e-PG Pathshala

Subject: Computer Science

Paper: Operating Systems

Module 29: Allocation of Frames, Thrashing

Module No: CS/OS/29

Quadrant 1 — e-text

29.1 Introduction

In an earlier module, we learnt what demand paging is. In demand paging, pages are brought into the physical memory only when the pages are needed. All the pages of a process need not be kept in the main memory at the same time. Only the pages that are used currently need to be kept in the main memory. Suppose a page is brought into the main memory and if there is no free frame in the main memory, a victim page is chosen from the main memory, the victim page is moved out to the backing store and the required page is brought into the main memory. Choosing a victim page for replacement is done by the page replacement algorithms.

The other option is to maintain a pool of free frames always and move in the required page into a frame. The victim page can be moved out later and that frame can be added to the pool of free frames.

In this module, we learn how to share the frames of the physical memory among different processes, which is called allocation of frames. We also learn what thrashing is and the ways to overcome thrashing.

29.2 Allocation of frames

When many processes are kept in the physical memory, the way in which the physical frames are allocated to the different processes is called allocation of frames. For example, if there are 50 free frames and five processes, how many frames does each process get?

In pure demand paging, all the 50 physical frames are in the list of free frames. During execution, page faults occur and the necessary pages are brought in to the free frames. If there is no free frame, a page replacement algorithm is used. After termination of a process, the frames that the process used are added back to the free frame list.

There is another variant where free frames are always kept available. When there is a page fault, a victim page is selected. But before moving out the victim page, the required page is moved into one of the free frames. The victim page is later moved out and the corresponding frame where the victim page was kept is moved to the list of free frames.

There are few constraints while allocating frames. It is not possible to allocate frames more than the total number of available frames. It is also needed that each process must be allocated at least a minimum number of frames. This is because, when the number of frames allocated to a process decreases, page-fault rate increases.

29.2.1 Minimum Number of Frames

Each process should be allocated a minimum number of frames. But the minimum number depends on the instruction-set. It depends on the number of pages that a single instruction can reference. If there is a machine in which all the memory-reference instructions refer to only one memory address, then it is enough to allocate a minimum of two frames for the process. That is,

one frame for instruction and one frame for the memory reference. Suppose the instruction set supports one level of indirect level addressing, then a minimum of three frames must be allocated per process. If there are multiple levels of indirection, then there are more problems. Hence it is necessary to place a limit on the levels of indirection (usually, 16).

29.2.2 Allocation Algorithms

Equal allocation:

The simplest way to split the frames among the processes is to share the frames equally. For example, if 100 frames are to be shared among 5 processes, each process is given 20 (100/5) pages. If there are 103 frames and 5 processes, then each process may be given 20 pages and the remaining 3 frames may be added to the pool of free frames.

Proportional allocation:

But the issue here is that each process has a different size. When each process has a different size, it is not fair to allocate the same number of frames to all the processes. Therefore, it is advantageous to go for proportional allocation. In proportional allocation, the number of frames allocated to a process is based on the size of the process.

Let s_i be the size of process p_i. Let m be the total number of frames present in the physical memory. Therefore the sum of the sizes of all the processes $S = \Sigma s_i$. The allocation for process p_i is $a_i = s_i / S \times m$.

Consider an example where $m = 64$. Suppose there are two processes p_1, p_2. Let the size of process p_1 be $s_1 = 10$. Let the size of process p_2 be $s_2 = 127$.

The number of frames allocated to process p_1 is $a_1 = 10/137 \times 64 = 5$. The number of frames allocated to process p_2 is $a_2 = 127/137 \times 64 = 159$.

Thus, we see that process p_1 gets less number of frames because it has a smaller size. Process p_2 gets more number of frames because it is larger.

Priority Allocation:

In this method, frames are proportionally allocated using priorities rather than the size. It is also possible to allocate frames based on a combination of size and priority.

29.2.3 Global vs. Local Allocation

Another issue that occurs when multiple processes compete for frames is explained in this section. Suppose process P_i generates a page fault. Hence, the required page of process Pi has to be brought into the physical memory. Suppose there is no free frame and a victim frame has to be selected for page replacement. Now, there are two possibilities in selecting the victim frame. One possibility is that the victim frame selected can be one of the frames allocated to process P_i. This is called local replacement. The other possibility is that the page selected for replacement can be a frame from a process (which may or may not be P_i) with a lower priority number. This is called global replacement.

Global replacement:

In global replacement, since a process selects a replacement frame from the set of all frames, one process can take a frame from another. This allows a high priority process to select frames from a low priority process. This may increase the number of frames allocated to a high priority process. However, the low priority processes will still suffer from page faults, since the

high priority process takes frames from the low priority process. Therefore, the set of pages in memory for a process may depend on the paging behaviour of other processes.

Local replacement:

In local replacement, each process selects from only its own set of allocated frames. Therefore, the number of frames allocated to a process does not change. The set of pages in memory for a process is affected by the paging behaviour of only that process. But a process may not be able to use other less used pages of other processes.

Global replacement gives better throughput and hence, global replacement is commonly used.

29.3 Thrashing

If a process does not have enough frames, the page-fault rate is very high. If all allocated pages are active page replacement will lead to more page faults. Suppose a process is allocated 5 frames and all pages are active. If there is a page fault now for that process, one of the 5 active frames is moved out and the required page is brought in. Since the page that is moved out is an active page, the moved out page may be needed immediately. There is a page fault again and another active page is moved out to bring in the required page. This leads to page faults again and again.

When the number of page faults becomes more, the system spends most of the time in paging activity (moving in and moving out pages). This results in low CPU utilization. Since the CPU utilization is low, the operating system thinks that there are not enough processes to use the CPU. Therefore, the operating system increases the degree of multiprogramming. That is, it adds another process to the system. This increases the paging activity even further. This results in thrashing, where a process spends more time on paging than executing.

Suppose global page replacement is used. If the currently executing process needs more frames, it takes frames currently allocated to other processes. This makes the other processes also to fault, taking frames from other processes. This activity continues and leads to a high paging activity. This decreases the CPU utilization further. The degree of multiprogramming is further increased and the CPU utilization is further reduced.

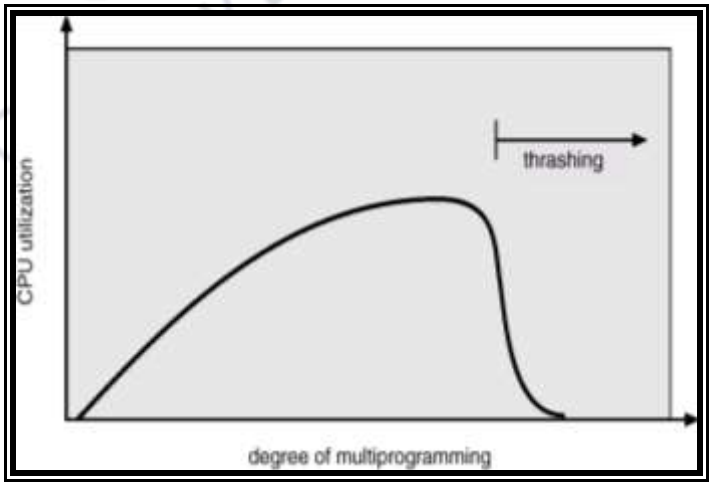

Fig. 29.1 Thrashing (Source: [1])

Figure 29.1 shows that the CPU utilization increases with the increase in the degree of multiprogramming to a certain extent. If the degree of multiprogramming is increased still

further, it results in the decrease of CPU utilization. This worsens the thrashing.

The effects of thrashing can be minimized by using a local replacement strategy. It is also necessary that the process is provided with as many frames as it needs at a particular time. That is, the process must be given some minimum number of frames so that page faults seldom occur and a local replacement strategy must be used.

Locality model of process execution:

The minimum number of frames to be allocated for a process may be decided based on the number of pages in the locality of the process at a particular time. The locality of a process refers to the set of pages that are used together by a process at a particular time. Any program is composed of several localities. During execution, a process migrates from one locality to another. For example, suppose a process is executing a subroutine. During that time, the process executes only the pages in that subroutine. The pages of that subroutine form the locality of the process during that time. It is possible for localities to overlap. If the number of frames allocated to a process becomes equal to the number of pages in the locality at that time, no page fault occurs. Only after the process moves to another locality, page faults may occur. Thus, if the minimum number of frames allocated to each process is equal to the number of pages in its locality, there is no page fault.

When the sum of the localities of all the processes becomes more than the total memory size, then thrashing occurs. We now see two methods used to avoid thrashing – the working set model and the page-fault frequency scheme.

29.3.1 Working-Set Model

The working-set model is based on the locality of the process. A working set is defined which is an approximation of the program's locality. Let $\Delta \equiv$ working-set window be a fixed number of page references. The working set refers to the set of pages in the most recent Δ page references. If a page of a process is active, then that page is present in the working set of that process. If a page is not used for some time, then that page will be dropped from the working set.

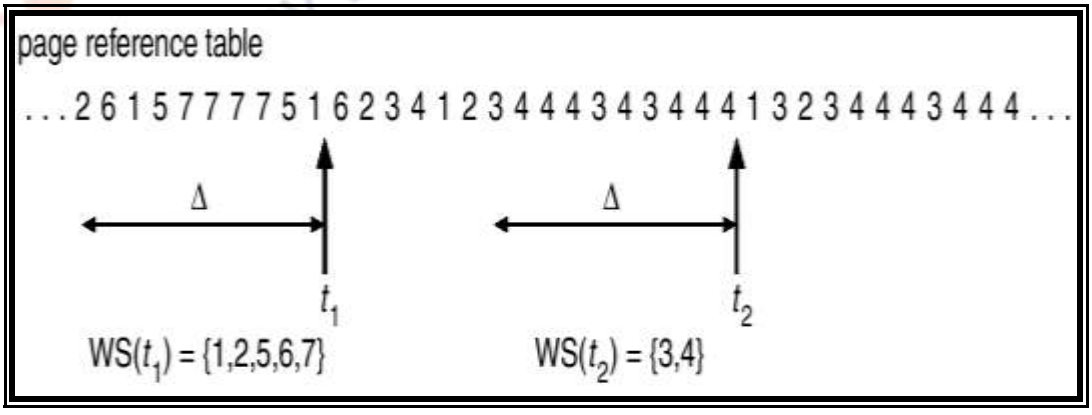

Fig. 29.2 Example of working-set model (Source: [1])

The working set size of a process is the total number of pages referenced in the process in the most recent Δ references. Consider the example shown in Figure 29.2. Let Δ = 10 memory references. At time t_1, the working set comprises of the (unique) pages 1,2,5,6 and 7. The working set size at t_1 is 5. At time t_2, the working set has changed and it comprises of the

pages 3 and 4. The working set size at t_2 is 2.

The accuracy of the working set depends on the selection of Δ. If Δ is too small, the working set will not encompass the entire locality. If Δ is too large, the working set will encompass several localities. If $\Delta = \infty$, the working set will encompass the entire program.

Let WSS_i denote the working set size of process P_i. Let $D = \Sigma\ WSS_i$ be the total demand of all the processes. If $D > m$, that is, if the sum of the working set sizes of all the processes becomes greater than the total number of frames, thrashing occurs. Therefore, if $D >$ m, then one of the processes must be suspended. If $D < m$, that is, if the total demand of all the processes becomes less than the total number of frames available, then another process is initiated. Thus thrashing can be avoided. This method also helps in keeping the degree of multiprogramming as high as possible.

Keeping track of the working set is difficult in this method.

Keeping Track of the Working Set:

The working set can be approximated using a fixed interval timer and a reference bit. For each page, additional 2 bits are kept in memory. For example, let the working set size Δ = 10,000 references. Let the timer interrupt after every 5000 time units. Whenever the timer interrupts, the reference bit is copied to the additional bit maintained in memory and the values of all the reference bits are set to 0. When there is a page fault, we can examine the current reference bit and the two in-memory bits to determine whether a page was used within the last 10,000 to 15,000 references. If it was used, at least 1 of these bits will be on. If it has not been used, these bits will be off. Those pages with at least 1 bit on will be considered to be in the working set. But, this may not be entirely accurate, because, we really don't know how many times a page is referenced within the 5000 time units. To improve the accuracy, we may increase the number of in-memory bits and increase the frequency of checking the reference bit (say, 10 in-memory bits and interrupt every 1000 time units).

29.3.2 Page-Fault Frequency Scheme:

The second method to avoid thrashing is the page-fault frequency scheme. Thrashing has a high a page-fault rate. Therefore, if thrashing can be controlled, page-fault rate can be controlled.

If the page-fault rate is too low, then it means that the process has more frames than needed and hence, it must lose frames. If the page-fault rate too high, it means that the process has fewer frames than needed and the process must gain frames.

Figure 29.3 shows how the page-fault rate varies with the increase in the number of frames. An upper bound and a lower bound are maintained. If the page-fault rate goes beyond the upper bound, the number of frames allocated to each process is increased. If the page-fault rate goes below the lower bound, the number of frames allocated to each process is decreased.

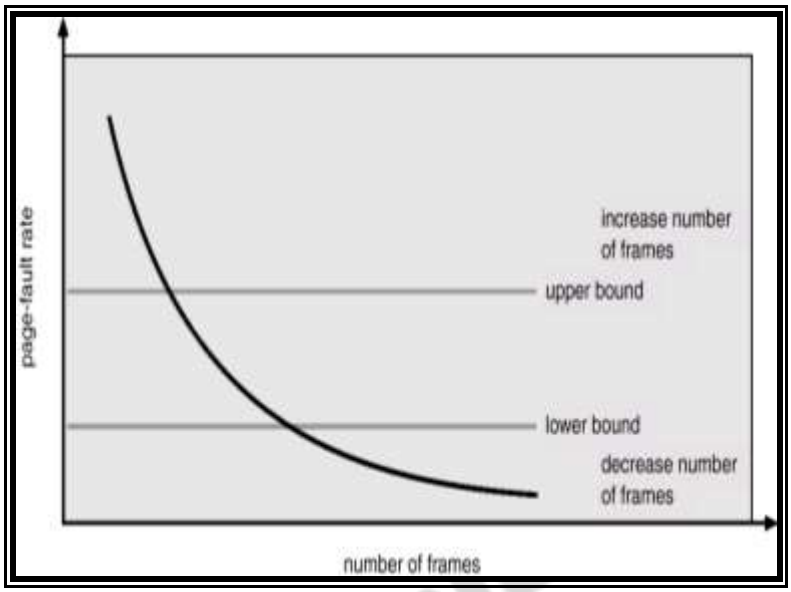

Fig. 29.3 Example of working-set model (Source: [1])

29.4 Summary

In this module, we learnt how frames can be allocated for processes. We learnt the difference between global and local replacement. We learnt what is meant by thrashing and the methods (the working-set model and the page fault frequency scheme) used to avoid thrashing.

References

1. Abraham Silberschatz, Peter B. Galvin, Greg Gagne, “Operating System Concepts”, Sixth Edition, John Wiley & Sons Inc., 2003.
2. Andrew S. Tanenbaum, Herbert Bos, “Modern Operating Systems”, Fourth Edition, Pearson Education, 2014.
3. Gary Nutt, “Operating Systems”, Third Edition, Pearson Education, 2009.