2012 International Conference on Information and Computer Networks (ICICN 2012)
IPCSIT vol. 27 (2012) © (2012) IACSIT Press, Singapore

A NEW APPROACH TO MINIMIZE PAGE FAULT

Vijay Srivastva [1], Gurmit Kaur Chhabra [2] and Suvrojit Das [3]

[1,2,3] National Institute Of Technology, Durgapur, India

Abstract. Memory management is important aspect of an operating system. We always try to make an efficient use of memory and CPU ,so that more and more process can execute simultaneously in the system. This is also known as multiprogramming. This can be done by loading many the programs at the same time in the memory, or just loading only parts of programs in the memory which is necessary to execute. This is done by dividing the program into pages and memory in frames and loading the page of programs that is necessary to execute at that time. This technique is known as dynamic paging. In this technique when next page is required by the CPU and that page is not in the memory then a page fault will occur. Then operating system fetches that page from the secondary memory and give it to the CPU. There may be situation when a page fault will occur but there is no frames available and so, for that a page replacement (load the fetched page in some frame which is occupied by some other page on the basis of some algorithm) is required. In the worst scenario every time a page is fetched from the secondary memory there will be a page fault again due to unavailability of frames and the system is busy in doing swapping. The CPU remains idle and this makes an inefficient use of CPU. So there is a need to avoid this situation to occur. In this paper we try to minimize page fault occurrence, so that there is no need to wait for the particular number of page fault to occur and then taking action based on some criteria. In this paper we are emphasizing on the number of page fault in a particular time and this is done by just checking periodically, the page fault after a fixed interval of time for every process. By this technique page fault can be minimized and it also prevent the occurrence of thrashing .

Keywords: Multiprogramming, dynamic paging, thrashing, pages and frames, page fault.

1. Introduction

To achieve multiprogramming, dynamic paging is required and in dynamic paging, occurrence of page fault is mandatory as we don't want to load the whole program in the memory, but the part of the the program that is needed for the execution at that instant and when another page required by the CPU to execute the next instruction of a program and that page is not in the memory then a page fault will occur.

There are many other techniques which allow multiprogramming :

- Contiguous memory allocation
- Dynamic storage allocation

Contiguous memory allocation:

In this technique each process is contained in a single contiguous section of memory. In this technique page fault will not occur because whole program is in the memory. But it will lead to less number of processes running simultaneously. With this technique it also leads to both internal and external fragmentation.

External fragmentation : As processes are loaded and removed from the memory, the free memory space is broken into little pieces. External fragmentation exists when there is enough total memory space to satisfy a request , but the available spaces are not contiguous this load inefficient use of memory.

[+]Corresponding author. Tel.: + (91 9734294105); fax: +(91 343 2547375).
E-mail address: (suvrojit.das@gmail.com).

Dynamic storage allocation:

Which concerns how to satisfy a request of size n from a list of free sections of memory. There are many solutions to this problem as:

1. First fit
2. Worst fit and
3. Best fit

First fit: Allocated the first hole that is big enough. Searching can start either at the beginning of the set of free holes (memory) or where the previous first fir search ended. After finding a hole which is large enough for that process then stop searching.

Best fit: Allocate the smallest hole that is big enough. There is need to search entire list ,unless the list is ordered by size. In this technique internal fragmentation is less.

Worst fit: Allocate the largest hole .In this also there is need to search the entire list , unless it is sorted by size. In this technique internal fragmentation is very large.

External and internal fragmentation left the chunks of memory underutilized. So there is a technique called paging which removes the above mentioned problems by utilizing the memory resources adequately to increase the efficiency.

Paging is a memory-management scheme that permits the physical space of a process to be noncontiguous. The basic method for implementing the paging involves breaking physical memory into fixed size blocks called **frames** and breaking logical memory into blocks of the same size called **pages.** when a process is to be executed ,its pages are loaded into any available memory frames from the backing store. Still in this technique we have to load the whole program in the memory.

To tackle the problem of loading whole program in the memory load only the relevant pages which is required for the execution of the particular program. This approach is known as DEMAND paging.

In this technique when we want to execute the process, we swap it into the memory. Rather than swapping the entire process into the memory, however, we can use a lazy swapper(pager). A pager never swaps a page into the memory unless that page will been needed.

If the process tries to access a page that was not brought into the memory. Access to a page marked invalid cause a page fault trap.

The paging hardware, in translating the address through the page table , will notice that the invalid bit is set , causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into the memory. Page table is the table contains the base address of each page in the physical memory.

The procedure for handling the page fault is shown in the figure 1.

As:

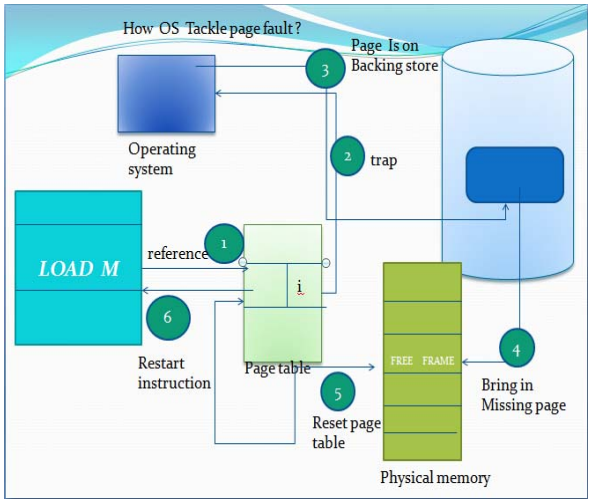

Fig. 1: How Operating System tackles Page Fault

[1] Check an internal table for this process to determine whether the reference was a valid or an invalid memory access.
[2] If the reference was invalid , just terminate the process. If it was valid , but we have not yet brought in that page, we now page it in.
[3] find the frame (by taking one from the free frame list).
[4] We schedule a disk operation to read the desired page in to the newly allocated frame.
[5] When the disk read is complete we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
[6] We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

In worst case if process does not have the number of frames it needs to support pages in active use,it will quickly page fault. At this point , it must replace some page. Since all its pages are in active use , it must replace a page that will be needed again right away. It quickly faults again and again and again replacing pages that it must bring back in immediately. this high paging activity is called **thrashing.**

2. Related Works & Motivation

Not much research could be found by surveying the related research done so far relating to minimization of page faults.

From [3, 2], we have analyzed, working set model and page fault frequency minimizing the page fault as follows:

The **working set model** states that a process can be in ram if and only if all of the pages that it is currently using can be in RAM. The model is an all or nothing model, meaning if the pages it needs to use increases, and there is no room in RAM, the process is swapped out of memory to free the memory for other processes to use.

Often a heavily loaded computer has so many processes queued up that, if all the processes were allowed to run for one scheduling time slice, they would refer to more pages than there is RAM, causing the computer to "thrash".

In other words, the working set strategy prevents thrashing while keeping the degree of multiprogramming as high as possible. Thus it optimizes CPU utilization and throughput.

Page fault Frequency It is based on the simple observation that, when the frequency of page faults for a process increases above some threshold, then more memory should be allocated to the process.

In both models page fault is minimized when thrashing occurs.

There may be a situation where the CPU may be idle from the moment of the first page fault occurrence (due to no space in the memory) to the moment when frequency of page fault increases above some threshold .This is a waste of CPU time.

In this paper we have given an algorithm which states that there is no need to wait for occurrence of thrashing. We are checking page fault at a regular interval of time so that CPU does not remain idle.

3. Proposed Work

To analyze the page fault rate in a fixed interval of time by identifying the process which might have maximum number of page fault and to minimize the page fault rate of the process, we take the following approach.

In this approach, we have used the following parameters:

max_page_fault, pid, P_FAULT, fixed_time, frame_list, P_ID,temp.

Where ,

max_page_fault : checks the maximum number of page fault from the linked list.

Pid: which have the process id of the process having maximum page fault.

P_FAULT: count the number of pagefault for a process in fixed interval of time.

fixed_time: it is the time after which the program p will run to check the process having maximum pagefault.

frame_list: this contains the list of free frames.

P_ID:it contains process id having page fault in fixed interval of time.

Temp: it is temporary variable for linked list.

This fixed interval of time can be calculated by analyzing the page fault in the system.

For that, it is required to maintain the record of number of page faults associated with each process for a fixed interval of time. For that we do check for the process if it is already present in the list so that we can increment the P_FAULT element of that node which denotes the number of page fault generated by that process. If that process is not found in that list then we shall create a new node And initialize it with the process id and number of page fault (say 1)and add to that list.

Now after a fixed_time, the program P will run which will check for pid of the process having max no of page fault from the linked list.

And then check the free frame list if it is available then nothing to do, if it is not available, it means that the page fault is occurring because of the unavailability of free frames.

Then we will take action based on some cost that is

1. How much that process is completed.

2. How many resources it have.

etc..

If it is very much costly(in terms of time and resources) to terminate or suspend this process then check for the process from the linked list and choose the best one to suspend and give the frame occupied by the suspended process to this process in order to minimize the page fault.

Then remove the linked list and terminate the program P.

Then again the previous process will be repeated (means again Operating system start adding the process when a page fault occur by the process till fixed interval of time).

The following figure fig 2. Show how all works:

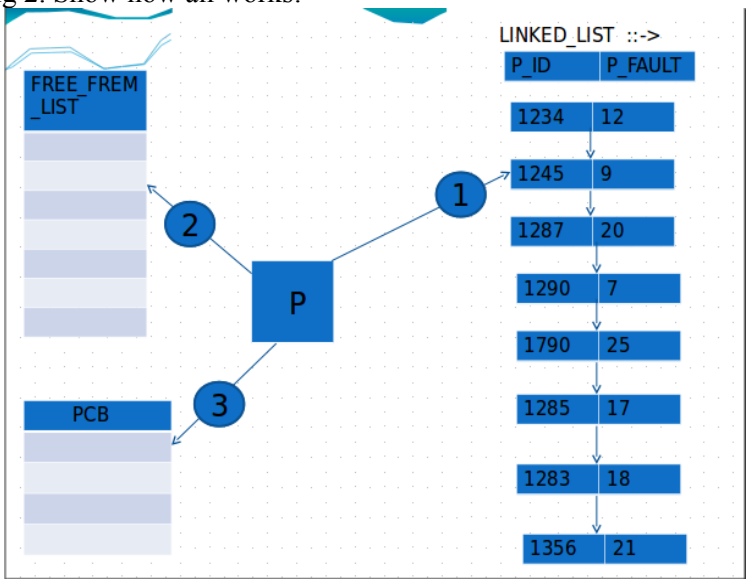

Fig 2. Our Algorithm

STEP **1**.

Program **p** is used to check the pid associated with the maximum number of page fault from the linked list maintend by operating system.Where linked list is used to store the process id and the associated page fault in a fixed interval of time.

Pseudocode for this program p is as follows:

Initialize max =-1;

```
            Pid=0;
While(temp!=NULL)
  {

     If(temp>P_FAULT>max_page_fault)
       {

          max_page_fault=temp>P_FAULT;
                pid=temp->P_ID;
          }
            temp=temp->next;
}
```

And the time complexity of this only o(n).And this program will run after every fixed interval of time.

STEP 2.

After finding the process having the maximum number of page fault this **FREE_FREM_LIST** will be checked.

STEP 3.

frame is not available then it check the **PCB** to decide whether to suspend the process or not.

4. Performance

The process requires only O(n) time.

After each page fault the process id of the process is added (if it is not present already) with the number of page fault associated with the process in a linked list. If it is already present, it increments the P_FAULT associated with that process.

It requires O(n) time for every page fault.

5. Conclusion And Future Work

n this paper we have derived an approach to minimize the page fault by checking the process's page fault by running a program p after every fixed interval of time. For that process having the maximum page fault number, first we check the free frame list for available free frames; if not available, then we check the PCB to find whether this process can be suspended or not based on some reasonable cost.

If we can not suspend that process due to higher cost we have to find another suitable process having low cost (from the linked list) that can be suspended. After suspending this selected process we allocate the freed frame to the process which had maximum page fault. So by executing this model of page fault handling we can decrease page fault number considerably. It may take some time of the CPU but CPU will never be idle because of thrashing. And it will increase the performance of the CPU even when no frame is available.

In future we need to find the fixed interval of time which will be the accurate time as a fixed time.

6. References

[1] Abraham silberschatz, Peter baer galvin,Greg gagne:Operating system principles.

[2] Chu, W.W.; Opderbeck, H.; Computer Volume: 9 Digital Object Identifier: 10.1109/C M.1976.218439 Publication Year: 1976 , Page(s): 29 - 38

[3] Gupta, R.K.; Franklin, M.A.; Computers, IEEE Transactions on Volume: C-27 Digital Object Identifier: 10.1109/TC.1978.1675177 Publication Year: 1978 , Page(s): 706 - 712