CGS 3763: Operating System Concepts Spring 2006

Memory Management – Part 6

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790
http://www.cs.ucf.edu/courses/cgs3763/spr2006

School of Electrical Engineering and Computer Science
University of Central Florida

LRU Approximation Algorithms
Counter-Based Algorithms

- Keep a counter of the number of references that have been made to each page.
- Least Frequently Used (LFU) Algorithm: replaces the page with the smallest counter value. 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 high reference count and remains in memory even though it is no longer needed.
  - One solution is to shift the bits in the counter to the right by 1 bit at regular intervals, forming an exponentially decaying average use count.

LRU Approximation Algorithms
Counter-Based Algorithms (cont.)

- **Most Frequently Used (MFU) Algorithm**: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

- As you might expect, neither LFU or MFU replacement is very common. The implementation of these algorithms is expensive, and they do not approximate the Optimal replacement algorithm well.

Allocation Of Frames

- In addition to selecting a victim for page replacement, we must also consider the allocation of frames to processes.
- We saw that with the FIFO page-replacement algorithm that the number of page faults may actually increase for an increase in frame allocation. Although stack algorithms do not suffer from Belady's anomaly, the performance of processes running under these types of page replacement protocols are certainly impacted by the number of frames allocated to the process.
- If for example, we have 100 free frames and two processes, how many frames does each process get?

Allocation Of Frames (cont.)

- Consider a single-user system with 128 KB of memory composed of pages 1 KB in size. This system would have 128 frames.
- Suppose the OS requires 35 frames, leaving 93 frames for the user process.
- Under pure demand paging, all 93 frames would initially be put on the free frame list. When a user process begins execution, it would generate a sequence of page faults.
- The first 93 faults would all get free frames from the free frame list.
- When the free frame list was exhausted, a page-replacement algorithm would be used to select one of the 93 in-memory pages to be replaced with the 94^{th}, and so on.
- When the process terminates, the 93 frames would be returned to the free frame list.

Allocation Of Frames (cont.)

- There are many variations on the simple strategy outlined on the previous page.
  - We could require that the OS allocate all its buffer and table space from the free-frame list, when not used by the OS, it can be used to support user paging.
  - We could try to maintain a three page frame reserve on the free frame list at all times. Thus, whenever a process page faults, there is always a free frame available to page into. While paging is occurring, a replacement can be selected, which is then written back to the disk as the user process continues to execute.
- Other variants are possible, but the basic strategy is clear: the user process is allocated any free frame.

Minimum Number Of Frames

- Strategies for the allocation of frames are constrained in a variety of ways.
- It is not possible, for example, to allocate more than the total number of available frames (unless there is page sharing).
- A minimum number of frames must also be allocated to each process.
- The obvious reason that a minimum number of frames must be allocated is performance.
  - As the number of frames allocated to each process decreases, the page-fault rate increases, slowing process execution.
- Another reason is that when a page fault occurs before an executing instruction is complete, the instruction must be restarted. Consequently, the process must have enough frames to hold all the different pages that any single instruction can reference.

Minimum Number Of Frames (cont.)

- An example is the IBM 370 MVC (move characters) instruction. This instruction takes 6 bytes and can straddle two pages. The block of characters to move and the area to which it is to be moved can each also straddle two pages. This situation would require six frames.
- The worst case scenario occurs when the MVC instruction is itself the operand of an EXECUTE instruction that straddles a page boundary; in this case, two additional frames are required, bringing the total to eight.
- Whereas the minimum number of frames is determined by the computer architecture (through its instruction set), the maximum number of defined by the amount of physical memory. In between these two values, we are left with a significant choice in frame allocation.

Frame Allocation Algorithms

- The easiest way to split *m* frames among *n* processes it to give each process an equal share, *m/n* frames. This scheme is known as equal allocation.
  - For example, if there are 93 frames and 5 processes, each process will get 93/5 = 18.6 = 18 frames. The leftover 3 frames can be used as a free frame pool.
- An alternative is to recognize that various processes will need differing amounts of memory.
  - For example, consider a system with 1 KB frames. A process of 10 KB and a second process of 127 KB are the only two processes running with 62 free frames. It makes no sense to give 31 frames to the 10 KB process which in the worst case will need only 10 frames, which will waste the other 21 frames, which could have been allocated to the 127 KB process.
- This alternative is known as proportional allocation.

Proportional Allocation

- Using proportional allocation we have the following:

$s_i = \text{size of virtual memory for process } p_i$

$S = \sum s_i$

$m = \text{total number of frames}$

$a_i = \text{allocation for } p_i = \frac{s_i}{S} \times m$

- Example

$m = 64$

$s_i = 10$

$s_2 = 127$

$a_1 = \frac{10}{137} \times 64 \approx 5$

$a_2 = \frac{127}{137} \times 64 \approx 59$

Note: With either equal or proportional allocation, the allocation may vary depending on the degree of multiprogramming. If the multiprogramming level is increased, each process will lose some frames to meet the allocation for the new process. Similarly, if the degree of multiprogramming is decreased, the frames that were previously allocated to the departed process can be spread over the remaining processes.

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size.

- If process P_i generates a page fault,
  - select for replacement one of its frames
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

- Another important factor in the way frames are allocated to the various processes is page replacement.
- With multiple processes competing for frames, we can classify page-replacement algorithms into two broad categories:
  - Global replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
  - Local replacement – each process selects from only its own set of allocated frames.
- For example, consider an allocation scheme where we allow high-priority processes to select frames from low-priority processes for replacement. A process can select a replacement among its own frames or the frames of any lower-priority process. This approach allows a high-priority process to increase its frame allocation at the expense of a low-priority process.

Global vs. Local Allocation (cont.)

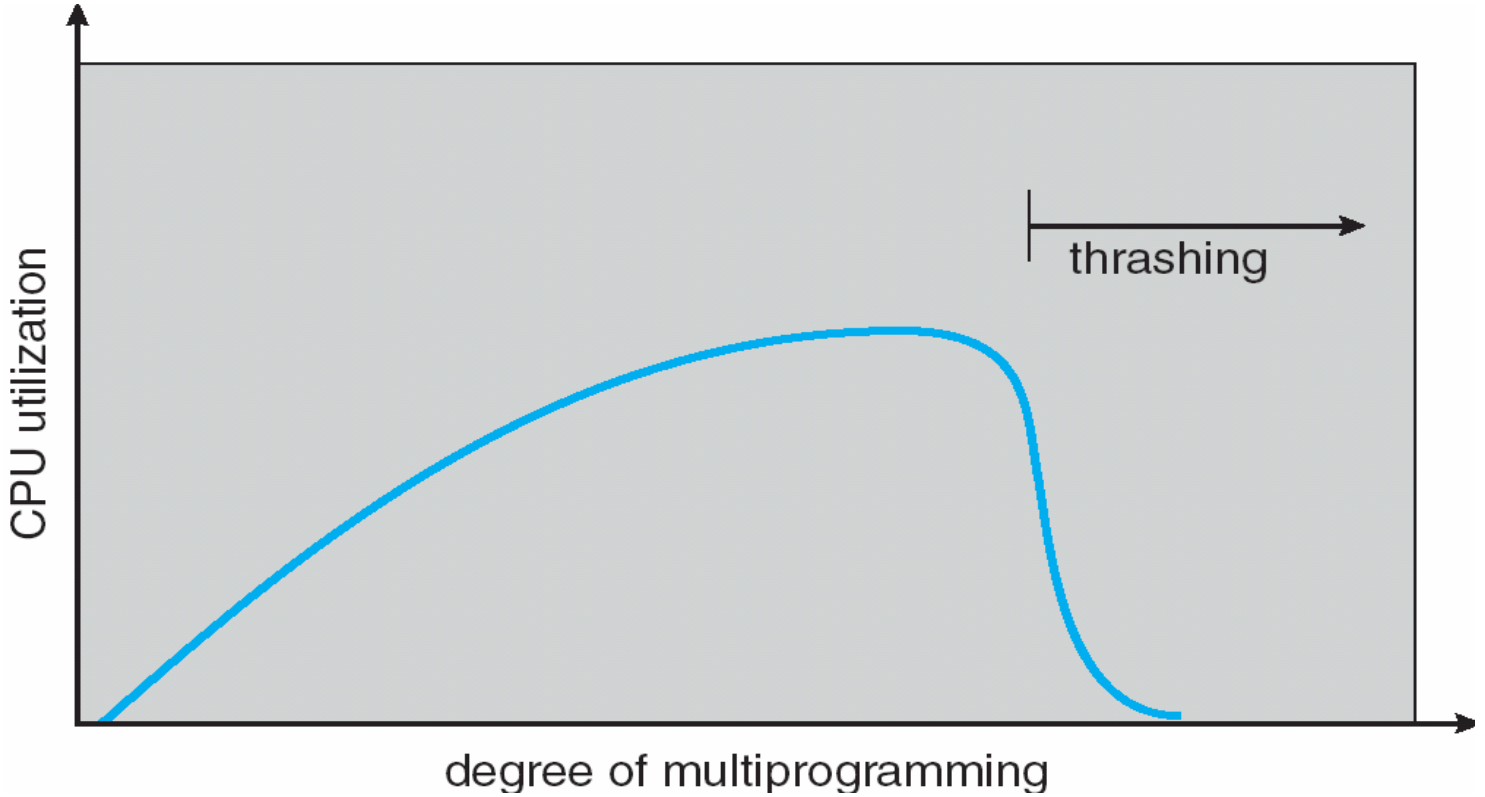
- With a local replacement strategy, the number of frames allocated to a process does not change (unless the degree of multiprogramming changes).
- With global replacement, a process may happen to select only frames allocated to other processes, thus increasing the number of frames allocated to it (assuming that other processes did not choose its frames for replacement).
- One problem with a global replacement algorithm is that a process cannot control its own page-fault rate. The 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 different executions because of totally external circumstances.
- Under local replacement, the set of pages in memory for a process is affected by the paging behavior of only that process.
- Local replacement might hinder a process, however, by not making available to it other, less used pages of memory.
- Thus, global replacement generally results in greater system throughput and is therefore the more common method.

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high.
- This leads to:
  - low CPU utilization
  - operating system thinks that it needs to increase the degree of multiprogramming
  - another process added to the system

- Thrashing ≡ a process is busy swapping pages in and out without accomplishing any real activity.

Thrashing (cont.)



Demand Paging and Thrashing

- Why does demand paging work?
  Locality model
  - Process migrates from one locality to another
  - Localities may overlap

- Why does thrashing occur?
  Σ size of locality > total memory size

Locality In A Memory-Reference Pattern