Science Bulletin of NCTU, Vol. V, No. 2

A DYNAMIC STORAGE ALLOCATION ALGORITHM FOR MULTIPROCESSOR COMPUTER SYSTEMS

WU-HAUNG CHENG and C.C. YANG

College of Engineering. National Chiao Tung University

(Received 3, July 1971)

Abstract—*In this paper a dynamic storage allocation algorithm is developed. The algorithm is very suitable for multiprocessor computer systems under multiprogramming and/or time-sharing environment. It may be considered as a dynamic Bayesian type. Since all the statistical information used in the algorithm is dynamically extracted, prerunning of programs is not required at all. The technique of biasing the working parameters and the concept of dynamic activity are both applied for developing the algorithm. It is shown that the system performance is improved on the average. Although some additional cost is small. In addition the algorithm is simple in implementation.*

I. INTRODUCTION

In order to efficiently utilize the resources of a computer system, various aspects of the resource allocation must be carefully considered when we design an operating system. The principal resources of a computer system are the central processing units (CPU's) and the the main memory. In multiprogramming and/or time-sharing systems many jobs reside in the main memory concurrently, and each has its own demands[1,2] of the processor time and the memory space. Since the processor management is to allocate the processor time and the capacity of the main memory is limited, competitions exist among jobs. Thus we have scheduling algorithms[3,4] for allocating the processor time and storage allocation algorithms[1,5-11] for allocating the main memory space.

Before the appearance of the working set model[1] in the literature, processor management and memory allocation had developed and progressed independently Denning[1] suggested that a unified approach is needed. Under this notion, resource allocation becomes the problem of balancing memory and processor demands against equipment[1,2]. It uses the working set of a process* to determine the memory demand. Since there is an intimate relation between a process and its working set, memory allocation and process scheduling must be closely related activities. In this manner, the page traffic would be minimized. Belady[8] mentioned that processor efficiency can be considered as a performance indicator. He presented biased replacement algorithms[7,8] indicator. He presented biased replacement algorithms which

* A process is assumed to be in the form of the processor demand of a program

favor one of the contending tasks for a period of (P) replacements such that the selected task has all of its pages exempted from consideration by the replacement algorithm within that period. Shemer and Gupta[10] presented Bayesian storage allocation algorithms, which use statistical information, such as page reference distribution function[6,10] and some usage bits, for fetching or replacing a page. Tung[11] developed the concept of apparent continuity of processing. Uunder this concept, the working ensemble of a job is saved at the end of the last job phase and is retrieved as a whole at the beginning of the next job phase. A job phase is defined such that it can be run without interruption. All the technigues, as briefly mentioned above, have the same goal: to solve the problem of memory allocation such that processor efficiency is maximized, or equivalently the number of page faults is minimized.

In this paper a two-level storage system is used. One is the main memory with limited capacity, which is partitioned into page frames and can be directly accessed by processors. The other is the auxiliary memory with unlimited capacity, which can not be directly accessed by processors.

The concept of virtual memory[12-17] has been widely accepted. And the techniques of segmentation and paging[13-17] have been developed. In this paper we don't discuss them in any detail. In addition we assume that a page is an information transferring unit.

Now, we consider the storage allocation problem. It may be classified into two parts: (1) page transferring and (2) page placing. The first part is a twofold: one is "fetch" which is the problem of deciding when a given page is to be loaded into the main memory; and the other is "replacement" which is the problem of deciding which page is to be removed from the main memory in order to make room for new allocation. As for the second part, it is the problem of deciding at which area of the main memory a page is to be placed[15,17].

Since the technique of paging is used, page placing is not an unsolved problem. When a page fault occurs, if the there is no non-occupied page frame, then the page chosen by the replacement algorithm is replaced from the main memory; and the page is placed at the area from where the old page is just removed.

To solve the "fetch" problem, we have demand-paging[16,18] policy for fetch algorithm. Since there is no advance source of allocation information and we won't load a page which is not used in the subsequent processing period, a page is fetched on the basis of demand.

Thus the main part of storage allocation we want to discuss is "replacement". This paper intends to develop a better replacement algorithm for a multiprocessor computer system under multiprogramming and/or time-sharing environment.

The replacement alsorithm presented in this paper may be considered as a Bayesian type, i. e., all the statistical information used in the algorithm is dynamically extracted. In Shemer and Gupta's Bayesian storage allocation algorithms, the usage frequency of a segment is the total number of references being made to that segment and may be obtained by previous runs. But we shall use "dynamic activity"

for a page, residing in the main memory, to indicate the usage activity of that page within a time interval in τ seconds prior to the time considered. And then we shall use "dynamic frequency" for a segment to be the sum of the dynamic activities of all "in-core" pages of that segment.

The τ mentioned above is defined as the "working parameter" of a process. Here we assume that a process is in the form of the processor demand of a segment. So, the pages, of a segment, being referenced within the last τ seconds constitute the "working part" of that segment. And the the working parts of the segments processed by a given processor constitute the "working collective" of that processor.

We shall use the technique of biasing the working parameters to develop a replacement algorithm for multiprocessor computer systems. Due to the general adherence of programs to the convex relationship[7,8], we can hope that the efficiency of processors is improved (the biasing technique used here is different from that used in[8], which is briefly stated in a previous paragraph).

In the following, we firstly introduce some preliminary considerations about the working part of a segment, the working collective of a processor, the mean processing time of a process, and the convex relationship. The technique of biasing the working parameters is then developed. Finally, a storage allocation algorithm is presented. It will be seen that the algorithm is simple in implementation.

II. PRELIMINARY CONSIDERATIONS

We assume that readers are already familiar with the concepts of segmentation and paging[13,17], and of program and addressing structure[14]. Thus we don't repeat those topics in any detail. What we intend to introduce are the following:

(1) *Working Part of a Segment:*

Let $S_{j(i)}$ denote the segment j of the program i*. The working part $W_{j(i)}(t, \tau)$ of $S_{j(i)}$ at time t is defined to be the collection of pages, of that segment, referenced by the processor during the processing time interval $(t-\tau, t)$.

Let $\omega_{j(i)}(t, \tau)$ be the size of the working part $W_{j(i)}(t, \tau)$ of a given $S_{j(i)}$. The

$$\omega_{j(i)}(t, \tau) = \text{no. of pages in } W_{j(i)}(t, \tau) \qquad (1)$$

It is quite clear that $\omega_{j(i)}(t, \tau)$ is a monotonically increasing function of τ with $\omega_{j(i)}(t, 0)=0$ and with p as the maximum value which is in fact the total number of pages of the given $S_{j(i)}$. And $\omega_{j(i)}(t, \tau)$ is concave downward[1] as shown in Fig. 1 (the smoothed curve only suggests the general character of $\omega_{j(i)}(t, \tau)$, and here the statistical regulariy is also assumed as[1] does).

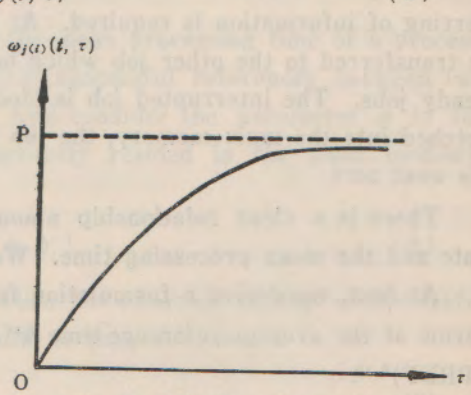

Fig. 1. Behavior of $\omega_{j(i)}(t, \tau)$.

* The program may be a user's program or a system program.

(2) *Working Collective of a Processor:*

The working collective $W_h(t, \tau)$ of the processor h at time t is defined to be the union of the working parts of the segments processed by that processor. Thus

$$W_h(t, \tau) = \bigcup_{\substack{\text{all working parts} \\ \text{processed by the} \\ \text{given processor } h}} W_{f(i)}(t, \tau) \qquad (2)$$

Let $\omega_h(t, \tau)$ be the size of the working collective $W_h(t, \tau)$ of a given processor. Since the elements of a segment are different from those of any other segment, we have

$$\omega_h(t, \tau) = \sum_{\substack{\text{all working parts} \\ \text{processed by the} \\ \text{given processor } h}} \omega_{f(i)}(t, \tau) \qquad (3)$$

From the memory usage viewpoint, $\omega_h(t, \tau)$ of a processor may be considered as the space allotment for that processor at time t provided that there is no overhap of information among the working collectives. Of course, this space allotment is dynamically changed. If the statistical regularity is assumed, this quantity may be considered as constant for a given working parameter τ.

(3) *Mean Processing Time of a Process:*

If the capacity of the main memory is large enough to accomodate the information of all active programs, then there is nothing to be discussed concerning the mean processing time, since any program will be executed from the beginning till its completion without a page fault. On the contrary if the capacity of the main memory is limited and many jobs reside in the memory concurrently, then each can only have a portion of its information which occupies some page frames. It is almost impossible that a job can be run from the beginning to the end without interruption. So page faults do occur.

Whenever a page fault occurs, the corresponding job is interrupted and transferring of information is required. At this moment the control of the processors is transferred to the other job which has the highest prierity in the queue of all ready jobs. The interrupted job is blocked. Wenever the required information is fetched into the main memory, the job becomes ready and waits in the queue for its next turn.

There is a clear relationship among the process efficiency, the page-faulting rate and the mean processing time. We shall discuss this relationship later.

At first, we derive a formulation for the mean processing time of a process in terms of the average reference time Δ[17] and the Page Reference Distribution Function (PRDF)[6,10].

In order to continue the presentation, we shall briefly review the PRDF. As stated in[10]: the PRDF, $F_{f(i)}$, is a continuous distribution function for $S_{f(i)}$ such that

$$F_{j(i)}(x, p) = \Pr.\left\{\begin{array}{l}\text{A reference within } S_{j(i)} \text{ is directed}\\ \text{to one of the last } x \text{ distinct pages}\\ \text{previously referenced in } S_{j(i)} \text{ given}\\ \text{that a reference is directed to } S_{j(j)}\end{array}\right\} \quad (4)$$

According to the respective manner by which memory words are referenced, the four distribution functions are analytically formulated as shown by the following equations. For detailed information, readers may refer to[6,10].

$$F_{j(i)}(x, p) = \begin{cases} x/p, & \text{if } 0 \le x < p \\ 1, & \text{if } x \ge p \end{cases} \quad (5)$$

Eq. (5) indicates the pure random case.

$$F_{j(i)}(x, p) = \begin{cases} 0, & \text{if } x = 0 \\ 1 - 1/n, & \text{if } 0 < x < p \\ 1, & \text{if } x \ge p \end{cases} \quad (6)$$

where n is the number of words contained in each page. Eq. (6) indicates pure sequential case.

$$F_{j(i)}(x, p) = \begin{cases} 0, & \text{if } x = 0 \\ 1 - (1 - x/p)/v, & \text{if } 0 < x < p \\ 1, & \text{if } x \ge p \end{cases} \quad (7)$$

where v is the number of words contained in each entry. Eq. (7) indicates the random sequential case. Note that for small v, this case tends to the pure random case; whereas for large v, it tends to the pure sequential case.

$$F_{j(i)}(x, p) = \begin{cases} 0, & \text{if } x = 0 \\ 1 - \dfrac{1}{v}\left[1 - \left\{\dfrac{p^2(u-1) + x(x+1)}{p(pu+1)}\right\}\right], & \text{if } 0 < x < p \\ 1, & \text{if } x \ge p \end{cases} \quad (8)$$

where u is the number of entries searched on each page. Eq. (8) indicates the sampled sequential search case. Note that for large u and v this function is similar to that of the pure sequential case.

Now, we show the PRDF contributes to the mean processing time of a process.

Define $Q_{n_{j(i)}}$ to be the probability of $(n-1)$ successful references between two consecutive page faults of the process $j(i)$. And consider the parameter x in the PRDF to be the number of pages of $S_{j(i)}$ currently resided in the main memory. It is quite clear that

$$Q_{n_{j(i)}} = [F_{j(i)}(x, p)]^{n-1} \cdot [1 - F_{j(i)}(x, p)] \quad (9)$$

Eq. (9) indicates a geometric distribution. Then the expected number of successful references between two consecutive page faults is $(\bar{n}_{j(i)} - 1)$, where

$$\bar{n}_{j(i)} = \sum_{n=1}^{\infty} n\, Q_{n_{j(i)}} = \frac{1}{1 - F_{j(i)}(x, p)} \quad (10)$$

And the mean processing time of the process $j(i)$ is

$$m_{j(i)}=\bar{n}_{j(i)}\,\Delta=\frac{\Delta}{1-F_{j(i)}(x,\,p)} \tag{11}$$

where Δ, the average reference time[17], is the mean time interval between two consecutive successful references.

On the average, the number x of pages of $S_{j(i)}$ currently resided in the main memory is equal to the size $\omega_{j(i)}(t,\tau)$ of its working part if this working information is used for storage allocation. Since according to the replacement algorithm developed in a later section, a page which is an element of a working part may be replaced; or a page which is not an element of a working part may still reside in the main memory. And we have assumed that $\omega_{j(i)}(t,\tau)$ is statistically regular. Thus, Eq. (11) becomes

$$m_{j(i)}(\tau)=\frac{\Delta}{1-F_{j(i)}[\omega_{j(i)}(t,\tau),\,p]} \tag{12}$$

That is, the mean processing time is a function of τ with the given parameter p.

Having derived the formulation of the mean processing time, we now use it to derive the page-faulting rate of a process and the process efficiency.

Define the page-faulting rate $\lambda_{j(i)}(\tau)$ of the process $j(i)$ to be

$$\lambda_{j(i)}(\tau)=\frac{1}{m_{j(i)}(\tau)}=\frac{1-F_{j(i)}[\omega_{j(i)}(t,\tau),\,p]}{\Delta} \tag{13}$$

That is, $\lambda_{j(i)}(\tau)$ is the average processing time rate at which a new page is entering $W_{j(i)}(t,\tau)$; in other words, the new page is fetched from the auxaliary memory to the main memory and becomes an element of $W_{j(i)}(t,\tau)$ (because we have assumed that $\omega_{j(i)}(t,\tau)$ is the number of pages of $S_{j(i)}$ currently resided in the main memory on the average). Since statistical regularity is assumed, $\lambda_{j(i)}(\tau)$ may also be considered as the page-removing rate (in processing time) in order that $\omega_{j(i)}(t,\tau)$ is constant in time.

Next, we define the process efficiency $\eta_{j(i)}(\tau)$ to be

$$\eta_{j(i)}(\tau)=\frac{m_{j(i)}(\tau)}{m_{j(i)}(\tau)+d}=\frac{\Delta}{\Delta+d\{1-F_{j(i)}[\omega_{j(i)}(t,\tau),\,p]\}} \tag{14}$$

where d is the expected system overhead for handling the interruption caused by a page fault. We assume that d is a constant.

(4) *Convexity:*

From Eqs. (5), (7), (8) and (12) it will be seen that there is a convex relationship between the mean processing time $m_{j(i)}(\tau)$ and the working part size $\omega_{j(i)}(t,\tau)$. Fig. 2 depicts a general convex curve. In Fig. 2, for $\omega_{j(i)}(t,\tau)=p$, $m_{j(i)}(\tau)$ tends to approach infinity. It is because that we have assumed that a process is in the form of the processor demand of a segment; so in principle, the segment may be executed many, many times according to the programmer's will.

Viewing from Fig. 2, the convex relationship yields the following result: For $\omega_{j(i)}(t,\tau_1)-\omega_{j(i)}(t,\tau_0)=\omega_{j(i)}(t,\tau_0)-\omega_{j(i)}(t,\tau_2)=s$, we have

$$m_{j(i)}(\tau_0)<[m_{j(i)}(\tau_1)+m_{j(i)}(\tau_2)]/2 \tag{15}$$

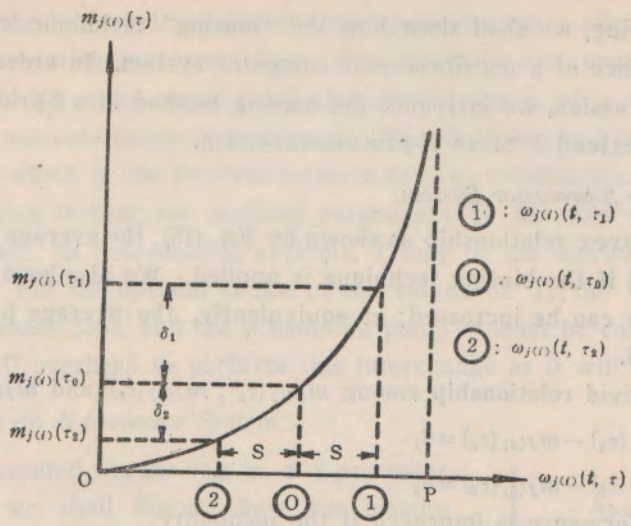

Fig. 2. General convex curve

This means that the average mean processing time of a process is increased if the technique of biasing the working parameters is applied.

III. BIASING IN A MULTIPROCESSOR COMPUTER SYSTEM

Having described some preliminary considerations, now, we shall show how the concept of biasing is applied to a multiprocessor computer system.

To solve the storage allocation problem of a multiprocessor computer system, at first insight, we may partition the main memory into memory modules of equal size and reserve a memory module for each processor; then solve the storage allocation problem with the same strategy as used in a singleprocessor computer system. In other words, each processor has its own part of the main memory such that other processors can not have accesses to this reserved memory area.

If the above strategy is used, there are two disadvantages. First, system utility programs may be duplicated. Many copies of a utility program may reside in the main memory simultaneously. This is a bad redundancy under the memory usage consideration. Second, if there is no enough available space in anyone of the memory modules for the initial loading of a program which has the highest priority in the queue, allocation is impossible and the program shall wait until there is enough available space in some one of the memory modules for its initial loading. On the contrary, the total available space of the main memory is enough for the initial loading, some memory space is wasted and throughput is decreased.

Alternatively, as will be presented in this paper, we may let all the processors share the whole main memory space. The space allotment of each processor at any time is determined by the proposed stroage allocation algorithm. In this manner, the space allotment of each processor is dynamically changed. The previously mentioned disadvantages will no longer exist under this strategy (note that we may use reentrant utility programs, so that many processors can reference a utility program simultaneously).

In the following, we shall show how the "biasing" technique can improve the system performance of a multiprocessor computer system. In order to simplify the analysis and discussion, we introduce the biasing method in a 2-processor system at first. Then we extend it to an N-processor system.

(1) *Biasing in a 2-processor System:*

From the convex relationship as shown by Eq. (15), the average mean processing time is increased if the biasing technique is applied. We also hope that the average process efficiency can be increased; or equivalently, the average page-faulting rate can be decreased.

There is a vivid relationship among $m_{f(t)}(\tau_1)$, $m_{f(t)}(\tau_0)$ and $m_{f(t)}(\tau_2)$. Define

$$m_{f(t)}(\tau_1) - m_{f(t)}(\tau_0) = \delta_1 \tag{16}$$

$$m_{f(t)}(\tau_0) - m_{f(t)}(\tau_2) = \delta_2 \tag{17}$$

The system performance is improved if the inequality

$$\frac{1}{\delta_2} - \frac{1}{\delta_1} > \frac{2}{m_{f(t)}(\tau_0)} \tag{18}$$

holds. (Of course, $\delta_1 > \delta_2$. This is due to the convex relationship.)

There are two approaches to obtain Eq. (18). The first one is to consider the average page-faulting rate, i. e., from the inequality

$$[\lambda_{f(t)}(\tau_1) + \lambda_{f(t)}(\tau_2)]/2 < \lambda_{f(t)}(\tau_0) \tag{19}$$

we will obtain Eq. (18). The second one is to consider the average process efficiency, i. e., from the inequality

$$[\eta_{f(t)}(\tau_1) + \eta_{f(t)}(\tau_2)]/2 > \eta_{f(t)}(\tau_0) \tag{20}$$

we will also obtain Eq. (18)*.

Thus the system performance is improved, if the biasing is applied, under the condition that the average space allotment is the same as that in the case where no biasing is applied.

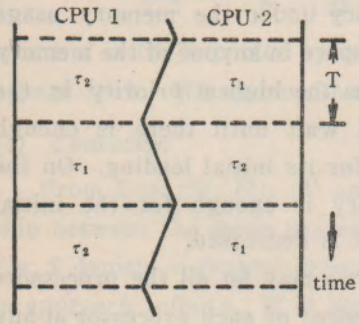

Fig. 3. Feature of space allotment

In this manner, in order to efficiently use the main memory and to avoid overloading and underloading, the two processors are paired such that: if τ_1 is the working parameter of one of the two working collectives, then τ_2 is that of the other working collective.

The space usage of the two processors may be expected to roughly follow the version as shown in Fig. 3 (here, the statistical uniformity is assumed, i. e., $\omega_h(t, \tau)$ is a processor independent quantity; and since the statistical regularity is also assumed, $\omega_h(t, \tau)$ is only a function of τ).

* Eqs. (16), (17) and (20) yield

$$m(\tau_0)\ (\delta_1 - \delta_2) - 2\ \delta_1\delta_2 + d(\delta_1 - \delta_2) > 0.$$

Since $\delta_1 > \delta_2$, we migh not consider the last term of the left-hand side in order to obtain Eq. (18).

In Fig. 3, the horizonal orientation represents the space allotment $\omega_h(t, \tau)$ and the vertical orientation represents time. The saw-tooth like curve shows the general characteristic of the space usage (note that the elements of a working collective may reside in non-contiguous page frames). The T shown in Fig. 3 is a "switching time interval", which is the interval between any two consecutive switching instants. At each switching instant, the working parameters of the two working collectives are interchanged. In time-sharing systems, T may be the service cycle time[3,4], or multiple of it. For the optimal choice of the values of T, the quantum size, the queue size of active jobs, and the scheduling policy[3,4] must be considered (note that there is no CPU overhead to perform this interchange as it will be seen later).

(2) *Biasing in an N-processor System:*

Having presented the biasing in a 2-processor system, now, we shall discuss how the biasing technique is applied to an N-processor system. We have assumed that the main memory is partitioned into page frames, and all the processors can have accesses to any area of the main memory. The processor-memory system is shown in Fig. 4.

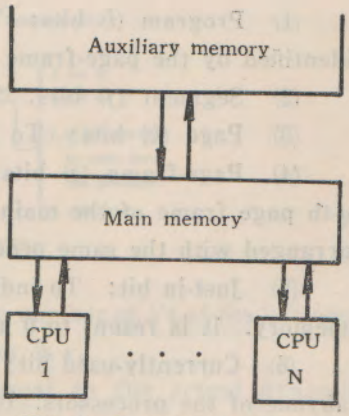

Fig. 4. A processor-memory system.

To apply the biasing technique to an N-processor system, we consider the following two cases:

(i) Case I: N is even. Within some switching time interval, each of the one half of the working collectives has τ_1 as its working parameter, and each of the other half of the working collectives has τ_2. During the next interval, the working parameters are interchanged.

(ii) Case II: N is odd. We may make the working parameter of a working collective (arbitrarily chosen) to be τ_0, and treat the other (N−1) working collectives with the same method as stated in Case I.

In this manner, we may say theoretically that the information with τ_1 for its working parameter will be less desirable for replacement than the information with τ_2 for its working parameter.

As we have mentioned that all the above analyses are based on the statistical nature, and time-independent is assumed. So at some replacement instants, the page, chosen by the algorithm presented later, for removing from the main memory may not be the best candidate. There may be some pages, which will be used in a near future, to be replaced. What we can hope is that on the average the system performance is improved.

IV. A DYNAMIC STORAGE ALLOCATION ALGORITHM WITH BIASING

In previous sections we have shown that biasing the working parameters of the working collectives will improve the system performance. Here, we shall show how it is implemented.

At first, we introduce what statistical information should be used and how to extract it.

We have an Allocation Status Word (ASW) for each page frame of the main memory. An ASW consists of those components as shown in Fig. 5.

Program (i) bits	Segment (j) bits	Page (k) bits	Pageframe (q) bits	Just in bit	Currently used bit	Control bit	Dynamic activity bits	Altered bit
Page identification			Physical location	Processing status			Usage status	

Fig. 5. An ASW

(1) Program (i) bits: To indicate that the page resided in the page frame, identified by the page-frame (q) bits, belongs to the program i.

(2) Segment (j) bits: To indicate that the page belongs to $S_{J(i)}$.

(3) Page (k) bits: To indicate that the page k of $S_{J(i)}$ is under consideration.

(4) Page-frame (q) bits: To indicate that this ASW is associated with the q-th page frame of the main memory (this is an option, since all the ASW's are arranged with the same order as the corresponding page frames).

(5) Just-in bit: To indicate that the page has just been loaded into the main memory. It is resent to 0 at the first time when it is referenced.

(6) Currently-used bit: To indicate that the page is currently referenced by anyone of the processors. It is reset to 0 when it is not currently referenced (note that for an N-processor system, there are at most N currently used pages in the main memory).

(7) Control bit[10]: To indicate that the process which processes this page is temporarily suspended and must wait for some pending activity such as a signal from another process, an I/0 interrupt, a page-in or out, or an external interrupt.

(8) Dynamic activity bits: To indicate the activity of the page since the last τ seconds, i.e., during the time interval $(t-\tau, t)$.

To obtain the value of the dynamic activity of an ASW, we have a Reference Pattern Word (RPW) and a dynamic activity counter for each ASW as shown in Fig. 6.

First, we consider a RPW as shown in Fig. 6(a). Whenever the page associated with the given RPW is referenced, the leftmost bit of the RPW is set to 1; but the RPW does not shift at this time. At the end of each sampling interval, the RPW is right shifted and its leftmost bit is set to 0 (the RPW is the same as the "use bits" in[1]). For a justin page, the RPW is cleared, so do the dynamic activity bits (note that shifting occurs only when the correspoding process currently gains control of a processor, since the previous analyses are based on the processing time consideration).

Define the dynamic activity of a page residing in the main memory to be the number of 1's of the associated RPW if this page is an element of a working

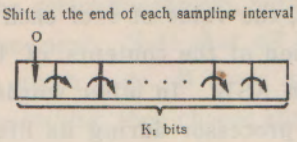

Fig. 6(a) A RPW.

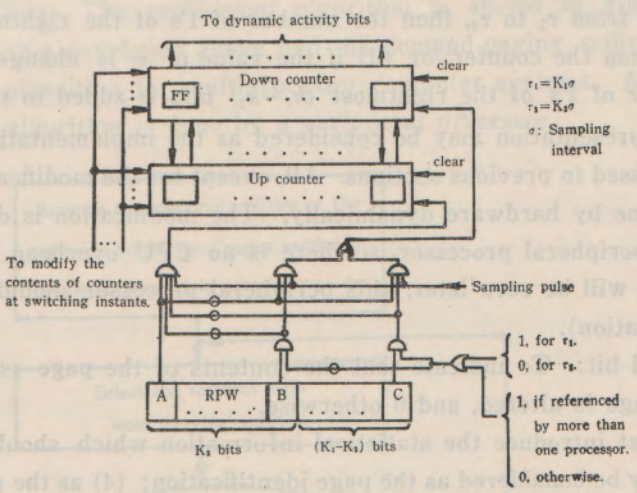

Fig. 6(b) A dynamic activity counter

collective whose working parameter is τ_1; or to be the number of 1's of the leftmost k_2 bits of the associated RPW if the working parameter is τ_2.

The dynamic activity defined above does not equal to the actual dynamic referencing frequency of the given page. Since it is worthless to make the value of the sampling interval having the same order as the average reference time Δ (of course, if this is made, the dynamic activity does equal to the actual dynamic referencing frequency). The choice of the value of the sampling interval will affect the value of the dynamic activity.

Now, we shall see how the dynamic activity counter works (note that the counter works only if the corresponding process currently gains control of a processor). We use a table to explain the operations. It is shown in the following:

Table 1. The operations of a dynamic activity counter

State at sampling instant		Operations
Bit A	Bit X	
0	0	No pulse is sent out.
0	1	A pulse is sent to the downcounter, whose contents are then transferred to the upcounter.
1	0	A pulse is sent to the upcounter, whose contents are then transferred to the downcounter.
1	1	No pulse is sent out.

Note: X=B for τ_2, and X=C for τ_1 (A, B, and C are shown in Fig. 6(b)). But if the page is referenced by more than one processor during its lifetime, then X=C no matter what is the working parameter.

At each switching instant, the value of τ of each working collective is changed to the other value. Modification of the contents of the dynamic activity bits may or may not be required for an ASW. In other words, if the corresponding page is referenced by more than one processor during its lifetime[8], then modification is not required; otherwise, it is required. The modification is a twofold: (i) If the value of τ is changed from τ_1 to τ_2, then the number of 1's of the rightmost (k_1-k_2) bits is subtracted from the counter, or (ii) if the value of τ is changed from τ_2 to τ_1, then the number of 1's of the rightmost (k_1-k_2) bits is added to the counter.

The above presentation may be considered as the implementation of the biasing technique discussed in previous sections. All, except for the modification at switching instants, are done by hardware dynamically. The modification is done by software executed by a peripheral processor, so there is no CPU overhead to perform this operation (as it will be seen later, this peripheral processor should do all the work of storage allocation).

(9) Altered bit: To indicate that the contents of the page is altered or not. It is 1 if the page is altered, and 0 otherwise.

We have just introduce the statistical information which should be used. (1), (2) and (3) may be considered as the page identification; (4) as the physical location; (5), (6) and (7) as the processing status; and (8) and (9) as the usage status.

Although additional hardware is required, the additional cost is only a little. For example, for a 262, 144-byte main memory with 4096 bytes for a page frame, there are 64 page frames. Thus, 64 sets of the additional hardware presented above are required; the additional cost is however small.

The ensemble of all ASW's may also be viewed as an associative memory. It is worthwhile that if the time required for relocation action by using this associative memory is less than that by using segment table and page table. That is, if the capacity of the main memory is not too large, the number of ASW's is also not too large such that the above condition holds, then we can use this ensemble as the associative memory.

Having presented the statistical information, now, we develop a replacement algorithm.

Define the dynamic frequency of a segment to be the sum of the dynamic activities of all "in-core" pages of that segment. Numely, $f_{j(i)}$ is the dynamic frequency of $S_{j(i)}$ (of course, $f_{j(i)}$ is changed with time).

Here, we shall show how the statistics $f_{j(i)}$ and $FF_{j(i)}(x, p)$ are used. During displacement, the attempt is to select pages which have the least marginal utility in the main memory, thereby minimizing the frequency of replacement[10]. Thus, a page is selected for replacement from $S_{j(i)}$ if

$$f_{j(i)}[F_{j(i)}(x, p) - F_{j(i)}(x-1, p)] = \min f_{n(m)}[F_{n(m)}(x, p) - F_{n(m)}(x-1, p)] \quad \text{All } n, m \qquad (21)$$

where x is the number of pages currently resided in the main memory for each of the respective segment.

Then a page, belonged to the selected segment, with the smallest value at "usage status" bits is selected for replacement. Note that all the "processing status" bits must be 0 in order to replace a given page. If there are two or more pages which have the same value at usage status bits, then choose anyone of them for replacement. If there is no such a page, then select a page from the segment with the next-to-least marginal utility. The replacemet algorithm is shown in Fig. 7. With the replacement algorithm developed above and the demand-paging policy, we have a storage allocation algorithm for multiprocessor computer systems. All work of the storage allocation algorithm is done by a peripheral processor.

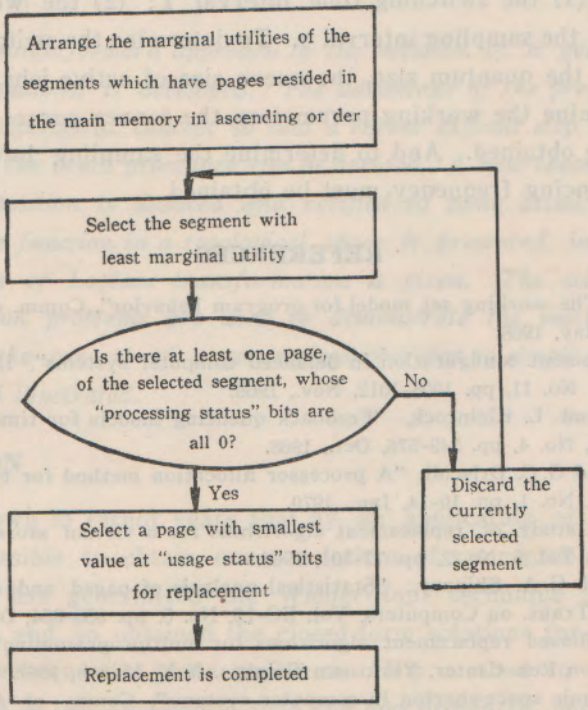

Fig. 7. The replacement algorithm

When a process completes, the associated ASW's are cleared and the corresponding page frames become "non-occupied". Due to this notion, the available space of the main memory may be considered as the non-occupied page frames. We use this for the initial loading of a program.

V. CONCLUSION

In this paper we have developed a storage allocation algorithm which is very suitable for multiprocessor computer systems under multiprogramming and/or time-sharing environment. All the information used by the algorithm are dynamically extracted. Prerunning of any program is not required.

The technique of biasing the working parameters and the concept of dynamic activity are applied for developing the replacement algorithm. Due to the goneral convex relationship between the mean processing time of a process and the working part size of a segment, we have shown that biasing the working parameters can

improve the system performance (i.e., the average process efficiency is increased or equivalently the average page-faulting rate is decreased) on the average. The candidate chosen for replacement will be better if the dynamic activity is used, since the dynamic activity refers to the currently referencing pattern of a page.

Memory-accessing conflicts become a serious problem in this system, since we have assumed that all the processors can have accesses to any area of the main memory. To solve this problem, an interleaved main memory should be used.

In the present paper, we only discuss the algorithm in principle. Many further simulations are required in order to determine the values of the parameters. These parameters are: (1) the switching time interval T; (2) the working parameters τ_1 and τ_2; and (3) the sampling interval σ. To determine the switching time interval, we must consider the quantum size, the queue size of active jobs and the scheduling policy. To determine the working parameters, the convex curves (Fig. 2) of typical programs must be obtained. And to determine the sampling interval, the pattern of dynamic referencing frequency must be obtained.

REFERENCES

1. P. J. Denning, "The working set model for program behavior", Comm. of ACM, Vol. 11, No. 5, pp. 323-333, May, 1968.
2. _____, "Equipment configuration in balanced computer systems", IEEE Trans. on Computer, Vol. C-18, No. 11, pp. 1008-1012, Nov., 1969.
3. E. G. Coffman and L. Kleinrock, "Feedback queueing models for time-shared systems", J. of ACM, Vol. 15, No. 4, pp. 549-576, Oct., 1968.
4. A. P. Mullery and G. C. Driscoll, "A processor allocation method for time-sharing", Comm. of ACM, Vol. 13, No. 1, pp. 10-14, Jan., 1970.
5. I. A. Belady, "A study of replacement algorithms for a virtual storage computer", IBM Systems Journal, Vol. 5, No. 2, pp. 77-101, 1966.
6. J. E. Shemer and G. A. Shippey, "Statistical analysis of paged and segmented computer systems", IEEE Trans. on Computers, Vol. EC-15, No. 6, pp. 855-864, Dec., 1966.
7. L. A. Belady, "Biased replacement algorithms for multiprogramming", Rep. NC 697, IBM Thomas J. Watson Res. Center, Yorktown Heights, N. Y. March, 1967.
8. _____, "Dynamic space-sharing in computer systems", Comm. of ACM, Vol. 12, No. 5, pp. 282-288, May, 1968.
9. M. V. Wilkes, "A model for core space allocation in a time-sharing system", AFIPS, SJCC, pp. 265-271, 1969.
10. J. E. Shemer and S. C. Gupta, "On the design of Bayesian storage allocation algoritms for paging and segmentation", IEEE Trans. on Computers, Vol. C-18, No. 7, pp. 644-651, July, 1969.
11. Chin Tung, "On the apparent continuity of processing in a paging environment", IEEE Trans. on Computers, Vol. C-19, No. 11, pp. 1047-1054, Nov., 1970.
12. T. Kilburn, D. B. G. Edwards. M. J. Lanizan, and F. H. Summer, "One-level storage system", IRE Trans. on Computers, Vol. EC-11, No. 2, pp. 223-235, April., 1962.
13. J. B. Dennis, "Segmentation and the design of multiprogrammed computer systems", J. of ACM, Vol. 12, No. 4, pp. 589-602, Oct., 1965.
14. W. Arden, B. A. Galler, T. C. O, Brien, and F. H. Westervelt, "Program and addressing structure in a timesharing environment", J. of ACM, Vol. 13, No. 1, pp. 1-16, Jan., 1966.
15. Br. Randell and C. J. Kuehner, "Dynamic storage allocation systems", Comm. of ACM, Vol. 11, No. 5, pp. 297-306, May, 1968.
16. J. L. Smith, "Multiprogramming under a page on demand strategy", Comm. of ACM, Vol. 10, No. 10, pp. 636-645, Oct., 1967.
17. P. J. Denning, "Virtual memory", Computing Sureys, Vol. 2, No. 3, pp. 153-189, Sep., 1970.
18. A. V. Aho, P. J. Denning, and J. D. Ullman, "Principles of optimal page replacement", J. of ACM, Vol. 18, No. 1, pp. 80-93, Jan., 1971.