Monograph on Thrashing

By Ms. Swati Jain

Introduction

In computer science, **thrashing** occurs when a computer's virtual memory resources are overused, leading to a constant state of paging and page faults, inhibiting most application-level processing. This causes the performance of the computer to degrade or collapse. The situation can continue indefinitely until either the user closes some running applications or the active processes free up additional virtual memory resources.

The term is also used for various similar phenomena, particularly movement between other levels of the memory hierarchy, where a process progresses slowly because significant time is being spent acquiring resources.

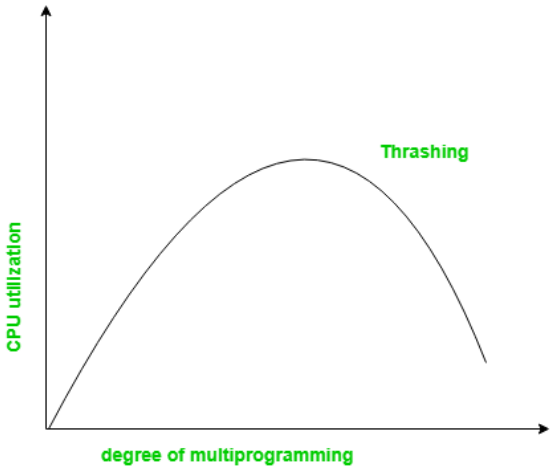
Virtual memory works by treating a portion of secondary storage such as a computer hard disk as an additional layer of the cache hierarchy. Virtual memory is notable for allowing processes to use more memory than is physically present in main memory and for enabling virtual machines. Operating systems supporting virtual memory assign processes a virtual address space and each process refers to addresses in its execution context by a so-called virtual address. In order to access data such as code or variables at that address, the process must translate the address to a physical address in a process known as virtual address translation. In effect, physical main memory becomes a cache for virtual memory which is in general stored on disk in memory pages.

Programs are allocated a certain number of pages as needed by the operating system. Active memory pages exist in both RAM and on disk. Inactive pages are removed from the cache and written to disk when the main memory becomes full.

If processes are utilizing all main memory and need additional memory pages, a cascade of severe cache misses known as page faults will occur, often leading to a noticeable lag in operating system responsiveness. This process together with the futile, repetitive page swapping that occurs is known as "thrashing". This frequently leads to high, runaway CPU utilization that can grind the system to a halt. In modern computers, thrashing may occur in the paging system (if there is not sufficient physical memory or the disk access time is overly long), or in the I/O communications subsystem (especially in conflicts over internal bus access), etc.

Depending on the configuration and algorithms involved, the throughput and latency of a system may degrade by multiple orders of magnitude. Thrashing is a state in which the CPU performs 'productive' work less, and 'swapping' more. The overall memory access time may increase since the higher level memory is only as fast as the next lower level in the memory hierarchy. The CPU is busy in swapping pages so much that it cannot respond to users' programs and interrupts as much as required. Thrashing occurs when there are too many pages in memory, and each page refers to another page. The real memory shortens in capacity to have all the pages in it, so it uses 'virtual memory'. When each page in execution demands that page that is not currently in real memory (RAM) it places some pages on virtual memory and adjusts the required page on RAM. If the CPU is too busy in doing this task, thrashing occurs.

Thrashing is a condition or a situation when the system is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults. As a result, no useful work would be done by the CPU and the CPU utilisation would fall drastically. The long-term scheduler would then try to improve the CPU utilisation by loading some more processes into the memory thereby increasing the degree of multiprogramming. This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming, called Thrashing.

Effect of Thrashing

Whenever thrashing starts, operating system tries to apply either Global page replacement Algorithm or Local page replacement algorithm.

Global Page Replacement

Since global page replacement can access to bring any page, it tries to bring more pages whenever thrashing found. But what actually will happen is, due to this, no process gets enough frames and by result thrashing will be increase more and more. So, global page replacement algorithm is not suitable when thrashing happens.

Local Page Replacement

Unlike global page replacement algorithm, local page replacement will select pages which only belong to that process. So there is a chance to reduce the thrashing. But it is proven that there are many disadvantages if we use local page replacement. So, local page replacement is just alternative than global page replacement in thrashing scenario.

Locality Model

A locality is a set of pages that are actively used together. The locality model states that as a process executes, it moves from one locality to another. A program is generally composed of several different localities which may overlap.

For example when a function is called, it defines a new locality where memory references are made to the instructions of the function call, it’s local and global variables, etc. Similarly, when the function is exited, the process leaves this locality.

Techniques to handle:

1. Working Set Model

This model is based on the above-stated concept of the Locality Model. The basic principle states that if we allocate enough frames to a process to accommodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of the current locality, the process is bound to thrash.

According to this model, based on a parameter A, the working set is defined as the set of pages in the most recent 'A' page references. Hence, all the actively used pages would always end up being a part of the working set.

The accuracy of the working set is dependent on the value of parameter A. If A is too large, then working sets may overlap. On the other hand, for smaller values of A, the locality might not be covered entirely.

If D is the total demand for frames and WSS_i is the working set size for a process i,

$D= \sum WSS_i$

Now, if 'm' is the number of frames available in the memory, there are 2 possibilities:

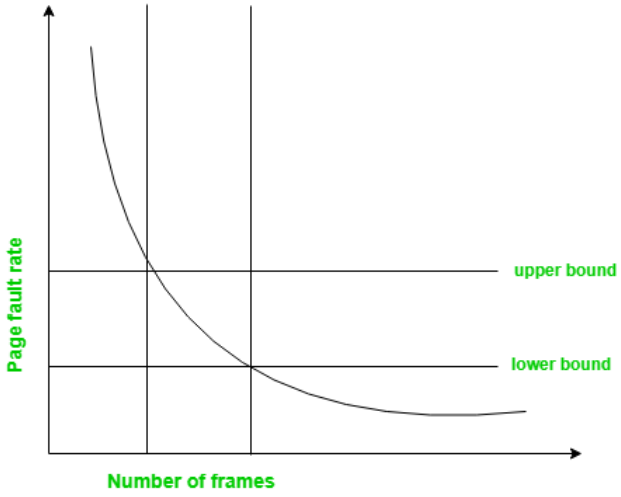
- D>m i.e. total demand exceeds the number of frames, then thrashing will occur as some processes would not get enough frames.
- D<=m, then there would be no thrashing.

If there are enough extra frames, then some more processes can be loaded in the memory. On the other hand, if the summation of working set sizes exceeds the availability of frames, then some of the processes have to be suspended (swapped out of memory).

This technique prevents thrashing along with ensuring the highest degree of multiprogramming possible. Thus, it optimizes CPU utilization.

2. Page Fault Frequency

A more direct approach to handle thrashing is the one that uses Page-Fault Frequency concept.



The problem associated with Thrashing is the high page fault rate and thus, the concept here is to control the page fault rate.

If the page fault rate is too high, it indicates that the process has too few frames allocated to it. On the contrary, a low page fault rate indicates that the process has too many frames.

Upper and lower limits can be established on the desired page fault rate as shown in the diagram.
If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page fault rate exceeds the upper limit, more number of frames can be allocated to the process.

In other words, the graphical state of the system should be kept limited to the rectangular region formed in the given diagram.

Here too, if the page fault rate is high with no free frames, then some of the processes can be suspended and frames allocated to them can be reallocated to other processes. The suspended processes can then be restarted later.

Causes

Thrashing affects the performance of execution. Initially, when the CPU utilization is low, then process scheduling mechanism, loads many processes into the memory at the same time so that degree of multi programming can be increased.

So now in this situation we have more number of processes in memory as compare to the available number of frames in memory. Allocation of limited amount of frames to each process.

When any higher priority process arrive in memory and if frame is not freely available at that time then the other process that occupied the frame which is resides in frame will move to secondary storage and this free frame is now allocated to higher priority process .

In other words we can say that as the memory fills up, process starts to spend a lot of time for the required pages to be swapped in, again CPU utilization becomes low because most of the processes are waiting for pages.

In virtual memory systems, thrashing may be caused by programs or workloads that present insufficient locality of reference: if the working set of a program or a workload cannot be effectively held within physical memory, then constant data swapping, *i.e.*, thrashing, may occur. The term was first used during the tape operating system days to describe the sound the tapes made when data was being rapidly written to and read. A worst-case scenario of this sort on the IBM System/370 series mainframe computer could be an execute instruction crossing a page boundary that points to a move instruction itself also crossing a page boundary, itself pointing to a source and a target that each cross page boundaries. The total number of pages thus involved in this particular instruction is eight, and all eight pages must be simultaneously present in memory. If any one of the eight pages can't be swapped in (for example to make room for any of the other pages), the instruction will fault, and every attempt to restart it will fail until all eight pages can be swapped in.

Other Uses

Thrashing is best known in the context of memory and storage, but analogous phenomena occur for other resources, including:

Cache thrashing

Where main memory is accessed in a pattern that leads to multiple main memory locations competing for the same cache lines, resulting in excessive cache misses. This is most problematic for caches that have low associativity.

TLB thrashing

Where the translation lookaside buffer (TLB) acting as a cache for the memory management unit (MMU) which translates virtual addresses to physical addresses is too small for the working set of pages. TLB thrashing can occur even if instruction cache or data cache thrashing are not occurring, because these are cached in different sizes. Instructions and data are cached in small blocks (cache lines), not entire pages, but address lookup is done at the page level. Thus even if the code and data working sets fit into cache, if the working sets are fragmented across many pages, the virtual address working set may not fit into TLB, causing TLB thrashing.

Heap thrashing

Frequent garbage collection, due to failure to allocate memory for an object, due to insufficient free memory or insufficient contiguous free memory due to memory fragmentation is referred to as heap thrashing.

Process thrashing

A similar phenomenon occurs for processes: when the process working set cannot be co-scheduled – so not all interacting processes are scheduled to run at the same time – they experience "process thrashing" due to being repeatedly scheduled and unscheduled, progressing only slowly.