PUBLISHED: 6/7/67
Major Revision
(Supercedes BG.6.05,
6/6/66)

Identification

The ranker
Peter G. Neumann and Mary R. Wagner

Purpose

This section describes the ranker which is invoked periodically to obtain the use bits from the page tables and to reorder one or more of the eligible lists. Each of these lists is maintained in an order within the pool corresponding to frequency and recency of use which is precisely the order in which the contents of the respective groups may be removed from core.

Introduction

There are several lists threaded through the group map by using the pointer of each group map entry. Of these lists, the eligible lists are maintained in the order in which these hyperpages may be removed from core. The page table list is used to drive the ranker.

Each list is by definition in the order in which hyperpages may be removed, in that hyperpages which are at the beginning of the list are removed first. New hyperpages are added to the end of the list. This list may be periodically reordered by the ranker, based upon the use bits in the appropriate page tables. The reordering is such that every hyperpage which has been used in the last rank cycle (i.e., the period since the last ranking of this list) finds itself nearer to the end of the list than any hyperpage which has not been so used. In the ranker algorithm described below, relative order is maintained among those pages which have been used, as is the case among those pages which have not been used. Thus pages which are used at least once during each rank cycle find themselves near the end of the list, while those which are not used at all percolate to the beginning of the list, and are eventually removed from core. Pages which are used at least once during every other rank cycle find themselves near the middle of the list, and so on.

The algorithm

The algorithm described here is the A. Jay Goldstein ranker (see Multics document B0013, September 28, 1965). (A J. Arthur ranker or J. Other ranker may alternatively be used.) The ranker is invoked periodically, as determined by the ranker parametizer (see Section BG.6.06). It is a two-pass algorithm, driven by a page table list. The use bits are copied from the page table word into the use switch in the group map on the first pass. Effectively the logical OR of the use bits for all pages

constituting a hyperpage is used. The eligible lists are then reordered as follows on the second pass.

The chain of pointers in the group map for a given eligible list is traced in order, from beginning to end. At the end of this (the second pass in the ranking), the list has been reordered as follows. The resulting list may be thought of as two lists, the first with use bit 0 groups, the second with use bit 1 groups, and with the end of the 0 list pointing to the beginning of the 1 list. Within each list, the order in the original list is preserved. During the course of the ranking, external pointers to the list entries keep track of the last group with use bit 0, the last one with use bit 1, and the group currently being considered. These pointers are maintained even when groups to which they point are removed from their places in the list due to other reasons (e.g., unassignment or change of status). The implementation is such that the list is in a self-consistent state whenever the process can lose control. Certain critical faults and interrupts are threfore masked during the short sequence of instructions for which self-consistency is not present. If an interrupt is taken which results in the loss of control for the process on whose behalf the ranker is running, no harm is done. In fact, in such cases it is not necessary that the reordering be completed, since the most likely candidates for removal may already have been considered.

In order to prevent a single list from being simultaneously ranked by more than one process, there is a snag word for each list similar to the interlock word for the core map. A process competes for the snag in order to have the right to rank the corresponding list. The snag is given a name distinct from "interlock" to differentiate it from an interlock, since entries in the list may still be available while the list is snagged. However, an individual entry is nevertheless locked when a change is made to it or to an entry to which it points.

Implementation

The specific implementation of the ranking algorithm is as follows. The effective use bit of a group is the inclusive OR of the use bits of all pages contained in that group. In the case of a group which contains a page of a segment, this is simply the use bit. Three indices, or external pointers, are defined as follows.

N0 is equal to the pointer contained in the last entry processed with use bit 0 (i.e. in principle, the index of the first entry processed with use bit 1).

N1 is equal to the pointer contained in the last entry processed with use bit 1 (i.e. the index of the entry currently being processed).

N is the pointer contained in the entry currently being processed (i.e. the index of the next entry to be processed).

In other words, all of these indices point to the position of the next entry with use bit 0, 1, or undetermined, respectively.

The following notation is introduced. If i is the index of an entry, that entry is denoted by (i). Thus the index N points to the entry (N) currently being processed.

For all but a possible few instructions in the examination of each entry, the list remains consistently and correctly linked. The entries under consideration may therefore be used by other processes, subject to the lock switch in the group map entry.

The general strategy is as follows. The entry (N1) is examined. If its use bit is 1, it is already in its proper place in the new ranking. If its use bit is 0, the entry must be removed from its present place in the list, and reinserted by swapping places with (N0). An entry remains locked from the time of its processing until it ceases to be pointed to by either N0 or N1.

Specifically, if i is the index of the entry (i), then let $\underline{i}$ denote the index of the entry ($\underline{i}$) pointed to by the pointer residing in the entry (i). That is, the entry (i) points to the entry ($\underline{i}$). Then the processing of the entry (N) in the list takes the following form.

```
N = N1                  /*get next entry in list*/
lock the entry (N)
if usebit of (N1) = 1 then
     unlock (N1)
     N1 = N             /*move up N1*/
if usebit of (N1) = 0 then
     swap (N1) and (N)
     N1 = N             /*remove (N) from the list*/
     swap (N0) and (N)
     N = N0
     N0 = N             /*add (N) after (N0)*/
     unlock (N0)
     N0 = N             /*move up N0*/
```

This sequence is then repeated for the next (N) in the list. Its veracitudinousness is left as an interesting exercise for the reader.

One-pass ranker

Because of the problems in implementing the two-pass ranker, which requires from one to three groups to be locked at all times during the second pass, a simplified version has been written for use until it appears that more sophistication is needed.

This interim ranker is a one-pass ranker, also driven by the page table list. It is essentially the first pass of the A. Jay Goldstein ranker with the exception that instead of just copying a use bit of "1", the corresponding hyperpage is removed from its current place in the list and reinserted at the end of the list.

The disadvantage of this algorithm, which is simplicity itself in theory and implementation, is the loss of all history for a given hyperpage prior to its last usage. Thus, although all hyperpages used within a given rank cycle are farther back on the list than any hyperpages unused since before that cycle, there is no ordering among hyperpages last used in that cycle. This implies a rather arbitrary order (based on the page table list) at the end of the hyperpage list, where all hyperpages have been used during the last rank cycle. However, at the beginning of the list, where pages have not been used for many cycles, the order closely approximates that given by the two-pass algorithm. Of course, the order is most important at the beginning of the list, whence pages are removed. This interim ranker may well be satisfactory for a long time.

A note on locking

With either technique of locking (individual group map entry locks controlled by a global core map lock, or just the global lock), and especially during the second pass of the two-pass ranker, there is the problem that the ranker spends far more time in core control than any other process, and may effectively lock all other processes out of the core map. To avoid this, the ranker must be coded to periodically unlock the entire core map and allow itself to be interrupted even although its task is not yet completed. During the one-pass ranker, this can conveniently take place "in between page tables", so to speak, with no particular problems. However, during the second pass of the A. Jay Goldstein ranker, there must be special pointers kept into the hyperpage list to "keep the place", specifcally by remembering the positions of N0 and N1. These pointers must follow the entry they refer to in case its contents are moved physically (as in the swapping of group map entries), or must be reset to the following entry in the list in case of removal of their entry. This complication is the main reason for the existence of the interim ranker.