CHARACTERIZATION AND REPRODUCTION OF THE REFERENCING DYNAMICS OF PROGRAMS

by

Domenico Ferrari

Memorandum No. UCB/ERL M81/22

22 April 1981

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Characterization and Reproduction of the Referencing Dynamics of Programs*

Domenico Ferrari

Computer Science Division
and the Electronics Research Laboratory
University of California, Berkeley

ABSTRACT

A model of dynamic program behavior based on the patterns of page arrival into and departure from a program's working set is proposed. This characterization is equivalent to one based on the time behavior of working set size and on the occurrence times of those page arrivals which do not change the working set size since they are accompanied by page departures. The model has intuitive appeal and several interesting properties. The possibility of generating strings which will obey a given characterization is one of the most important such properties. An algorithm is described which may be used to generate these reference strings, and necessary and sufficient conditions under which the algorithm can successfully complete its task are given.

1. Introduction

Since the beginning of the history of virtual memory systems, the referencing behavior of programs has been recognized as one of the factors having a substantial influence on the performance of these systems. Because of its importance, a great deal of program behavior modeling and measurement work has been performed. Most of the work has been done with the ultimate objective of designing or improving memory management policies [1], but even the complementary approach consisting of improving the performance of programs in a virtual memory environment has received considerable attention [2].

In spite of all the efforts made so far, a viable approach to the characterization of the dynamic behavior of a program has not been proposed yet. At one extreme, we have a complete characterization, namely, the reference string generated by the program when processing a given set of input data (in this paper, by "behavior of a program" we shall always refer to the behavior of a program-input pair). The reference string (i.e., the sequence of addresses issued by the CPU during the program's execution) contains complete referencing dynamics information but is excessively long for most purposes, hard to classify and compare to other strings, inflexible (i.e., hard to modify in a controlled manner), and input-data dependent in unknown ways. At the opposite extreme, we find program behavior models whose dynamics either is totally unrelated to that of the original string (e.g., the Independent Reference Model

* The work reported here has been supported in part by the Computer Systems Design Program of the National Science Foundation under grant MCS-7824618.

[3]) or cannot be easily related to it (e.g., the two-level Markov-LRU Stack Model described in [4]).

The work reported in this paper may be seen as an attempt at bridging the gap between the two extremes just described. Ideally, the dynamic behavior of a program should be characterized by an amount of information substantially smaller than that contained in a reference string. This information should be easy to use for the purpose of classifying programs according to their dynamic behavior, hence to their expected performance in a given virtual memory environment. Furthermore, the program behavior model should be usable to generate artificial reference strings with approximately the same dynamic behavior as the modeled program; this behavior should be easily modifiable for use in sensitivity studies. Finally, the characterization should be based on quantities having a physical meaning, so that their values may be estimated from descriptions of the program or even of the algorithms implemented by the program. For some programs, the modifications of the values of these quantities could be derived from the knowledge of the input data variations or of the changes to be made to the code.

The ability to reproduce a certain dynamic behavior by generating an artificial reference string which has a given characterization is quite useful in most applications of trace-driven simulation to the study of memory policies and program behavior. An artificial string is generally less expensive to procure, more easily modifiable in a controlled way, and less space-consuming than a real address trace. The last characteristic comes from the fact that in most cases an artificial string does not have to be stored between consecutive instances of its use, but can be produced by the generating program and processed on the fly every time it is needed. Furthermore, an artificial string can be designed and implemented according to given specifications (e.g., a characterization of its dynamics), thereby allowing one to perform completely controlled simulation experiments, which cannot be performed with a real string because of its lack of flexibility.

Section 2 discusses these problems and proposes a dynamic behavior characterization which exhibits to some extent all the ideal properties listed above. The context in which the discussion is held throughout this paper is that of a paged virtual memory. Even though the proposed characterization is based on the working set model [5], its use should not be confined to virtual memories managed by working-set-like policies. However, the conditions under which it retains its accuracy and other properties in different memory management environments are still to be investigated. An algorithm which produces an artificial reference string with a given dynamic behavior (characterized as described in Section 2) is illustrated in Section 3. The main properties of the algorithm and the most important research questions which remain to be answered are discussed in Section 4.

2. Characterizing Program Referencing Dynamics

Two of the aspects of an address trace that are most relevant in a paged virtual memory environment are the following:

(a) references to information items belonging to the same page cannot be distinguished by the memory manager and therefore always have the same effect on program and system performance;

(b) references to pages which, because of the amount of memory space allotted to the program and of the memory policy adopted in the system, tend to be in memory at the time they are referenced usually have a beneficial impact on performance, whereas references to pages which tend to be out of memory have a negative influence.

Statement (a) authorizes us to deal with page reference strings rather than with address traces, guaranteeing that this will not have any impact on the accuracy of the results. Statement (b) refers to the strong relationship existing between program locality [6] and virtual memory performance. Thus, a characterization of program dynamics which allowed us to specify "how local" the behavior of the program is at any given time (or during any given interval) of its execution would be quite meaningful and useful in practice. A policy-independent characterization of this type could be based on page inter-reference times [3]. However, to be specified, the sequence of inter-reference times requires as much information as the string it intends to characterize, and the alternative model consisting of the distribution (or distributions) of such times does not lend itself well to the representation of referencing dynamics, as is usually the case when time is not explicitly represented in a model. Furthermore, the latter characterization cannot be easily used to generate artificial strings with the desired properties (in this case, with a given distribution of inter-reference times).

Among the policy-oriented approaches that could be proposed, one based on the working set model [5] looks particularly attractive. The appeal of this characterization stems from the attractiveness of that model, from the increasing popularity of working-set-like policies, and from the natural and relatively compact way in which it can represent the dynamics of programs. It should be immediately stressed that the characterization to be presented below, though based on the working set model and hence somehow biased towards variable-size memory policies of the working set type, is not intended only to describe the dynamic behavior of a program in a working set environment, but to model those aspects of such behavior which are the most relevant ones in any virtual memory environment. Unfortunately, however, this characterization is expected to become less and less accurate as the actual operating conditions of the program get farther and farther from those assumed while building the model (i.e., a working set policy with a given window size).

Let virtual time be measured in memory references rather than in microseconds, and assume for simplicity that references are equally spaced on the virtual time axis. In this discrete-time context, the working set $W(t,T)$ at time t with window-size T can be defined as the set of the pages referenced in the closed interval $(t-T+1,\ t)$. If the reference string generated by the program is the finite sequence $r = \{r_t\}$ $(t = 1,2,\dots n)$, where r_t is the name of the page referenced at time t, the above definition of $W(t,T)$ can be extended to the interval $1 \le t \le T$ assuming that, for $2 - T \le t \le 0$, r_t is defined and is the name of a non-existing page whose size is 0.

A natural way to represent the dynamic behavior of a program is by using the curve $w(t,T)$, which shows how the size of its working set varies with virtual time for a given window size T. Under the assumptions made in the previous paragraph, this characterization is totally equivalent to the one based on the string of integers $w(t,T) = \{w_t\}$, $(t = 1,2,\dots n)$, which represents the consecutive values of working set size. The objection that the amount of information string w contains is as large as that found in the program's reference string can be answered by observing that the important aspects of a dynamic phenomenon can often be captured by assigning the coordinates of a relatively small number of points of the curves which describe it; also, in certain cases, it may be possible to specify $w(t,T)$ by one or a few analytic expressions.

However, the knowledge of $w(t,T)$ is not sufficient to characterize a program's behavior for our purposes. Not all of the arrivals and the departures of pages cause working set size variations. While an increment in the working set size is always the result of the arrival of a new page (when running the program with those input data and that window size in a pure working set environment, this arrival would produce a page fault), there are arrivals which do not change the working set size. This is the case when the arrival of a new page is accompanied by the simultaneous departure of an old page from the working set. The above statement can be repeated for decrements in $w(t,T)$ and departures. Simultaneous arrivals and departures are important aspects of program dynamics since they are found in non-local phases of a program's execution (inter-locality transitions), and particularly in purely sequential referencing patterns. The magnitude of their presence in the reference string to be characterized can be specified in several alternative ways, the most compact and least accurate one being that of assigning the mean page fault rate generated by the program. To simplify our discussion, we shall assume that their occurrence times are given, a situation which all the other characterizations will have to lead to if artificial reference strings with the assigned properties are to be generated. Thus, the characterization we are proposing consists of the two finite sequences of integers

$$w = \{w_t\} \qquad (t = 1,2,\ldots n),$$
$$f = \{f_j\} \qquad (j = 1,2,\ldots k),$$

where f_j is the time index at which the j-th arrival-departure event occurs. For the sake of brevity, an arrival-departure event will be called a *flat fault* in the sequel, and the above will be called a *(w,f) characterization*.

Flat faults are the manifestations of purely non-local behavior. When each reference in a long sequence causes the arrival of a new page into the working set, the size of this set climbs to T and, since the time it reaches T, every reference generates a flat fault. This is the case of purely *sequential* referencing behavior (if reference r_t is to page i, reference r_{t+1} is to page $i+1$ for all t), as shown in the last portion of the string in Figure 1.

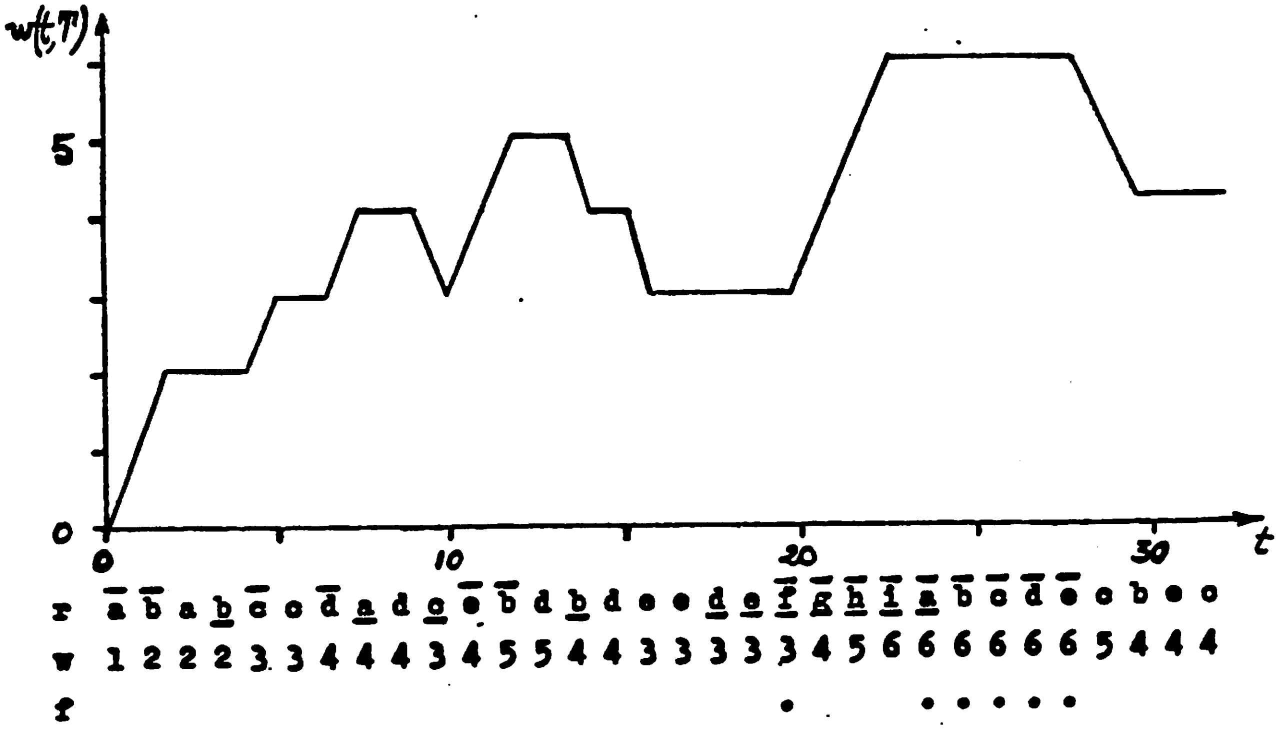

Figure 1. A reference string r, its working set size string w for T=6, its flat fault string f (whose entries are the times marked by a dot), and its $w(t,T)$ curve. Note that the references with a bar correspond to page arrivals, and the underlined ones to page departures.

Another type of non-local behavior, namely, *random* referencing, has similar characteristics but, in its pure form (i.e., when references are uniformly distributed over the program's virtual address space), produces less flat faults due to the non-zero probability of referencing pages which are in the working set. A simple but useful addition to the characterization described above would be to specify for each flat fault whether it is sequential (i.e., caused by the page following the one which produced the previous flat fault) or not. Alternatively, one might assign the percentage of sequential flat faults for the whole string or for a number of time intervals into which the virtual lifetime of the program could be divided. All these and other similar additions to a (w,f) characterization can be easily accommodated by the string generation algorithm to be described in Section 3.

What conditions are to be satisfied by a (w,f) characterization in order for it to represent a reference string? Can w and f be assigned arbitrarily? Before answering these questions, we must introduce the departure set $D(t,T)$. First, we observe that a departure occurs at time t if either

$$w_t = w_{t-1} - 1$$

or

$$f_j = t$$

for some j. Second, we notice that in both cases the departing page is the one which has been referenced at time $t-T$. The *departure set* $D(t,T)$ is the set of pages which drop out of $W(t,T)$ during the closed interval $(t+1, t+T-1)$; in other words, it is the set of pages referenced between $t-T+1$ and t which are not referenced between t and $t+T-1$; its cardinality d_t at time t is equal to the sum of the number of working set size decrements and of the number of flat faults occurring between $t+1$ and $t+T-1$ (the page referenced at time t cannot drop out of W before $t+T$). Note that a *decrement* is said to take place at time $t+1$ if $w_{t+1} = w_t - 1$. Note also that sets W and D can be defined for $t < T$ assuming that all references for $t \leq 0$ are made to a non-existing page whose size is 0, and which does neither arrive nor depart.

An answer to the previous questions can now be provided.

Theorem 1.

Given a reference string $r = \{r_t\}$ $(t = 1,2,\ldots n)$, and assuming that the working set of the program which has produced r is initially empty, its (w,f) characterization has the following properties:

(a) $w_1 = 1$;

(b) $0 < w_t \leq min(p,T)$ (t = 1, 2, ...n), where p is the total number of pages in the program;

(c) $f_1 > 0$, $f_j > f_{j-1}$ (j = 2, 3, ...k);

(d) $|w_t - w_{t-1}| \leq 1$ (t = 2, 3, ...n);

(e) $w_{f_j} = w_{f_j - 1} < p$ (j = 1, 2, ...k);

(f) $d_t < w_t$ (t = 1, 2, ...n).

Proof.

(a) Since $W(t,T) = \emptyset$ for $t \leq 0$, after the first reference, made to, say, page q, we have $W(1,T) = \{q\}$, and $w_1 = 1$.

(b) The working set, by definition, can never contain more than p pages or T pages, whichever is smaller. Also, its size must be positive at all times.

(c) The f_j's are positive time indices arranged in chronological order and corresponding to the occurrences of flat faults.

(d) By definition, the working set size after each reference can either change by 1 or remain constant. All other changes are impossible.

(e) A flat fault cannot occur when the working set size is either increasing or decreasing, or when it contains all the program's pages.

(f) Since no page may drop out of the working set more than once during an interval of T references, and the page referenced at time t is guaranteed to be in $W(t+T-1,T)$, the maximum cardinality of $D(t,T)$ equals $w_t - 1$. Q.E.D.

The inverse question now arises: Given a (w,f) characterization which satisfies the conditions of Theorem 1, can one construct a reference string which will be described by that characterization? We shall see in the next section that this question has an affirmative answer.

3. Generating a Reference String With a Given Dynamic Behavior

An algorithm will now be presented for obtaining a reference string having a given (w,f) characterization. It will then be shown that, if the conditions stated in Theorem 1 are satisfied, the algorithm completes its task successfully, and that, if they are not satisfied, the algorithm cannot generate the entire string.

The assigned dynamic behavior can be reproduced by referencing one of the pages not belonging to the working set whenever w or f indicate that an arrival is to take place, and one of the pages already in the working set otherwise. In addition, T references before the time a departure is expected to occur, the page which has just been referenced is to be marked so that further references to it will be prevented until its departure time. If no departure is to take place T references from the current time t, the page just referenced will have to be referenced again on or before time $t+T$ (if it is referenced at $t+T$ for the first time after t, we shall consider it as a re-referenced page rather than viewing this as the departure and the simultaneous arrival of the same page).

In order to execute the algorithm just described, we need to know at any time t the current contents of the working set, whether an arrival must take place at t, and whether a departure must take place at $t+T$. Also, the pages in the working set are to be divided into two categories: the marked ones, which cannot be referenced before they will drop out, and the unmarked ones, which must be referenced before they drop out. It is therefore convenient to construct and maintain three sets, the set of unmarked pages, to be called the *candidate set C*, the set of marked pages or *forbidden set F*, and the *external set E*, which contains all the pages not in the working set. These three sets are disjoint, and their union coincides with the set P of all pages. Thus,

$$C(t,T) \cup F(t,T) = W(t,T) = P - E(t,T).$$

To simplify our symbology, we shall leave the dependence of these sets on t and T implicit and write C,F, and E whenever no ambiguity may arise.

Page r_t, referenced at time t, will either join F or C, depending on whether or not its departure is expected to take place at $t+T$. In any case, the page will be assigned $t+T$ as its time index, and, when required, we shall write $r_t[t+T]$: if the page is added to F, the time index is the time at which it will leave F and join E, thereby departing from the working set; if it is added to C, the time index represents the latest possible time at which the page is to be referenced again in order for the given dynamics to be accurately reproduced. While the page to be referenced at time t can be chosen arbitrarily from among the members of C (if no arrival is to occur) or of E (if an arrival must take place), it may be convenient to keep the members of C ordered according to their time indices, i.e., in FIFO order, so as to minimize the probability of unwanted departures which will require repeating the string generation procedure for the last window with different page selections. On the other hand, the choice of a page from E does not have to satisfy any such condition and can be made according to several criteria; for example, trying to reproduce given proportions of sequential and random referencing behaviors (one approach would be to assign the probability that the next external reference is to the next page, if indeed this page is not in the working set at that time). Set F does not have to be ordered, but the amount of searching to be performed at each reference to see whether the current time coincides with the time index of any member of F is minimized if F is treated as a FIFO queue.

```
Inputs: $n, k, T, P, w_t$  $(t = 1,n)$, $f_j$  $(j = 1,k)$.
Output: $r_t (t = 1,n)$.
Initialization
1. $C \leftarrow \emptyset$, $F \leftarrow \emptyset$, $E \leftarrow P$, $w_0 \leftarrow 0$, $f_{k+1} \leftarrow 0$, $t \leftarrow 1$, $j \leftarrow 1$.
2. For $i \leftarrow 1$, T do $w_{n+i} \leftarrow w_n$.
Reference Selection
3. If $w_t > w_{t-1}$ or $f_j = t$ then if $E = \emptyset$ then error, $r_t \leftarrow e \in E$, $E \leftarrow E - \{e\}$
   else if $C = \emptyset$ then error, $r_t \leftarrow first(C)$, $C \leftarrow C - first(C)$.
4. If timeindex $[first(C)] = t$ then error.
Set Updating
5. If $w_{t+T} < w_{t+T-1}$ or $f_m = t + T$ for some $m \geq j$ then $F \leftarrow F \cup \{r_t[t+T]\}$
   else $C \leftarrow C \cup \{r_t[t+T]\}$.
6. If timeindex $[first(F)] = t$ then $E \leftarrow E \cup \{first(F)\}$, $F \leftarrow F - \{first(F)\}$.
Loop Control
7. If $f_j = t$ then $j \leftarrow j + 1$.
8. If $t < n$ then $t \leftarrow t + 1$, go to 3 else stop.
```

Figure 2. A description of the string generation algorithm.

The algorithm discussed informally so far in this section is more formally described in Figure 2, where we have assumed that both C and F are FIFO queues. Note that in the figure *first* (Q) represents the oldest member of FIFO queue Q, and the union operator $\cup$ in $Q \cup \{x\}$ appends page x to FIFO queue Q. Note also that, if F is empty when Step 6 is executed, the test of the time index of its first element will have a negative result as if such element existed and its time index were different from t. Finally, whenever the procedure "error" is invoked, the algorithm halts, since it can no longer accurately reproduce the given dynamics.

A sample application of the algorithm in Figure 2 is presented in Figure 3. The given (w,f) characterization is the one of the reference string considered in Figure 1. Step 5 of the algorithm clearly shows that, at all time instants, r_t is appended either to C or to F with time index $t+T$. This suggests that we can keep track of the contents of both C and F by writing the name of each referenced page on a single line (the C/F line in Figure 3) T time instants ahead, i.e., at the time corresponding to its time index. When the first element of C is used as the next reference, its symbol on the C/F line is crossed out and rewritten T instants ahead. The elements of F, whose symbols appear underlined on line C/F in Figure 3, are crossed out and join set E when the current time goes beyond their position (see Step 6). The generated string r^* does not, of course, coincide with the original string r but has the same dynamic properties as characterized by the (w,f) pair.

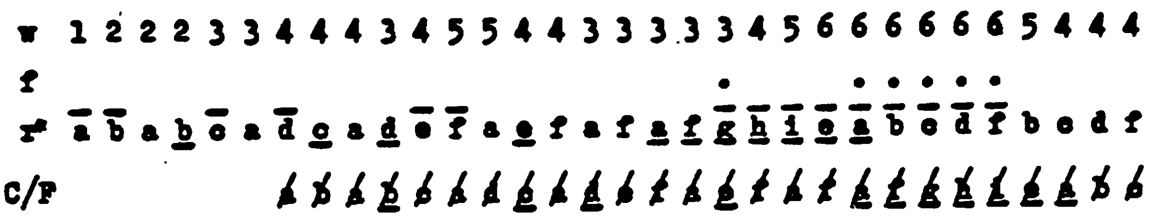
w 1 2 2 2 3 3 4 4 4 3 4 5 5 4 4 3 3 3 3 3 4 5 6 6 6 6 6 6 5 4 4 4
f
r* a b a b c a d c a d e f a e f a f a f g h i e a b c d f b c d f
C/F

Fig.3. Generation of a string r^* with the same (w,f) characterization as string r in Fig.1 $(T=6)$.

We can now prove that the conditions of Theorem 1 are necessary and sufficient for the algorithm in Figure 2 to generate a reference string with an assigned dynamic behavior.

Theorem 2.

The string generation algorithm described in Figure 2 produces a reference string with a given (w,f) characterization if and only if this characterization exhibits properties (a) through (f) of Theorem 1.

Proof.

(A) The conditions are necessary. Since by Theorem 1 all reference strings have properties (a) through (f), the given (w,f) characterization must have these properties in order for the generated string to be faithfully represented by it.

(B) The conditions are sufficient. Let the given (w,f) characterization exhibit properties (a) through (f). The discussion of the algorithm in the first part of this section shows that, by construction, the algorithm generates a string r whose behavior is characterized by the given (w,f) description. This conclusion can also be reached by formally analyzing the reference selection phase (Step 3) and the set updating phase (Steps 5 and 6) in Figure 2: page arrivals are handled by referencing pages outside the working set (Step 3), page departures by preventing the referencing of the pages expected to leave (Steps 3 and 5) and by transferring them from F to E (i.e., out of the working set) at the proper times (Step 6). Thus, the only case in which we cannot obtain the desired result is when the algorithm is not allowed to terminate. There are three error exits in the description of Figure 2.

(i) Let $w_t > w_{t-1}$ at some time t. If E were empty after the generation of r_{t-1}, we would have $w_{t-1} = p$, where p is the number of pages in the program. This would require $w_t > p$, which is impossible if condition (b) is satisfied. If on the other hand we had $f_j = t$ for some j and some t, E could not be empty since we would have $w_{t-1} = p$ and condition (e) would not hold. Thus, the first error exit in Step 3 can never be taken if the given characterization has properties (b) and (e).

(ii) Let $w_t \le w_{t-1}$ and $t \ne f_j$ for all j (i.e., no page arrival at time t). By definition, $D(t,T)$ is the set of pages leaving the working set between $t+1$ and $t+T$-1, and $F(t$-1$,T)$ is the set of departures expected to take place between t and $t+T$-1. Thus, if there is a departure at time t (i.e., if $w_t < w_{t-1}$), we have $|F(t-1,T)| = d_t + 1$; if, on the other hand, there is no departure (i.e., $w_t = w_{t-1}$), then $|F(t-1,T)| = d_t$. Since $W = C \cup F$, and C and F are disjoint, we have

$$w_{t-1} = |C(t-1,T)| + |F(t-1,T)|,$$

and, if $w_t < w_{t-1}$,

$$w_t < |C(t-1,T)| + d_t + 1.$$

If $C(t-1,T)$ were empty, condition (f) would not be satisfied, since we would have $w_t \le d_t$.

Let us now assume $w_t = w_{t-1}$. Then

$$w_t = |C(t-1,T)| + d_t,$$

and, again, assuming $|C(t-1,T)| = 0$ contradicts condition (f). Thus, C can never be empty when no arrival is to occur, and the second error exit in Step 3 will never be taken if condition (f) is satisfied. It is easy to verify, using the above relationships, that condition (f) is necessary for $C(t-1,T)$ to be non-empty in this case.

(iii) Let timeindex $[first(C)] = t$ for some t. Then, $first(C)$ is a page which has been referenced at time $t-T$ and never referenced again since. During the interval $(t-T+1,t)$ there have been, say, a page arrivals and d page departures. Let $c = |C(t-T+1,T)|$. If timeindex $[first(C)] = t$, then the total number $T-a$ of references which are not page arrivals in $(t-T+1,t)$ equals c-1, the number of pages in FIFO queue C which at $t-T$ preceded the one being considered (see Step 3). From

$$W(t-T+1,T) = C(t-T+1,T) \cup F(t-T+1,T)$$

we obtain $w_{t-T+1} = c + d$, since the cardinality of F at $t-T$+1 equals the number of departures between $t-T$+1 and t. From

$$w_t - w_{t-T+1} = a - d = T - c + 1 - d$$

we obtain

$$w_t = T + 1,$$

which contradicts condition (b). When this condition is satisfied, we have $c \le T - a$, and timeindex $[first(C)]$ is guaranteed to be always greater than t after the execution of Step 3. Thus, the third error exit in (Step 4) can never be taken. Q.E.D.

4. Conclusions

The algorithm described in Section 3 can be used to generate a reference string with a given (w, f) characterization of dynamic behavior provided that the strings of integers w and f exhibit properties (a) through (f) of Theorem 1. As can be derived from an analysis of Figure 2, the complexity of the algorithm is low both in time and in space. Time-wise, the complexity is a linear function of n, the length of the string to be generated. The coefficient of n in the expression of the computation time is approximately constant; in the worst case, it equals the time for performing 7 tests, 1 extraction and 1 addition of an element from and to a set, 1 extraction of the top element of a queue, 1 addition of an element to the end of a queue, 1 short search through string f, and the incrementing of 2 variables. The search may be eliminated if a pointer m to f_m in Step 5 is maintained and used in the same way as j for f_j in Step 3. In a Pascal implementation of the algorithm running on a DEC VAX-11/780, the mean value of this coefficient is around 100 μs. Thus, 1 million references can be generated in approximately 100 seconds of VAX CPU time. Also the algorithm's complexity in space is quite low. Of the given w string, only the four values w_t, w_{t-1}, w_{t+T}, w_{t+T-1} are needed at each time instant t; of string f, only the values f_j and f_m; the largest amount of space is that needed for the three sets C, F, and E, whose total size equals the number of pages in the program.

The algorithm can be easily modified to accommodate several variations in the characterization of dynamic program behavior described in Section 2. Some of these variations exploit the freedom of choice for the member of E to be referenced (see Step 3) and, to a lesser extent, that for the member of C (which in Figure 2, Step 3, has been assumed to be restricted to the top element in the FIFO queue, but which need not be so constrained). Many variations may be proposed for the specifications of the flat fault string f and of the working set size string w. The algorithm assumes that both are completely specified, as they will eventually have to be before they are processed. However, the users will often want to assign them in a different, more compact way, which will generally vary with the application. Some of these specifications will have a stochastic flavor. Others might be based on analytic expressions or on various kinds of interpolations of given points. Others yet will make use of mean values such as the mean page fault rate.

Two main problems, among several others, should, and in the near future will, be investigated. One has to do with the implementation of an artificial program which, when running, generates a given reference string. The availability of such a program will allow controlled measurement experiments to be performed on memory policies, both in uniprogramming and in multiprogramming contexts. It is very difficult to solve this problem exactly, but some promising solutions exist which seem to provide a satisfactory accuracy for most practical purposes. A study of the merits and limitations of these solutions has been undertaken and will hopefully result in a methodology for synthesizing programs with given dynamic characteristics.

The other problem is more fundamental: What are the strengths and weaknesses of the (w, f) characterization? In what contexts is it an adequate way of representing program dynamics? The most evident limitation of the proposed model is related to the choice of T. How should T be chosen when characterizing a particular program or an entire workload? How different will be the effects of a change of T on the dynamics of the artificial string and on that of the modeled program? Can the arbitrary choices which the algorithm makes be guided so that a better match will result at different window sizes? Can an algorithm be found which will generate a reference string obeying two working set size strings corresponding to two window sizes T and T'? Another limitation stems from the working-set orientation of the (w,f) characterization. Can artificial strings and programs produced by the algorithm be used to study other memory policies without introducing an unwanted bias into the results? How should T be selected in those cases? Clearly, the more accurate characterizations alluded to above should decrease the likelihood of introducing a substantial bias in an investigation of memory policies. But how far can we go in the direction of greater sophistication before we incur the penalty of diminishing returns?

Acknowledgements

The author is grateful to Bob Hagmann, Bill Joy, Ron Dutt, and several members of the PROGRES group for the enlightening discussions he had with them on the subject of this paper. The author is also indebted to Caren Weisglas for her invaluable help in the preparation of the manuscript.

References

[1] J. Spirn, *Program Behavior: Models and Measurements*. Elsevier North-Holland, 1977.

[2] D. Ferrari, "The improvement of program behavior", *Computer 9*, 11(Nov. 1976), 39-47.

[3] E. J. Coffman, Jr. and P. J. Denning, *Operating Systems Theory*. Prentice-Hall, 1973.

[4] P. J. Denning and K. Kahn, "A study of program locality and lifetime functions", *Proc. Fifth SIGOPS Symp. on Operating Systems Principles*, (Nov. 1975), 207-216.

[5] P. J. Denning, "The working set model for program behavior", *Comm. ACM 11*, 5(May 1968). 323-333.

[6] P. J. Denning, "Virtual memory", *Comp. Surveys 2*, 3(Sept. 1970), 153-189.