Principles of Operating Systems

Lecture 7 - Virtual Memory

Ardalan Amiri Sani (ardalan@uci.edu)

[*lecture slides contains some content adapted from previous slides by Prof. Nalini Venkatasubramanian, and course text slides © Silberschatz]*

Some benefits of virtual memory (with paging and segmentation)

- Only *PART* of the program needs to be in memory for execution.
- Allocated memory in logical address space can therefore be much larger than physical address space.
- Need to allow pages to be swapped in and out.

Demand Paging

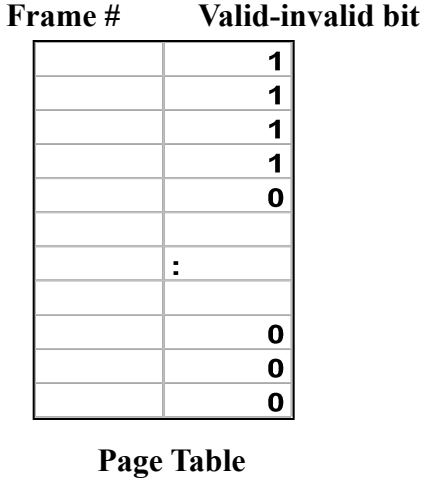
- Bring a page into memory only when it is needed.

Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- The first reference to an unmapped page will trap to OS with a page fault.
- OS looks at some data structure or another table to decide
  - Invalid reference - abort
  - Valid but not in memory.

Valid-Invalid Bit

- With each page table entry a valid-invalid bit is associated (1 ⇒ translation present, 0 ⇒ translation not present).
- Initially, valid-invalid bit is set to 0 on all entries.
  - During address translation, if valid-invalid bit in page table entry is 0 **--- *page fault*** occurs.
  - Example of a page-table snapshot



Page Table

Handling a Page Fault

- Page is needed - reference to page
  - Step 1: Page fault occurs - trap to OS (process suspends).
  - Step 2: Check if the virtual memory address is legitimate. Kill process if access is illegitimate. If legitimate, then page fault means the page content not in memory yet, continue.
  - Step 3: Bring into memory - Find a free page frame, find content on disk and fetch disk content into page frame. When disk read has completed, add virtual memory mapping to indicate that page is in memory.
  - Step 4: Restart instruction interrupted by illegitimate address trap. The process will continue as if page had always been in memory.

What happens if there is no free frame?

- Page replacement - find some page in memory that is not in use and swap it.
  - Need page replacement algorithm
  - Performance Issue - need an algorithm which will result in minimum number of page faults and page replacements.
- Same page may be brought into memory many times.

Performance of Demand Paging

- Page Fault Ratio - $0 \le p \le 1.0$
  - If $p = 0$, no page faults
  - If $p = 1$, every reference is a page fault
- Effective Access Time

  EAT = (1-p) * memory access +
  p * (page fault overhead +
  swap page out (only when needed) +
  swap page in +
  restart overhead +
  memory access)

Demand Paging Example

- Memory is always full
  - Need to get rid of a page on every fault
- Memory Access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Page fault and restart overheads are negligible
- Swap Page Time (either in or out) = 10 msec = 10,000 microsec

Demand Paging Example

- Memory is always full
  - Need to get rid of a page on every fault
- Memory Access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Page fault and restart overheads are negligible
- Swap Page Time (either in or out) = 10 msec = 10,000 microsec

  EAT = (1-p) *1 + p (15000 + 1) = 1 + 15000p microsec
- EAT is directly proportional to the page fault rate.

Page Replacement

- Determines which page need to be evicted to disk when space is needed in memory.

- With page replacement, large virtual memory can be provided on a smaller physical memory.

Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- Assume reference string in examples to follow is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

First-In-First-Out (FIFO) Algorithm

Reference String: 1,2,3,4,1,2,5,1,2,3,4,5

- Assume x memory frames

3 frames

Frame 1	1	4	5
Frame 2	2	1	3
Frame 3	3	2	4

9 Page faults

4 frames

Frame 1	1	5	4
Frame 2	2	1	5
Frame 3	3	2	
Frame 4	4	3	

10 Page faults

FIFO Replacement - *Belady's Anomaly* -- more frames does not mean less page faults

Optimal Algorithm

- Replace page that will not be used for longest period of time.
  - Typically not possible in a practical setting
  - Generally used to measure how well an algorithm performs.
  - Reference String: 1,2,3,4,1,2,5,1,2,3,4,5

4 frames

Frame 1	**1**	**4**
Frame 2	**2**	
Frame 3	**3**	
Frame 4	**4**	**5**

6 Page faults

Least Recently Used (LRU) Algorithm

- Use recent past as an approximation of near future.
- Choose the page that has not been used for the longest period of time.

- Reference String: 1,2,3,4,1,2,5,1,2,3,4,5

4 frames

Frame 1	**1**		**5**
Frame 2	**2**		
Frame 3	**3**	**5**	**4**
Frame 4	**4**	**3**	

8 Page faults

Implementation of LRU algorithm

- Counter Implementation?
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be evicted, look at the counters to determine which page to evicted (page with smallest time value).

Implementation of LRU algorithm

- Counter Implementation: slow!
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be evicted, look at the counters to determine which page to evicted (page with smallest time value).
  - Needs to search the page entries, which is slow
  - Needs hardware to update entries; software update will be slow

Implementation of LRU algorithm

- Counter Implementation: slow!
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be evicted, look at the counters to determine which page to evicted (page with smallest time value).
  - Needs to search the page entries, which is slow
  - Needs hardware to update entries; software update will be slow
- Stack Implementation?
  - Keeps a stack of page numbers
  - When a page referenced, move it to the top of the stack
  - No search required for replacement

Implementation of LRU algorithm

- Counter Implementation: slow!
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be evicted, look at the counters to determine which page to evicted (page with smallest time value).
  - Needs to search the page entries, which is slow
  - Needs hardware to update entries; software update will be slow
- Stack Implementation: slow!
  - Keeps a stack of page numbers
  - When a page referenced, move it to the top of the stack
  - No search required for replacement
  - Stack update mainly in software, which is slow

LRU Approximation Algorithms

- High-level idea: Use reference Bit (also called the accessed bit)
  - With each page, associate a bit, initially = 0.
  - When page is referenced, bit is set to 1 (by hardware).
  - Replace the one which is 0 (if one exists). Do not know order however.

LRU Approximation Algorithms

- Additional Reference Bits Algorithm
  - Record reference bits at regular intervals.
  - Keep 8 bits (say) for each page in a table in memory.
  - Periodically, shift reference bit into high-order bit, i.e. shift other bits to the right, dropping the lowest bit.
  - During page replacement, interpret the 8 bits as unsigned integer.
  - The page with the lowest number is the LRU page.

LRU Approximation Algorithms

- Second Chance
  - Core is a FIFO replacement algorithm
  - Implemented with circular queue (hence called the clock algorithm sometimes)
  - Need a reference bit.
  - When a page is selected, inspect the reference bit.
  - If the reference bit = 0, replace the page.
  - If page to be replaced (in clock order) has reference bit = 1, then
    - set reference bit to 0
    - leave page in memory
    - Move on to check the next page (in clock order). Checking of the next page will be subject to same rules.

LRU Approximation Algorithms

- Enhanced Second Chance
  - Need a reference bit and a modify bit as an ordered pair.
  - Modify bit is set by hardware when page is written to.
  - 4 situations are possible:
    - (0,0) - neither recently used nor modified - best page to replace.
    - (0,1) - not recently used, but modified - not quite as good, because the page will need to be written to disk before replacement.
    - (1,0) - recently used but not modified - probably will be used again soon.
    - (1,1) - probably will be used again, will need to write out before replacement - worst page to replace.

Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
  - LFU (least frequently used) algorithm
    - replaces page with smallest count.
    - Based on the argument that the page with the smallest count will not be used frequently in the future either
  - MFU (most frequently used) algorithm
    - replaces page with highest count.
    - Based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Page Buffering Algorithm

- Keep a pool of free frames
  - When a page fault occurs, choose victim frame.
  - Desired page is read into free frame from pool before victim is written out.
  - Allows process to restart soon, victim is later written out and added to free frame pool.

Page Buffering Algorithm

- Keep a pool of free frames
  - When a page fault occurs, choose victim frame.
  - Desired page is read into free frame from pool before victim is written out.
  - Allows process to restart soon, victim is later written out and added to free frame pool.
- Expansion 1
  - Maintain a list of modified pages. When disk is idle, write modified pages to disk and clear modify bit.

Page Buffering Algorithm

- Keep a pool of free frames
  - When a page fault occurs, choose victim frame.
  - Desired page is read into free frame from pool before victim is written out.
  - Allows process to restart soon, victim is later written out and added to free frame pool.
- Expansion 1
  - Maintain a list of modified pages. When disk is idle, write modified pages to disk and clear modify bit.
- Expansion 2
  - Keep frame contents in pool of free frames and remember which page was in frame. If desired page is in free frame pool, no need to page in.

Allocation of Frames

- Single user case is simple
  - User is allocated any free frame
- Problem: Demand paging + multiprogramming

Allocation algorithms

- Equal Allocation
  - e.g., if 100 frames and 5 processes, give each 20 frames.
- Proportional Allocation
  - Allocate according to the size of process
    - Sj = size of process Pj
    - $S = \sum Sj$
    - m = total number of frames
    - aj = allocation for Pj = $(Sj/S) * m$
    - If m = 64, $S1$ = 10, $S2$ = 127 then

      $a1$ = 10/137 * 64 ≈ 5

      $a2$ = 127/137 * 64 ≈ 59

Allocation algorithms (cont.)

- Priority allocation
  - May want to give high priority process more memory than low priority process.
  - Use a proportional allocation scheme using priorities instead of size

Global vs. Local Replacement

- Global Replacement
  - Selects a replacement frame from the set of all frames.
  - One process can take a frame from another.
  - Process may not be able to control its page fault rate.
  - Semi-global replacement: selects a replacement from frames allocated to some other processes, but not all.
- Local Replacement
  - Selects from process' own set of allocated frames.
  - Process slowed down even if other less used pages of memory are available.
- Global replacement has better throughput
  - Hence more commonly used.

Priority Allocation (cont.)

- If process Pi generates a page fault
  - select for replacement one of its frames (local allocation)
  - select for replacement a frame form a process with lower priority number. (semi-global allocation)

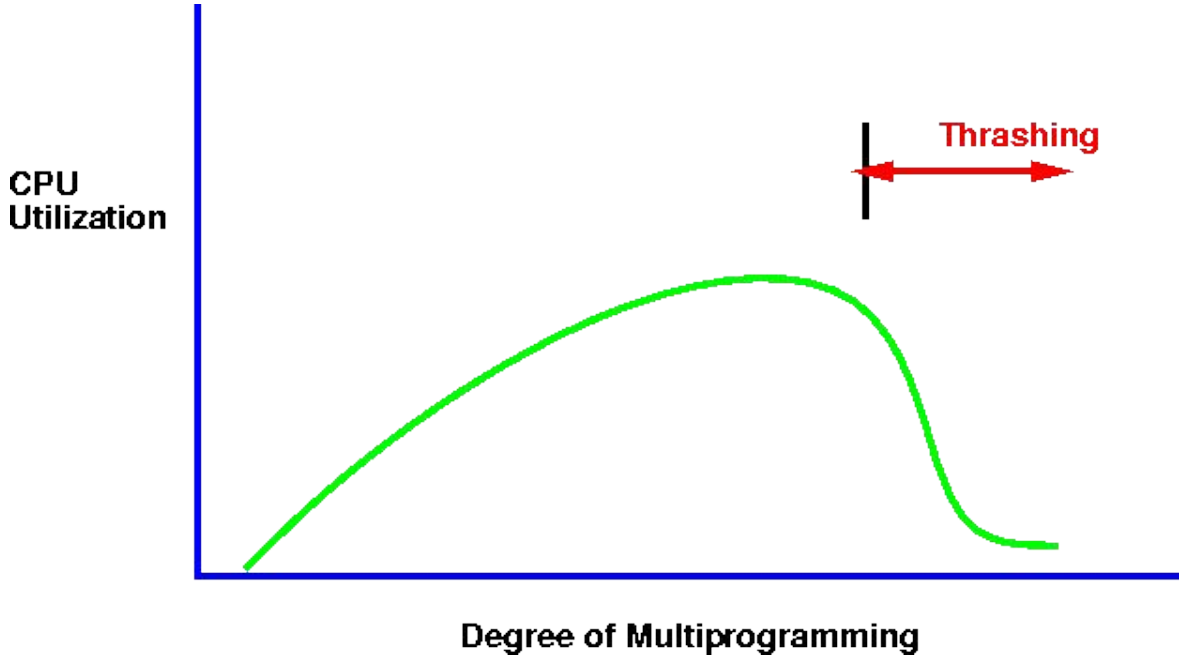
Thrashing

- If a process does not have enough frames, the page fault rate can be very high. This leads to:
  - low CPU utilization.

Thrashing

- If a process does not have enough frames, the page fault rate can be very high. This leads to:
  - low CPU utilization.
  - OS thinks that it needs to increase the degree of multiprogramming
  - Another process is added to the system.
  - System throughput plunges...
- *Thrashing*
  - A process is busy swapping pages in and out.
  - In other words, a process is spending more time paging than executing.

Thrashing (cont.)



Thrashing (cont.)

- Locality: set of pages that are actively used together.
  - Computations have locality!
  - Process migrates from one locality to another.
  - Localities may overlap.

Thrashing (cont.)

- Why does thrashing occur?
  - ∑ (size of locality) **>** total memory size
- Solution to thrashing:
  - If ∑ (size of locality) **>** total memory size, then suspend one of the processes

Working Set Model

- Working set is an approximation of the size of the locality
- Δ ≡ working-set window
  - a fixed number of page references, e.g., 10,000
  - WSSj (working set size of process Pj) = total number of pages referenced in the most recent Δ
    - If Δ too small, will not encompass entire locality.
    - If Δ too large, will encompass several localities.
    - If Δ = ∞, will encompass entire program.

Use working set model to avoid thrashing

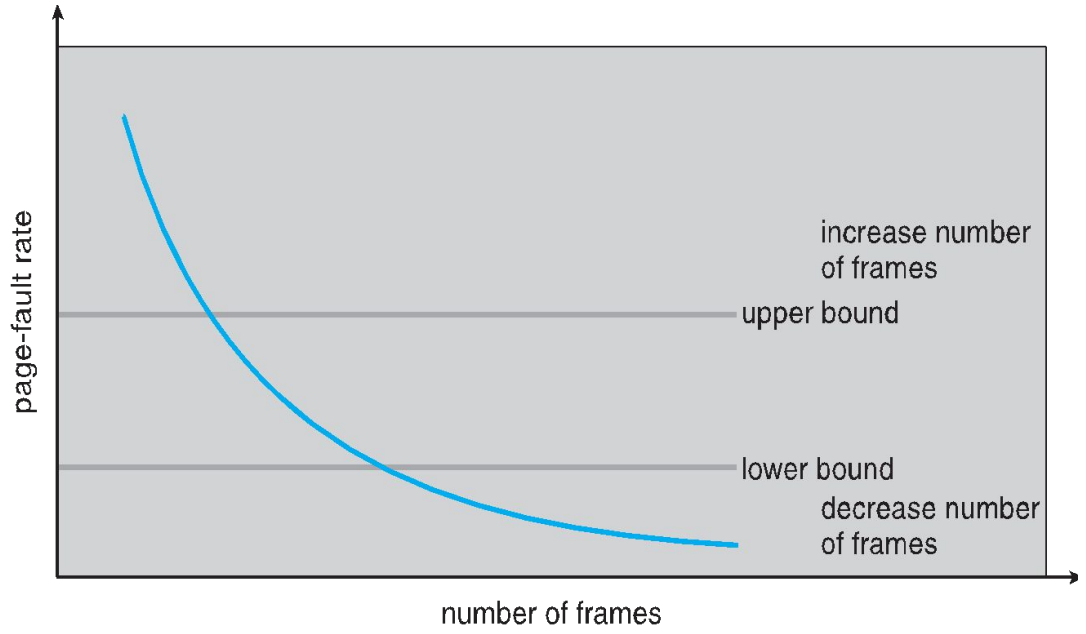
- D = ∑ WSSj ≡ total needed pages
  - If D > m (number of available frames) ⇒ thrashing
  - Policy: If D > m, then suspend one of the processes.

Measure working set in practice

- Use
  - interval timer + the reference bit
  - Example: Δ = 10,000 references
    - Let's assume we can get the timer to interrupt after every 5000 references (will be an approximation).
    - Whenever a timer interrupts, copy and set the values of all reference bits to 0.
    - Keep in memory 2 bits for each page
    - If one of the bits in memory = 1 ⇒ page in working set.
  - Not completely accurate - cannot tell where reference occurred.
  - Improvement - 10 bits and interrupt every 1000 time units.

Page fault Frequency Scheme

- Control thrashing by establishing *acceptable* page-fault rate (an upper bound and a lower bound).
  - If page fault rate too low, process loses frame.
  - If page fault rate too high, process needs and gains a frame.



Demand Paging Issues

- Prepaging
  - Tries to prevent high level of initial paging.
    - E.g., If a process is suspended, keep list of pages in working set and bring entire working set back before restarting process.
    - Tradeoff - page fault vs. prepaging - depends on how many pages brought back are reused.
- Page Size Selection
  - fragmentation
  - table size
  - I/O overhead
  - locality

Demand Paging Issues

- Program Structure
  - Array A[1024,1024] of integers
  - Assume each row is stored on one page
  - Assume only one frame in memory
  - Program 1

```
for j := 1 to 1024 do
for i := 1 to 1024 do
        A[i,j] := 0;
```

How many page faults?

Demand Paging Issues

- Program Structure
  - Array A[1024,1024] of integers
  - Assume each row is stored on one page
  - Assume only one frame in memory
  - Program 1

    ```
    for j := 1 to 1024 do
    for i := 1 to 1024 do
        A[i,j] := 0;
    ```

    1024 * 1024 page faults
  - Program 2

    ```
    for i := 1 to 1024 do
    for j:= 1 to 1024 do
        A[i,j] := 0;
    ```

    How many page faults?

Demand Paging Issues

- Program Structure
  - Array A[1024,1024] of integers
  - Assume each row is stored on one page
  - Assume only one frame in memory
  - Program 1

    ```
    for j := 1 to 1024 do
    for i := 1 to 1024 do
        A[i,j] := 0;
    ```

    1024 * 1024 page faults
  - Program 2

    ```
    for i := 1 to 1024 do
    for j:= 1 to 1024 do
        A[i,j] := 0;
    ```

    1024 page faults

Demand Paging Issues

- I/O (DMA) considerations
  - Process A issues I/O request, which requires DMA
  - CPU is given to other processes
  - Page faults occur - process A's pages are paged out.
  - DMA now tries to occur - but frame is being used for another process.
- Solution 1: never do DMA to process memory - DMA takes place in kernel memory, which is never paged out. Copying Overhead!!
- Solution 2: Lock/pin pages in memory - cannot be selected for replacement.

Demand Segmentation

- Used when segmentation is used.
- OS allocates memory in segments, which it keeps track of through segment tables.
  - Segment table contains valid bit to indicate whether the segment is currently in memory.
    - If segment is in main memory, access continues.
    - If not in memory, segment fault is triggered. Then segment is then brought to memory if access is legitimate.