Lekha V. Bhandari, Mahip M. Bartere, Sneha U. Bohra / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue4, July-August 2012, pp.1996-1998

Demand Partitioned Virtual Memory

Lekha V. Bhandari*, Mahip M. Bartere, Sneha U. Bohra*****

*(M.E (Scholar) G.H.Raisoni College of Engineering and Management, Amravati.
Department of Computer Science & Engineering)
** (M.E (CSE) G.H.Raisoni College of Engineering and Management, Amravati.
Department Computer Science & Engineering)
*** (M.E (Scholar) G.H.Raisoni College of Engineering and Management, Amravati.
Department Computer Science & Engineering)

ABSTRACT

Partitioning a memory into many blocks and catching a certain amount of data in main memory from disk where blocks allocated for process are sufficient to execute process is very useful mechanism provide multiprogramming and cpu utilization. However creating appropriate allocation and replacement algorithm are daunting tasks. This paper proposed an on demand memory partition mechanism for scalable and efficient memory management .This paper propose that process start with different size. The allocation algorithm then gradually divides a process into many blocks as main memory receives more requests from process from memory. It catches them in new blocks from available once (free blocks). Consequently: memory partition done on demand where user's task determines the appropriate size of memory.

I. INTRODUCTION

Most operating system such as OS/360 running on IBM hardware used the fixed partition memory management method .In this scheme main memory was divided into various sections called partitions. This partition could be of different sizes, but once decided at time of system generation, they could not be changed. To change partitions, the operations have to stop and operating system has to be load and create different partition specification.
In short, evolutions of this scheme is wastage of memory, access time is not very high. Time complexity is very low becauseallocation / deallocation routines are simple, as partition is fixed [Silberschatz Abraham et. al.].

New embedded devices are heavily relay on demand memory due to unpredictability of the input data on number of application running concurrently defined by user. In relation with this context, required such design methodologies that can handle preciously the complex demand memory behavior. Demand Partitioned Virtual

Memory is solution on this problem and it can implement with demand segmentation or demand paging with linked list allocation. This paper proposed use of link list for allocation so to link between two differently located block, to read a process. It also minimizes wastage of memory due to fragmentation.

II. RELATED WORK

Red Brick Warehouse starts to allocate some of memory to a server, allocated additional memory on demand. The actual varies by server and depends on the row length of its intermediate result set. Unless a server demands more memory, it never exceeds its initial size. When a server does demand more memory than allowed, it spills blocks of intermediate result set to disk.

Davide Atienza et. al. had worked on new portable consumer embedded devices that execute multimedia and wireless applications that demand extensive memory. New portable device heavily rely on Demand Memory (DM) due to unpredictability of the input data and system behavior. They proposed new methodology that allows designing custom DM management mechanism with a reduced memory for such kind of dynamic applications. The experimental results show 60% of effective memory utilization.

III. DEMAND PARTITION

This paper proposes to use linked list allocation with new free space management police for demand partition. It also proposed to use the Indexed allocation to support direct accessing.
Benefits of demand partitioned virtual memory are
i) Minimizing memory Wastage
ii) Sharing and control
iii) All processes get address space
iv) Monitoring RAM and VM
v) Efficient free space management

IV. CACHE REPLACEMENT POLIC

This paper proposed to use genetic algorithm based page replacement policy. Structure of GA-based page is shown in figure 1. Several attributes of a cache page are selected to include in the structure. These strings provide direct and indirect measurement that can be used for fitness value calculation.

Hit Count	Access Count	Last Access Time	Time in Cache

Figure 1: Structure of Page

The Hit Count is the number of times a cache page is referenced without being loaded from secondary memory, i.e. no page fault. Hit count and page fault are antagonistic attributes, i.e. a higher hit count means less page faults. It is desirable to have a page replacement policy with as lesser page faults, or high hit count.

The Access Count is the number of time a cache page is referenced regardless to page fault. A high count means the page is referenced often and it should stay in cache so that the next reference to this page can be a page hit.

The Last Access Time is the time when the page is last referenced. It is a timestamp value in millisecond since Epoch time. The larger the value, the more recent the timestamp is. LRU policy uses this value to choose a page that has the smallest value for replacement.

The Time in Cache is the duration that the page has been loaded into cache. A high value can be interpreted either good or bad depending on the page replacement policy is used: A page with high Time in Cache value in FIFO algorithm means the page should be replaced, but can be kept in LRU algorithm if the Last Access Time is also high. This exercise interprets this value using the latter idea because of the combination with hit count.

These attributes are real-coded because of speed consideration. Time values tend to be large numbers because they are expressed in milliseconds. Coding them in real-values avoids extraneous mathematical conversion which takes up computation cycles. In a real OS, program execution halts at page fault to load the new page. The first step to load a new page is to select a page for removal. Therefore, selecting a page speedily is not only desirable, but necessary for quick resumption of program execution.

V. FITNESS FUNCTION

With the above attributes identified as strings to form the structure, we can define the fitness function. Our fitness function is a summation of the above attributes with minor tweaks because they are unsuitable to be used directly to calculate the fitness value. For reasons that will be explained in the next section, only selection and mutation operators are used. No crossover operation is performed. Since the actual values of hit count and access count are used for statistics, and hence evaluation, they cannot be mutated (modified) directly. To mutate these values without affecting the statistics, new counters, Hit Count_{GA} and Access Count_{GA}, only used by the fitness function are introduced for hit count and access count respectively. These new counters will be incremented at the same time when the real hit count and real access count are incremented. They will also be modified during mutation.

Besides the real counters, the time values are also unsuitable to apply directly to the fitness function because of their large number relative to the hit count and access count. Doing so will make the contribution from the counters insignificant. To alleviate this problem, two steps were taken. First, only the time difference from the cache load is used instead of the timestamp. Second, the time values are calculated in seconds as opposed to milliseconds.

Thus, we have the fitness function in equation (1):

$$\text{Fitness} = \text{Hit Count}_{GA} + \text{Access Count}_{GA} + \text{Age at Last Access} + \text{Time in Cache} \quad (1)$$

$$\text{Age at Last Access} = \text{Timestamp at Last Access} - \text{Timestamp at Page Allocation} \quad (2)$$

$$\text{Time in Cache} = \text{Current timestamp} - \text{Timestamp at Page Load} \quad (3)$$

Equation (2) defines Age at Last Access as the difference between the timestamp value at last page access and the timestamp value when the page is allocated. A page is allocated when it is instantiated in the secondary memory, regardless whether it is loaded into the cache.

Equation (3) defines Time in Cache as the difference between the current timestamp and the timestamp when the page is loaded in cache. Note that if a page is swapped out of the cache and loaded back in, the timestamp at page load will be reset to the new timestamp value when the page is loaded back in.A page candidate with a higher fitness value in equation (1) means that it is more suitable to stay in cache. The logic goes like this. A page with high hit count is a direct proof that residence of this page in cache is beneficial for the program execution. A page with high access count means that this page is referenced frequently and hence should stay in cache. A page with older age at last access means that the page usage is spread throughout the program and hence should stay because it is more likely to be used again. A page with longer time in cache means that the page has spent more time in cache, possibly has survived more page selection, hence should stay in cache.

VI. CONCLUSION

Dynamic memory allocation using linked list allocation for sequential access performs well in comparison of fixed static allocation where memory wastage is more (in dynamic memory allocation

memory used by pointer is just 0.78%of its total size.).When it used with efficient page replacement policy it gives best results.

GA-based page replacement policy performs almost as well as the LRU policy under various page sizes and cache sizes. GA has shown in many areas to produce human-competitive results [3]. It is a very promising result to see that a simple GA-based cache replacement policy model can perform so closely to other well-studied computer algorithms. With a more well-tuned fitness function, perhaps GA can perform even better than LRU algorithm.

REFERENCES

[1] Silberschatz Abraham and Gane Greg,Galvin Petter Baer, "operating system concept", Published Addison-Wesley Reading, 7 Edition, May 2006.

[2] Davide Atienza and S. Mamagkakis, F. Catthoor, J.M. Mendias,"Dynamic Memory Management Design Methodology for Reduced Memory Footprint in Multimedia and Wireless Network Applications", Publisher IEEE Computer Society Press, Washington DC, USA.

[3] D. E. Goldberg, "Real-coded Genetic Algorithms, Virtual Alphabets, and Blocking" Complex Systems, May 1991 139-167.

[4] Red Brick Warehouse available at http://www.ibm.com/software/data/informix/redbrick/

[5] http://www.ehow.com

[6] http://www.drdobbs.com/article