CS 3003D – OPERATING SYSTEMS

ASSIGNMENT – 4 LECTURE NOTES

GROUP - 15

THRASHING - WORKING SET MODEL - PAGE FAULT FREQUENCY

SUBMITTED BY

YACHA VENKATA RAKESH

B180427CS

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY CALICUT

TABLE OF CONTENTS

1. Background Theory – Virtual Memory:

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 best useful 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 memory which is in general stored on disk in memory pages. Programs are allocated a certain number of pages as needed by the operating systems. 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 sever 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 are known as thrashing. This frequently leads to high, runaway CPU utilization that can grind the system to halt. In modern computers, thrashing may occur in paging system or in the I/O communications subsystem etc.,

2. Introduction:

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. Thrashing is also used in contexts other than virtual memory systems; for example, to describe cache issues in computing or silly window syndrome in networking. Performance of paged memory systems has not always met expectations. Consequently, there are some who would have us dispense entirely with paging, believing that programs do not generally display behaviour favourable to operation in paged memories. We shall show that troubles with paged memory systems arise not from any misconception about program behaviour, but rather from a lack of understanding of a three-way relationship among program behaviour, paging algorithms, and the system hardware configuration. Paging's poor performance is not unfavourable program behaviour but rather the large time required to access a page stored in auxiliary memory, together with sometimes stubborn determination on the part of the system designers to simulate large virtual memories by paging small real memories. Thrashing occurs on a program that works with huge data structures, as its large working set causes continual page faults that drastically slow down the system. Satisfying page faults may require freeing pages that will soon have to be re-read from disk.

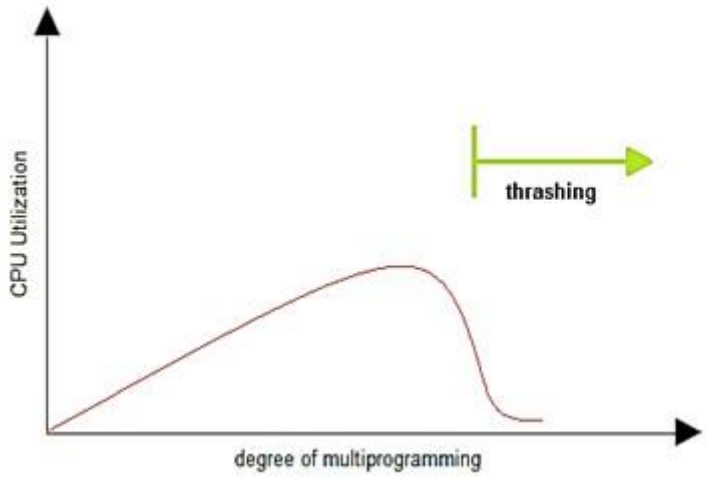
3. Thrashing:

Thrashing in short refers to a process that is spending more of its time on paging than the original execution time. If in case the number of frames allocated to a low-priority process reduces below the minimum number required by the computer architecture, we must suspend that process's execution, keep it in wait state. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling. Sometimes, the pages which will be required in the near future have to be swapped out. This would lead to an extra work of swapping in these pages in the near future. In fact, look at any process that doesn't have enough frames. If the process doesn't have the number of frames it needs to support pages in active use, it will quickly throw a page-fault. At this point, it must be replaced to some page at memory. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, replacing pages that it must bring back in immediately. This high paging activity is called *thrashing*. In other words, A process is thrashing if it is spending more time paging than executing and causing excessive overhead and severe performance degradation or collapse caused by too much paging. It is referred to a high paging activity. It was a serious problem in early demand paging systems reducing performance of paged memory systems reducing computing giants to computing dwarfs.

Thrashing mainly occurs when a system's virtual memory resources are overused, leading to a constant state of paging and page faults, suspending most application-level processing. This causes performance of the computer to degrade or collapse. This situation might continue infinitely until either the user closes the running applications or the currently active processes free up additional virtual memory resources. We know increasing the degree of multiprogramming, CPU utilization increases. But this occurs only up to a certain limit after which it suddenly starts decreasing drastically. This point of local maxima is referred to as thrashing.

Disadvantages when memory gets overcommitted:

- Suppose the pages being actively used by the current threads can't completely fit in physical memory.
- Each page fault causes one of the active pages to be moved to disk and the required page is replaced in its place. Now there are chances that another page fault occurs to get back this replaced place sooner and hence it's unnecessarily performing double work which will lead to decline in the application-level process's performance.
- The system will have to spend all its time reading and writing pages, and won't get much of its work done.
- This situation is called thrashing, it was a serious problem in early demand paging systems. And this is one of the problems that Operating System has to take care so that all the user level programs can be smoothly functionable and user can experience the real experience of that application.



Observations from the above graph:

- As the degree of multiprogramming Increases, CPU utilization also increases.
- If the degree of multiprogramming is Increased even further, thrashing sets in, And CPU utilization drops sharply.
- Under this situation, we must decrease the degree of the multiprogramming. And this peak represents **thrashing.**

4. Effects of Thrashing:

- Low CPU Utilization.
- Operating system thinking that it needs to increase degree of multiprogramming loads multiple processes into the memory or ready queue at the same time, allocating a limited(lesser) number of frames to each process.

5. Cause of Thrashing:

Thrashing results in severe performance problems. Consider the following scenario, which is based on the actual behaviour of early paging systems. The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page-replacement algorithm is used; it replaces pages without regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames. It starts faulting and taking frames away from other processes. These processes need those pages, however, and so they also fault, taking frames from other processes. These faulting processes must use the paging device to swap pages in and out. As they queue up for the paging device, the ready queue empties. As processes wait for the paging device, CPU utilization decreases. The CPU scheduler sees the decreasing CPU utilization and *increases* the degree of multiprogramming as a result. The new process tries to get started by taking frames from running processes, causing more page faults and a longer queue for the paging device. As a result, CPU utilization drops even further, and the CPU scheduler tries to increase the degree of multiprogramming even more. Thrashing has occurred, and system

throughput plunges. The page fault rate increases tremendously. As a result, the effective memory-access time increases. No work is getting done, because the processes are spending all their time paging. This phenomenon is illustrated in Figure 8.18, in which CPU utilization is plotted against the degree of multiprogramming. As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.

Methods to deal with trashing:

- Effect of thrashing can be reduced by using the local replacement algorithm. With local replacement algorithm, if one process starts thrashing, it cannot steal frames from another process and cause the latter to thrash as well. However, the problem is not entirely solved. If processes are thrashing, they will be in the queue for the paging device most of the time. The average service time for a page fault will increase because of the longer average queue for the paging device. Thus, the effective access time will increase even for the paging device. Thus, the Effective Access Time (EAT) will increase even for the process that is not thrashing.
- Provide as many frames as required for the process. We can get to know about the number of frames using working set model strategy that is discussed below.
- If a single process is too large for memory, there is nothing the OS can do. That process has to be simply terminated.
- If the process arises because of the sum of the several processes
    - Firstly, try to find out how much memory is actually required for each process to prevent thrashing. This is called its working set.
    - Only allow a few numbers of processes to execute at a time, such that their working sets fit in memory.

When does thrashing occur?

- If the sum of the size of locality of all working sets of all processes exceeds the total memory size then thrashing occur.
  $\sum$ size of locality > Total memory size
- If we can load localities of all the processes in main memory then thrashing will not occur but there is chance for page fault when there is change in locality.
- Thrashing generally occurs on a program that works with huge data structures, as its large working set causes continual page faults that drastically slow down the system.

6. Other Forms of Thrashing:

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

1. TLB (Translation Lookaside Buffer) Thrashing
2. Heap Thrashing
3. Cache Thrashing
4. Process Thrashing

TLB (Translation Lookaside Buffer) Thrashing:

Where the 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.

Heap thrashing:

Frequency 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.

Cache Thrashing:

This occurs 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.

Process Thrashing:

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

Thrashing is also used in contexts other than virtual memory systems; for example, to describe cache issues in computing or silly window syndrome in networking.

7. Locality Model:

To prevent thrashing, we must provide a process with as many frames as it needs. In order to know the total number of frames required by a particular process there are several techniques. The working-set strategy starts by looking at how many frames a process is actually using. This approach defines the **locality model** of process execution.

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. Consider a function for example say 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. Thus, we see that localities are defined by the program structure and its data structures. The locality model states that all programs will exhibit this basic memory reference structure. Note that the locality model is the unstated principle behind the caching discussions so far in this book. If accesses to any types of data were random rather than patterned, caching would be useless. Suppose we allocate enough frames to a process to accommodate its current locality. It will fault for the pages in its locality until all these pages are in memory; then, it will not fault again until it changes localities. If we do not allocate enough frames to accommodate the size of the current locality, the process will thrash, since it cannot keep in memory all the pages that it is actively using.

8. Different methods to control Trashing:

There are two known methods for controlling Thrashing.

They are: 1. Working set model

2. Page Fault Frequency

8.1 Working – set model:

- Most programs operate on a small number of code and data pages compared to the total memory the program requires. The pages most frequently accessed are called the *working set*.
- This is implemented based on locality of reference or locality model previously discussed.
- Works by assuming past predicts future. Changes continuously and hence very hard to maintain an accurate number.
- 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 **Δ**, the working set is defined as the set of pages in the most recent **Δ** page references. Hence, all the actively used pages would always end up being a part of the working set.

  Here **Δ** = Working – set window = A fixed number of page references

  It has values in terms of number of instructions.

  E.g.: **Δ** = 10,000 instructions. Mean we are going to have 10,000 reference pages.

- If a page is in active use, it will be in the working set. If the page is not in the use it will be dropped from the working set.
- Accuracy of the working set depends on the selection of the '**Δ'.**
- Working Set Size (WSS) of a process i = Total number of pages referenced in the most recent **Δ.**
  - If **Δ** is very small means that it will not encompass the entire locality.
  - If **Δ** is very large means that several localities are available in that working set.

    - If **Δ** is Infinity then that means that the entire program is available in the working set.
- If D is the total demand for frames and WSS_i is the working set size for a process i, then $D = \sum WSS_i$
- 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.
    - If D <=m, then there would be no thrashing but page fault might occur.
- If there are some enough extra frames, then some more processes can be loaded into 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 (i.e., Swapped out of memory).
- The Operating system monitors the working set of each process and allocate frames accordingly to each of the processes.

8.1.1 Keeping track of Working set:

Approximate with interval timer + a reference bit

Example: **Δ** = 10,000

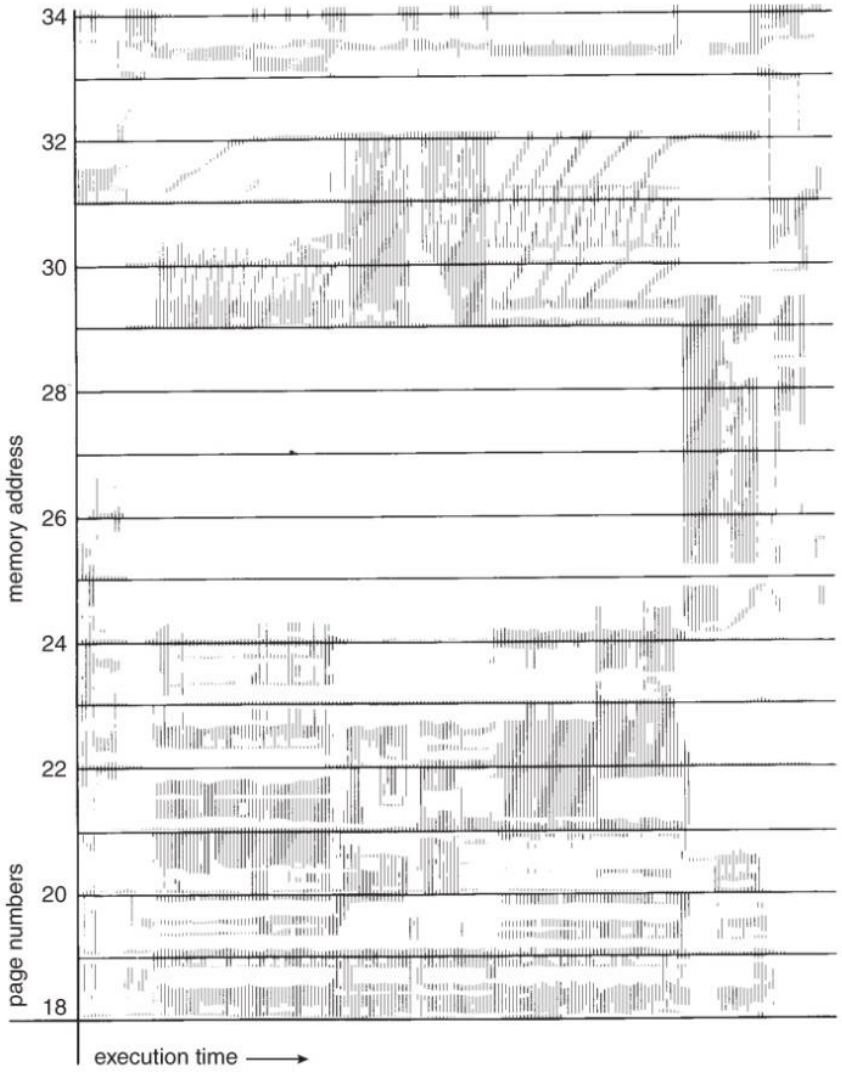
- Timer interrupt after every 5000-time units.
- Keep in memory 2 bits for each page.
- Whenever a timer interrupt copy and sets the value of all reference bits to 0.
- If one of the bits in memory = 1, then that page is in working set.
- But this is not completely accurate since it is only checking after 5000 time units.
- So, improvement would be "keep in memory 10 bits for each page and timer interrupt after every 1000 time units ".

8.1.2 Principle of Locality:

- Principle of Locality also called as Locality of reference is the tendency of a processor to access the same set of memory locations over a short period of time.
- There are two basic types of reference localities they are temporal locality and spatial locality.
- Temporal locality refers to the reuse of specific data, resources, within a small-time duration.
- Spatial locality refers to the use of data elements within relatively close storage location. Sequential locality is a special case of spatial locality occurs when the data elements are arranged and accessed linearly i.e., traversing the elements in a one-dimensional array.
- Program and data references within a process ten to cluster.
- Only a few pieces of a process will be needed over a short period of time.
- Possible to make intelligent guesses about which pieces will be needed in the future.
- This suggests that virtual memory may work efficiently.

8.1.3 Locality in Memory Reference Pattern:

Example



How can we find a working set model?

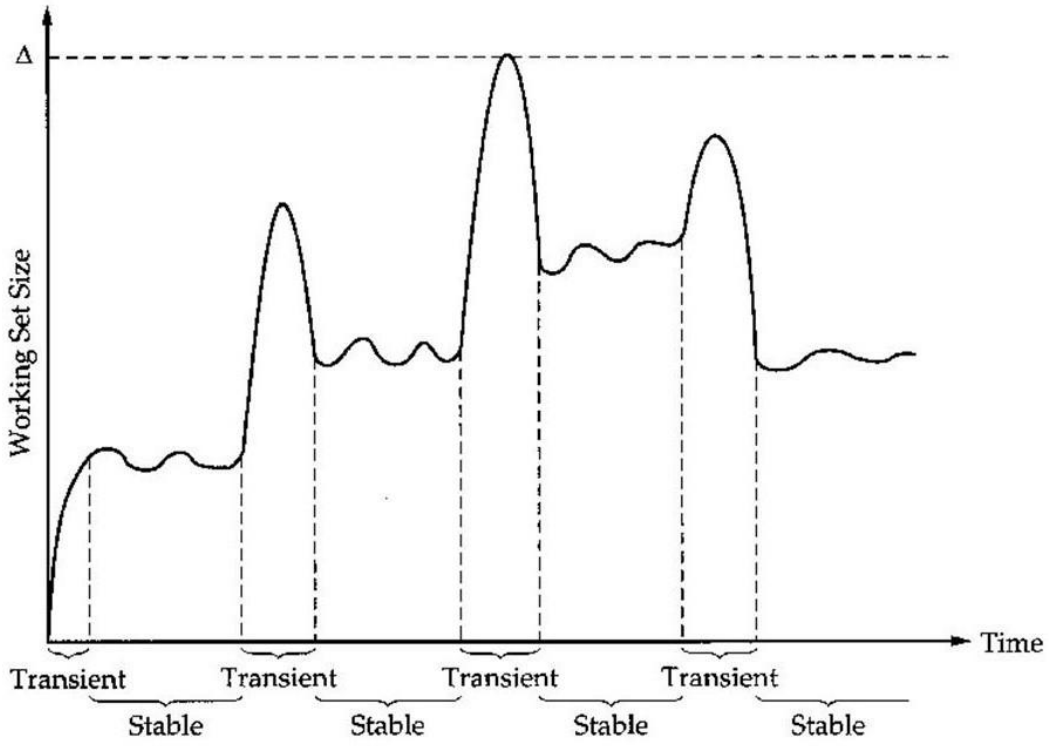
Suppose **Δ** = 10 and Consider the sequence of pages 2, 1, 3, 3, 4, 5, 7, 7, 8, 9, 6, 2, 3, 4, 1, 2, 3, 4, 4, 4, 3, 4, 3, 4, 4, 4, 6 be some page sequence

then if t1 denotes the presently reading page is 7 then WS(t1) = {1,2,3,4,5,7,8,9}

if t2 denotes the presently reading page is 6 then WS(t2) = {3,4}

Here WS denotes unique set of pages that are referenced in the last **Δ** pages. In this way we can find the working set model for a given problem by including all the pages in the past **Δ** pages. But this changes continuously hence hard to maintain accurate number.

8.1.4 Graph of Working set size over time:



Graph of Working set size variations w.r.t time

8.1.5 Allocation of Frames:

- Each process needs minimum number of pages. Consider a single OS with 256k memory, with page size of 1K. Suppose that the OS takes 50K, Leaving the other 206 frames for the user process.
- So now these 206 frames are available for processes and a process can get required pages from the first 206 page faults from the free frame list. But when the free frame list is exhausted, then a page replacement algorithm is used to allocate the frame.

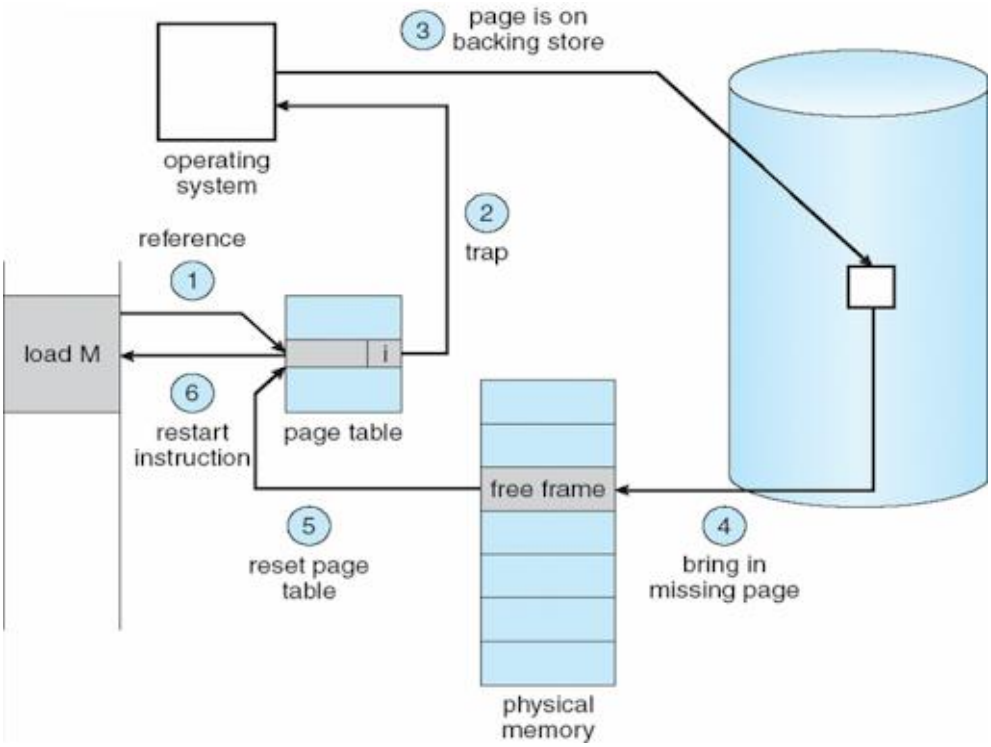
Constraints:

- Can't allocate more than the total number of available frames.
- Only a minimum number of frames can be allocated and this number depends on the Instruction Set Architecture (ISA).
- As the number of frames allocated decreases, the page fault increases and this slows down the process execution.

Benefits of using this model:

This technique prevents thrashing along with ensuring the highest degree of multiprogramming possible. Thus, it helps in optimizing the CPU utilisation.

8.1.6 Steps in handling a Page Fault:



- The process has requested a page that is not currently in the main memory.
- Check an internal table for the target process to determine if the reference was valid (do this in hardware).
- If it was valid, but page isn't available, then try to get if from the secondary storage.
- Find a free frame; a page of physical memory not currently in use (May be need to free up a page).
- Schedule a disk operation to read the desired page into the newly allocated frame.
- When the memory is filled, modify the page table to show that the page is now available in the memory.
- Restart the instruction that failed.

What if no free pages available?

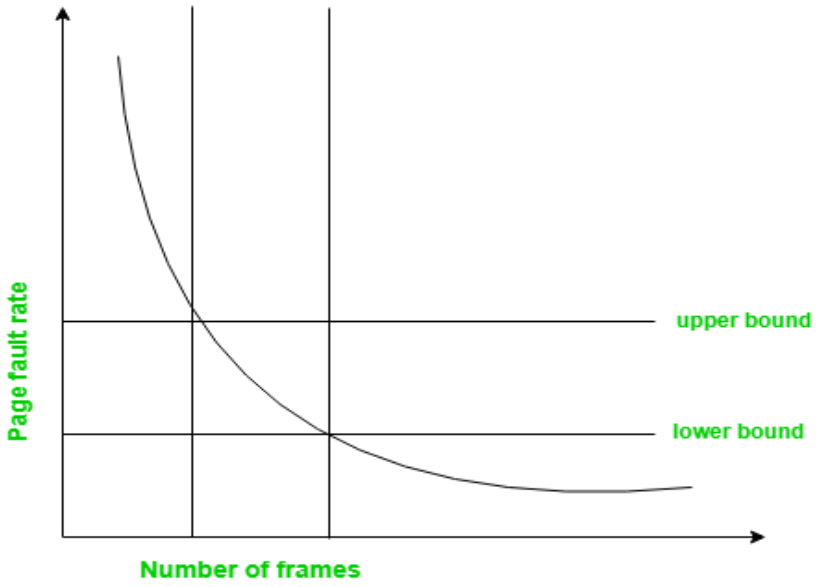
Then we have to use some Page replacement algorithm to find some page in memory which is not really in use and check if its dirty bit is 0 or 1. If it's 0 then nothing else is written into that page by the process so we can just replace the page with the required page. But if the dirty bit of that page is 1 then we have to copy that page back to its secondary storage i.e., disk and then replace this with the required page. Page Replacement completes separation between logical memory and physical memory. Thus, a large virtual memory can be provided on a smaller physical memory.

8.2 Page fault Frequency: One of the best techniques for computing working sets

- Major issues with Thrashing are the high page fault rate and hence the concept here is to control the page fault rate.
- Working set model requires need of more hardware and also, we need to check timer interrupts and that time we have to check all the reference bits.
- This is a more direct approach than the previously discussed Working set model.
- At any given time, each process is allocated a fixed number of physical page frames (assumes per-process replacement).
- Monitor the rate at which page faults are occurring for each process. This works under local replacement policy.
- If the rate gets too high for a process, assume that its memory is overcommitted. Increase the size of its memory pool. (i.e., process gains frames)
- If the rate gets too low for a process, assume that its memory pool can be reduced in size. (i.e., process loses frames)
- Page Fault Frequency is denoted by PFF and is also referred to as Page Fault Rate.
- PFF = No. of page faults / Total number of instructions executed

8.2.1 Relationship between Page Fault rate and Number of Frames:

The below plot shows upper and lower boundaries for Page Fault Frequency and shows its relation with the number of frames.



Observations from the graph:

- If the Page Fault Frequency falls below lower limit, frames can be removed from the process.
- If the Page Fault Frequency exceeds the upper limit, then a greater 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 above diagram.
- However, if Page Fault rate is high but 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 be later restarted.

8.2.2 Relationship between Page Fault Frequency and Working set:

The below graph depicts direct relationship between working set of a process which changes over time with page fault frequency shown by peaks and valleys over time.

Page Fault Frequency over time

Observations from the graph:

There is a direct relationship between the Page Fault Frequency of a process and its working set. As shown in the above graph Working set over time, the working set of a process changes over time as references to data and code sections move from one locality to another. Assuming there is sufficient memory to store the working set of a process (that is, the process is not thrashing), the Page Fault Frequency of the process will transition between peaks and valleys over time. This general behaviour is shown in above graph of Page Fault Frequency over time.

Any particular peak in the Page Fault Frequency occurs when we begin demand paging a new locality. However, once the working set of this new locality is in memory, the Page Fault Frequency falls. When the process moves to a new working set, the Page Fault Frequency rises towards a peak once again, returning to a lower rate once the new working set is loaded into memory. The span of time between the start of one peak and the start of the next peak represents the transition from one working set to another working set.

9. Modern perspectives of Thrashing:

- None of these solutions is very good:
  - Once a process becomes inactive, it has to stay inactive for a long time (many second), which results in poor response for the user.
  - Scheduling the balance set is tricky.
- In practice, today's operating systems don't worry much about thrashing:
  - Typically, users based on their requirements can buy more memory.
  - Or, manage balance set by hand.

- Thrashing was a bigger issue for timesharing machines with dozens or hundreds of users:
  - Why should I stop my processes just so you can make progress?
  - System had to handle thrashing automatically.

10. Conclusion:

The performance degradation or collapse brought about by excessive paging in computer systems, known as Thrashing can be traced to the very large speed difference between the main memory and secondary storage. The large traverse time between these two levels of memory makes efficiency very sensitive to the changes in the missing-page probability. Certain paging algorithms permit this probability to fluctuate in accordance with the total demand for memory, making it easy for attempted overuse of memory to trigger a collapse of service. The notion of locality and based on it, the working set model, can lead to a better understanding of the problem, and brings to solutions. If memory allocation strategies guarantee that the working set of every active process is present in main memory, it is possible to make programs independent one another in the sense that the demands of one program do not affect the memory acquisition of another. Then the missing page probability depends only on the choice of the working set. So, the choosing right working set size will help in better controlling Thrashing. Thus, using Working set model and Page Fault frequency we could reduce Thrashing to a greater extent.

11. References:

- Class Notes (https://drive.google.com/drive/folders/1dMsS4h8iugIh_pR5mfdhMCSqiDqkvK6X?usp=sharing)
- https://www.geeksforgeeks.org/techniques-to-handle-thrashing/
- https://web.stanford.edu/~ouster/cgi-bin/cs140-winter12/lecture.php?topic=thrashing
- https://en.wikipedia.org/wiki/Thrashing_(computer_science)
- https://www.scs.stanford.edu/12au-cs140/notes/l9.pdf
- https://cs.uwaterloo.ca/~brecht/courses/epfl/Possible-Readings/vm-and-gc/thrashing-denning-afips-1968.pdf
- Operating System Concepts 9th Edition Gagne, Silberschatz, Galvin Textbook.
- http://courses.mpi-sws.org/os-ss11/lectures/mem4.pdf
- http://www.freebookcentre.net/CompuScience/Free-Operating-Systems-Books-Download.html
- http://www.cse.psu.edu/~trj1/cse473-s08/slides/cse473-lecture-17-virtual.pdf
- https://www.youtube.com/watch?v=V66Tz5qUwmM