Operating System-2 : CS3523

Lecture Notes : Virtual Memory (Chapter 10)

- Rajesh Kedia

CS21BTECH11020 (Harsh Goyal)

CONTENTS

I. PAGE REPLACEMENT

It prevents over-allocation of memory by modifying page-fault service routine / handler to include page replacement, i.e swapping of pages and freeing their corresponding frames so that other processes can use them.

Use of modify bit (also known as dirty bit) reduces the extra overhead in swapping. If the bit is set, the page is modified (or dirty), so we have to write the page to storage and load the new page to that frame (2 I/O). If the bit is not set means the page is in read-only, we can overwrite its content with the new page (1 I/O). It improves page-fault service time.

Page replacement completes separation between logical memory and physical memory - large virtual memory can be provided on a smaller physical memory.

A. Basic Page Replacement

1) Find the location of the desired page in secondary storage.
2) Find a free frame:
   a) If there is a free frame, use it.
   b) If there is no free frame, use a page-replacement algorithm to select a **victim frame**.
   c) Write the victim frame to secondary storage (if necessary).
   d) Update the page and frame tables accordingly.

3) Read the desired page into the newly freed frame; change the page and frame tables.
4) Continue the process from where the page fault occurred.

Page replacement is basic to demand paging. Without page replacement we get restricted by the size of physical memory, therefore no virtual memory can be implemented. To solve this problem, we must develop

Frame-allocation algorithm determines

- How many frames to give each process
- Which frames to replace

Page-replacement algorithm

- Want lowest page-fault rate on both first access and later accesses.

These algorithms are evaluated by running them on a particular string of memory references **(reference string)** and computing the number of page-faults on that string.

- String is just page numbers, not full addresses
- Repeated access to the same page does not cause a page fault
- Results depend on number of frames available
- Example : 7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

B. Page Replacement Algorithms

Stack Algorithm: A stack algorithm is an algorithm for which it can be shown that the set of pages in memory for n frames is always a subset of the set of pages that would be in memory with $n + 1$ frames.

Note: Number of page fault decreases with the number of frames. When some algorithm contradicts this, it is known as **Belady's anomaly**.

Fig. 4: Graph of page faults versus number of frames with no Belady's anomaly

FIFO (First In First Out)	**OPT** (Optimal Page Replacement)	**LRU** (Least Recently Used)
A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.	The optimal algorithm replaces the page that will not be used for the longest period of time.	An LRU algorithm replaces the page that has not been used for the longest period of time.
Generally, it creates more page-fault than other two algorithm.	It is an optimal algorithm. It produces least number of page-fault, viz. ones which are unavoidable.	It produces less page-fault than FIFO in general but needs hardware support to implement.
It is implemented using a FIFO Queue.	It cannot be implemented since we can not predict the future.	It can be implemented using a counter (low count value show least recently used page) as well as stack (least recently used page on top).
It shows Belady's anomaly. Example : 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 , page fault increases when number of frames shifts from 3 to 4.	It does not show Belady's anomaly.	It does not show Belady's anomaly.
It is not a Stack Algorithm.	It is a Stack Algorithm.	It is a Stack Algorithm.

C. LRU Approximation Page Replacement

LRU need special hardware support and still slow. Therefore, there are other algorithms which can be approximated to LRU

Pages contain a **reference bit** which is set on when the page is referenced. Initally all reference bits are initialised to 0.
Few Algorithms are:

- **Additional-Reference-Bits Algorithm**
  1) Keep history bits (say 8-bit) in each page. At some regular interval (say, 100 milliseconds), timer interrupt is made which shift the higher order bit to the right and discard the lowest bit (11101011 -> 01110101).
  2) If a page is referenced very frequently, Its higher order bit (refernce bit) is always 1. Therefore, It corresponds to the larger decimal value. Hence, Page with lower corresponding decimal value is evicted since it is the one with is not used in recent past intervals.
- **Second-Chance Algorithm**
  1) Also Known as Clock Algorithm.
  2) When a page has been selected, however, we inspect its reference bit. If the value is 0, we proceed to replace this page; but if the reference bit is set to 1, we give the page a second chance and move on to select the next FIFO page. When a page gets a second chance, its reference bit is cleared, and its arrival time is reset to the current time.
  3) Second-chance replacement degenerates to FIFO replacement if all bits are set.

- **Enhanced Second-Chance Algorithm**
  1) We can enhance the second-chance algorithm by considering the reference bit and the modify bit as an ordered pair.
     a) (0, 0) neither recently used nor modified - best page to replace
     b) (0, 1) not recently used but modified - not quite as good, because the page will need to be written out before replacement
     c) (1, 0) recently used but clean - probably will be used again soon
     d) (1, 1) recently used and modified - probably will be used again soon, and the page will be need to be written out to secondary storage before it can be replaced
  2) Clock Algorithm is used but instead of checking whether the page to which we are pointing has the reference bit set to 1,we examine the class to which that page belongs. We replace the first page encountered in the lowest nonempty class. We may have to scan the circular queue several times before we find a page to be replaced.

D. Counting-Based Page Replacement

We can keep a counter of the number of references that have been made to each page and develop the following two schemes.

- **Least Frequently Used (LFU)** : This Page-replacement algorithm requires that the page with the smallest count be replaced. The reason for this selection is that an actively used page should have a large reference count. A problem arises, however, when a page is used heavily during the initial phase of a process but then is never used again. Since it was used heavily, it has a large count and remains in memory even though it is no longer needed. One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.
- **Most Frequently Used (MFU)** : This Page-replacement algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

E. Page-Buffering Algorithms

- Idea is to keep a pool of free frame (page buffer). When a process demands for a frame in the free-frame pool which is empty , It can be allocated from that page buffer. Later, a victim can be selected and evicted.

This procedure allows the process to restart as soon as possible, without waiting for the victim page to be written out. When the victim is later written out, its frame is added to the free-frame pool.

- An expansion of this idea is to maintain a list of modified pages. Whenever the paging device is idle, a modified page is selected and is written to secondary storage. Its modify bit is then reset. This scheme increases the probability that a page will be clean when it is selected for replacement and will not need to be written out.
- We can keep the free frame contents intact and note what is in them so that If It referenced again before reused, we do not need to load contents again from disk. It is generally useful to reduce penalty if wrong victim frame is selected

F. Application and Page Replacement

- Memory intensive applications can cause double buffering. OS as well as application can keep the memory for buffering.
- Some operating systems give special programs the ability to use a secondary storage partition as a large sequential array of logical blocks, without any file-system data structures. This array is some- times called the **raw disk**, and I/O to this array is termed raw I/O.

II. Allocation of frames

1) Minimum number of frames per process are decided by the architecture whereas the maximum number of frames per process are decided by the amount of available physical memory.
2) Allocating minimum number of frames to process improves performance. Since less frame leads to more page fault which ultimately decrease system performance.

There are some allocating algorithms for frame distribution:-

A. Allocation Algorithms

- **Equal allocation algorithm**
  1) If there are m frames are available and there are n processes, each process will get m/n frames.
  2) Since need of every process is not same, some might used small portion of the frame allocated leads to wastage of memory.
- **Proportional allocation algorithm**
  1) Memory are allocated to each process according to their size. Let the virtual memory size of the process p_i is s_i. Let
  $$S = \sum_i s_i$$
  . If the total number of available frame are m. Each process will get a_i frame ,which is
  $$a_i = \frac{s_i}{S} \times m$$
  2) Increase in multiprogramming leads processes to lose its memory to fulfill the need of other processes whereas decrease in multiprogramming makes the process to distribute its frame among the remainig processes.
- In both the algorithm, high priority processes are treated same as low priority processes. To give more memory to the high priority processes, instead of using relative size of the process, we can either use priorities of processes or a combination of size and priority as our Proportional factor.

B. Global versus Local Allocation

Global Replacement	Local Replacement
A process can select a replacement frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another	Each process select from only its own set of allocated frames.
Global Replacement policy allows a high-priority process to increase its frame allocation at the expense of a low-priority process	With a local replacement strategy, the number of frames allocated to a process does not change.
Set of pages in memory for a process depends not only on the paging behavior of that process, but also on the paging behavior of other processes. Therefore, the same process may perform quite differently (for example, taking 0.5 seconds for one execution and 4.3 seconds for the next execution) because of totally external circumstances	The set of pages in memory for a process is affected by the paging behavior of only that process leads to More consistent per-process performance

Note: Global replacement generally results in greater system throughput. Hence It is more commonly used.

Implementing Global Replacement policy :

- A kernel routines (known as **Reapers**) are triggered when number of available frames fall below a minimum threshold which start reclaiming pages until the number of free frames (or available memory) crosses a certain maximum threshold.

- If the reaper routine is unable to maintain the list of free frames below the minimum threshold, it suspend the second-chance algorithm and use pure FIFO.
- In Linux, when the amount of free memory falls to very low levels, a routine known as the **out-of-memory (OOM) killer** selects a process to terminate, thereby freeing its memory. Which process to terminate is decided by its oom_score (higer oom_score get terminated) which is proportional to the memory process is using.

C. Major and Minor page faults

- **Major Faults :** A major page fault occurs when a page is referenced and the page is not in memory. Servicing a major page fault requires reading the desired page from the backing store into a free frame and updating the page table.
- Demand paging typically generates an initially high rate of major page faults.
- **Minor Faults :** Minor page faults occur when a process does not have a logical mapping to a page, yet that page is in memory.
- Minor fault occur because one of the two reason:
  1) First, a process may reference a shared library that is in memory, but the process does not have a mapping to it in its page table. In this instance, it is only necessary to update the page table to refer to the existing page in memory.
  2) Second, when a page is reclaimed from a process and placed on the free-frame list, but the page has not yet been zeroed out and allocated to another process. When this kind of fault occurs, the frame is removed from the free-frame list and reassigned to the process.

Note: One can check the major and minor faults in their Linux system using command `ps -eo min flt,maj flt,cmd`

D. Non-Uniform Memory Access

- In real system, memory are not accessed equally. Many system are NUMA which is speed to access to memory varies.
- The performance difference depends on how is CPU interconnected to the memory. Each CPU can made a faster access to its own local memory than other memory

- Memory frames are allocated "as close as possible" (closeness refers to the minimum latency) to the CPU on which the process is running in order to improve the performance.
- To further improve the performance, scheduler keep track of the last CPU on which the process runs. If the scheduler tries to schedule each process onto its previous CPU, and the virtual memory system tries to allocate frames for the process close to the CPU on which it is being scheduled, then it leads to improvement in cache hits and decrement in memory access time.
- Solaris uses **lgroups** (locality groups) to solve this problem.Solaris tries to schedule all threads of a process and allocate all memory of a process within an lgroup. If that is not possible, it picks nearby lgroups for the rest of the resources needed. This practice minimizes overall memory latency and maximizes CPU cache hit rates.

III. THRASHING

When we do not have enough number of frames and each time whenever a page evict (due to page fault) which is in active use, process make page-fault repeatedly. A process is **thrashing** if it is spending more time in paging than in executing. This leads to poor performance.

A. Cause of Thrashing

Operating system montiors the CPU utilization. If CPU utilization is too low, it increases the degree of multi-programming by introducing new processes to the system. New processes demand for frame allocation and start faulting. If number of frame is not enough, Some frames get evicted. Since all process are active, Evicted frame can be requested by any other processes which can cause faulting followed by eviction of some other frames.

These faulting processes use the swap device to move page in and out. Due to multiple swapping request, processes will queue up for the paging device. Therefore, ready queue becomes empty.

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. Adding new processes further decrease CPU utilization cause OS to add more processes which ultimately cause thrashing and and system throughput to plunge.

Due to increase in page-fault rate, the effective memory-access time increases. No work is getting done, because the processes are spending all their time in paging.

Fig. 2: As the degree of multi- programming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased further, thrashing sets in, and CPU utilization drops sharply.

Possible solution :

1) We can limit the effects of thrashing by using a local replacement algorithm (or priority replacement algorithm). As mentioned earlier, local replacement requires that each process select from only its own set of allocated frames. Thus, 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 a process that is not thrashing.

2) Using locality model of process execution.
   a) Key idea is process should be provided with as many frames as it needs at particular instance.
   b) The locality model states that, as a process executes, it moves from locality to locality. A locality is a set of pages that are actively used together. A running program is generally composed of several different localities, which may overlap.
   c) Example: when we call a function in a program, process might jump to a new locality and remains their until function returns. This locality can be accessed again after sometimes.
   d) The locality model states that all programs will exhibit this basic memory reference structure (or pattern).
   e) Now, Process 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.
   f) If enough frames are not provided to accomodate it current locality, process will thrash.

B. Working-Set Model

The **working-set model** is based on the assumption of locality. **Working-set window**, Δ is used to examine the most recent page references. The set of pages in the most recent Δ page references is the **working set** . If a page is in active use, it will be in the working set. If it is no longer being used, it will drop from the working set Δ time units after its last reference.
Working set is an approximation of the program's locality.

The accuracy of the working set depends on the selection of Δ

1) if Δ is too small, it will encompass the entire locality
2) if Δ is too large, it may overlap several locality. In extreme cases, Δ is infinite, all the demanded page is in working set.

How working set prevents thrashing?

1) The operating system monitors the working set of each process and allocates to that working set enough frames to provide it with its working-set size.
2) If there are enough extra frames, another process can be initiated.
3) If the sum of the working-set sizes increases, exceeding the total number of available frames, the operating system selects a process to suspend. The process's pages are written out (swapped), and its frames are reallocated to other processes. The suspended process can be restarted later.

It keep the degree of the multiprogramming as high as possible (maximizing CPU utilization) as well as avoiding thrashing.

Working-Sets and Page-Fault Rates :- While demand paging, when process shifts from one locality to other, large number of page faults occur whereas when process remains in its locality, very low page faults happens. 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

Major difficulty is to maintain the working set and determining its size.

C. Page-Fault Frequency

1) When page fault is above some threshold (upper limit), OS allocates new frames to that process.
2) When page fault is below some threshold (lower limit), OS remove some frames from that process.
3) As with working-set strategy, if the page fault increases and no free frames are available, OS suspend some process and reallocate its frames to other process. The suspended process restarted later.