program r is being served at the CPU,

$= 0$ if program r is not being served at the CPU.

Also consider the state of station 2 to be represented by

$$y_2 = (n_{21}, n_{22}, \cdots, n_{2N}),$$

where

$n_{2r} = 1$ if program r is using the I/O device,

$= 0$ if program r is not using the I/O device.

If P_{ij} is the probability that a program moves to station j after the completion of its service at station i, then $p_{11} = 0$, $p_{12} = 1$, $p_{21} = 1$, and $p_{22} = 0$. So from [4], the equilibrium state probability can be written as

$$P(y_1, y_2) = Cg_1(y_1)\,g_2(y_2), \quad (A1)$$

where

$$g_1(y_1) = n_1! \prod_{r=1}^{N} \frac{1}{n_{1r}!} \left(\frac{e_{1r}}{\lambda_r}\right)^{n_{1r}} \text{ and} \quad (A2)$$

$$g_2(y_2) = \prod_{r=1}^{N} \frac{1}{n_{2r}!} \left(\frac{e_{2r}}{\mu_r}\right)^{n_{2r}}. \quad (A3)$$

Here n_1 is the number of programs served at the CPU, i.e., $n_1 = \sum_{r=1}^{N} n_{1r}$. Then e_{1r} and e_{2r} must satisfy the following relations:

$$e_{1r} \cdot 0 + e_{2r} \cdot 1 = e_{1r} \quad (A4)$$

$$e_{1r} \cdot 1 + e_{2r} \cdot 0 = e_{2r}. \quad (A5)$$

From Eqs. (A.4) and (A.5) we conclude that

$$e_{1r} = e_{2r}$$

because $n_{ir} = 0$ or 1, then $n_{ir}! = 1$. Therefore by substituting Eqs. (A2) and (A3) into (A1), we obtain

$$P(y_1, y_2) = cn_1! \prod_{r=1}^{N} \left(\frac{e_{1r}}{\lambda_r}\right)^{n_{1r}} \left(\frac{e_{2r}}{\mu_r}\right)^{n_{2r}}. \tag{A6}$$

But by knowing that $e_{1r} = e_{2r}$ and $n_{1r} + n_{2r} = 1$, Eq. (A6) can be written as

$$\begin{aligned} P(y_1, y_2) &= cn_1! \prod_{r=1}^{N} (e_{1r}) \left[\prod_{r=1}^{N} \left(\frac{1}{\lambda_r}\right)^{n_{1r}} \left(\frac{1}{\mu_r}\right)^{n_{2r}}\right] \\ &= \frac{c \prod_{r=1}^{N} e_{1r}}{\prod_{r=1}^{N} \frac{1}{\mu_r}} n_1! \left[\prod_{r=1}^{N} \left(\frac{\mu_r}{\lambda_r}\right)^{n_{1r}}\right] \\ &= c' n_1! \left(\prod_{r=1}^{N} x_r^{n_{1r}}\right), \end{aligned} \tag{A7}$$

where $x_r = \mu_r/\lambda_r$. Thus $P(y_1, y_2)$ actually depends only on the state y_1, which was not unexpected because $y_1 + y_2 = (1, 1, \cdots, 1)$. Thus $P(y_1, y_2)$ can be written as

$$P(n_{11}, n_{12}, \cdots, n_{1N}) = c' n_1! \left(\prod_{r=1}^{N} x_r^{n_{1r}}\right).$$

But because

$$\sum_{\sum_{r=1}^{N} n_{ir} \le N} p(n_{11}, n_{12}, \cdots, n_{1N}) = 1;$$

then

$$c' = \left\{ \sum_{n_{11}=0,1} \sum_{n_{12}=0,1} \cdots \sum_{n_{1N}=0,1} \left[\sum_{r=1}^{N} n_{1r}\right]! \prod_{r=1}^{N} x_r^{n_{1r}} \right\}^{-1}.$$

Thus the probability that the CPU will be idle, $P(0, 0, \cdots, 0)$, can be written as $\Pi_0 = c'$, which implies that

$$(\Pi_0)^{-1} = \sum_{n_{11}=0,1} \sum_{n_{12}=0,1} \cdots \sum_{n_{1N}=0,1} \left[\sum_{r=1}^{N} n_{1r}\right]! \prod_{r=1}^{N} x_r^{n_{1r}}.$$

Acknowledgments

The author thanks L. A. Belady, P. J. Denning, W. D. Frazer, and J. R. Spirn for their valuable comments.

Cited and general references

1. L. A. Belady and C. J. Kuehner, "Dynamic Space Sharing in Computer Systems," *Comm. ACM* **12,** 5 (May 1969).
2. P. J. Denning and J. R. Spirn, "Dynamic Storage Partitioning," *4th ACM Operating System Symposium*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, October 1973.
3. M. Z. Ghanem, "On the Optimal Memory Allocation Problem," *Research Report RC 4443*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1974.
4. M. Z. Ghanem, "Experimental Study of the Behavior of Programs," *Research Report RC5427*, IBM Thomas J. Watson Research Center, Yorktown Heights, New York, 1975.
5. F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues With Different Classes of Customers," *J. ACM* **22** (April 1975).
6. P. H. Oden and G. S. Shedler, "A Model of Memory Contention in a Paging Machine," *Comm. ACM* **15,** 8 (August 1972).
7. P. J. Denning and G. S. Graham, "Multiprogramming and Memory Management," *Proc. IEEE* (Interactive Computer Systems) **63,** 924 (1975).
8. R. A. Howard, *Dynamic Probabilistic Systems*, Vol. 1, John Wiley and Sons, Inc., New York 1969.
9. A. E. Ferdinand, "An Analysis of the Machine Interference Model," *IBM Syst. J.* **10,** 2 (1971).
10. J. R. Spirn, "A Model for Dynamic Allocation in a Paging Machine," *8th Annual Princeton Conference on Information Sciences and Systems*, Princeton University, March 1974.
11. D. D. Chamberlin, S. H. Fuller, L. Y. Liu, "An Analysis of Page Allocation Strategies for Multiprogramming Systems with Virtual Memory," *IBM J. Res. Develop.* **17,** 404 (1973).

Received October 14, 1974; revised April 23, 1975

The author is located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.