CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Fall 2016 Lecture 33

Virtual Memory

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

FAQ

- How does the virtual memory respond when the stack or heap grows? Or is the hole between them always there?
  Hole is always there in virtual memory. Frames in memory allocated as needed.
- How is the TLB affected when a page is moved from Memory to Disk, and is replaced by a page brought in from Disk? TLB is cache, has mechanism for removing and adding info to it.
- When does a TLB need to be flushed completely? Context switch
- Can more than one page loaded into memory when a process starts? prefetching
- Why are disk addresses of non-resident pages not stored in the page table? Generally contains only information used on page hits.

Is CS Graduate Program Right For You?

- What better career opportunities may a CS grad program provide?
- What is life like as a CS graduate student?
- What opportunities are there for paying for grad school without taking out more loans?
- Get answers from industry representatives, current graduate students and faculty

Join us

CS GRADUATE RECRUITMENT EVENT
WEDNESDAY, NOVEMBER 16, 2016
5:00 PM – 7:00 PM
Rm 130 CSB

AND PIZZA

Page Replacement Algorithms

- **Page-replacement algorithm**
  - Which frames to replace
  - 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
  - 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
- In all our examples, we use 3 frames and the **reference string** of referenced page numbers is

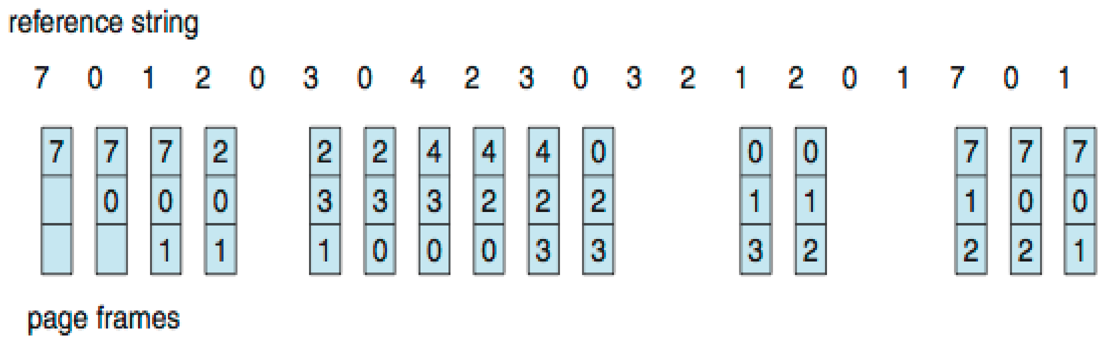
  7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1

FIFO page replacement algorithm: Out with the old; in with the new

- When a page must be replaced
  - Replace the oldest one
- OS maintains list of all pages currently in memory
  - Page at head of the list: Oldest one
  - Page at the tail: Recent arrival
- During a page fault
  - Page at the head is removed
  - New page added to the tail

First-In-First-Out (FIFO) Algorithm

- Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
- 3 frames (3 pages can be in memory at a time per



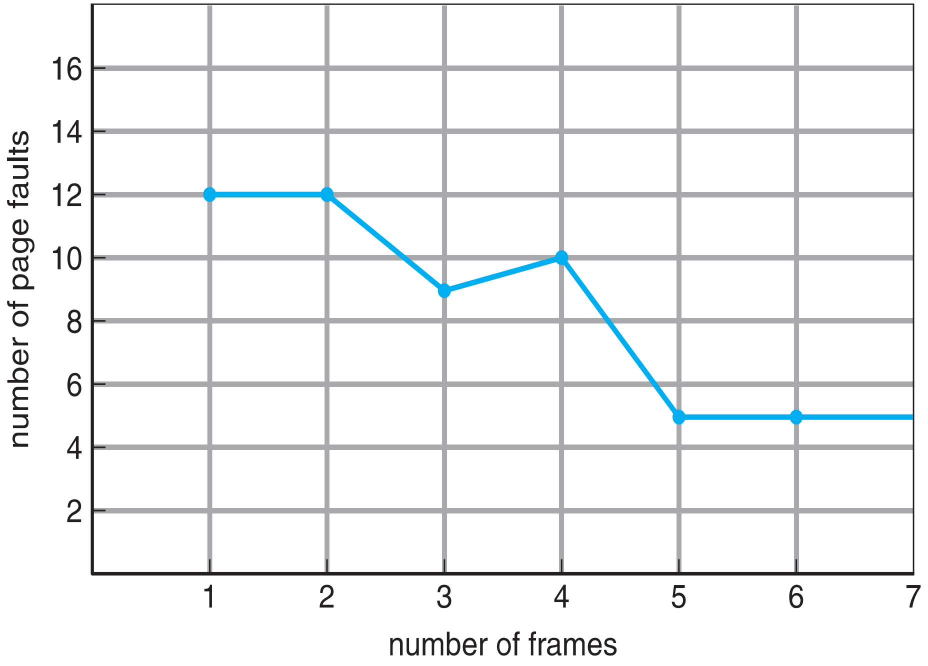
- 15 page faults
- Sometimes a page is needed soon after replacement 7,0,1,2,0,3,0, ..

Colorado State University

Belady’s Anomaly

- Consider Page reference string 1,2,3,4,1,2,5,1,2,3,4,5
  - 3 frames, 9 faults, 4 frames 10 faults!
  - Adding more frames can cause more page faults!
    - **Belady’s Anomaly**

Belady was here at CSU. Guest in my CS530!

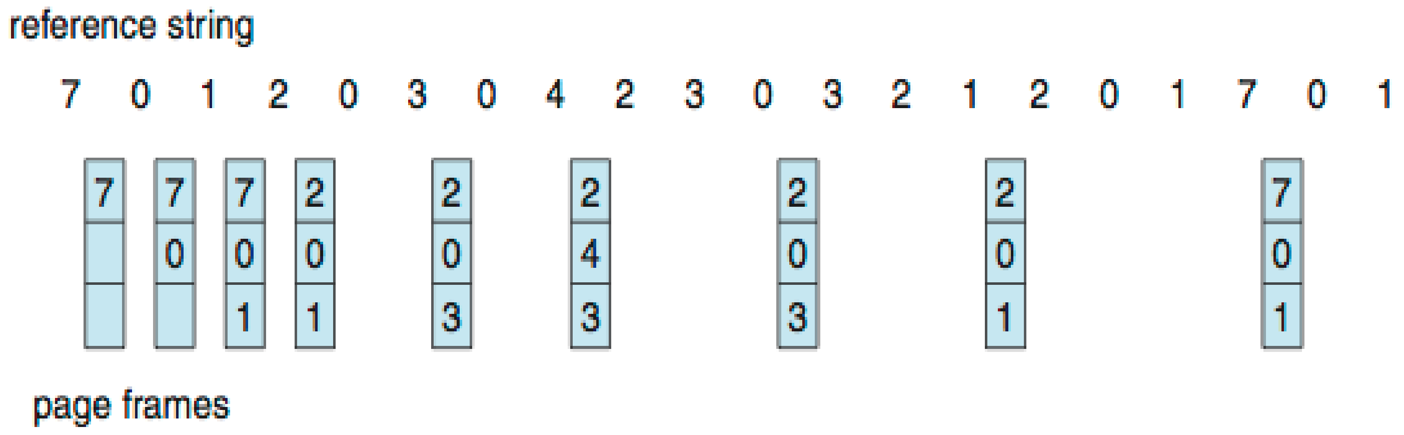


3 frames: 9 page faults
4 frames: 10 page faults

Colorado State University

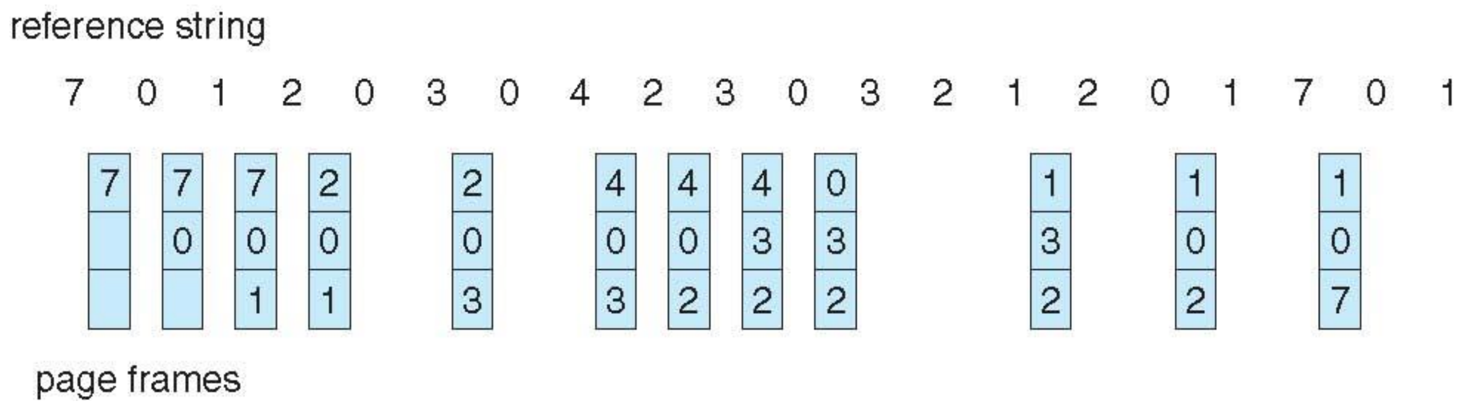
"Optimal" Algorithm

- Replace page that will not be used for longest period of time
  - 9 page replacements is optimal for the example
    - 4th access: replace 7 because we will not use if got the longest time…
- But how do we know this?
  - Can't read the future
- Used for *measuring* how well an algorithm performs



Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time (4th access – page 7 is least recently used …_)
- Associate time of last use with each page



- 12 faults – better than FIFO (15) but worse than OPT (9)
- Generally good algorithm and frequently used
- But how to implement it by tracking the page usage?

Colorado State University

LRU Algorithm: Implementations

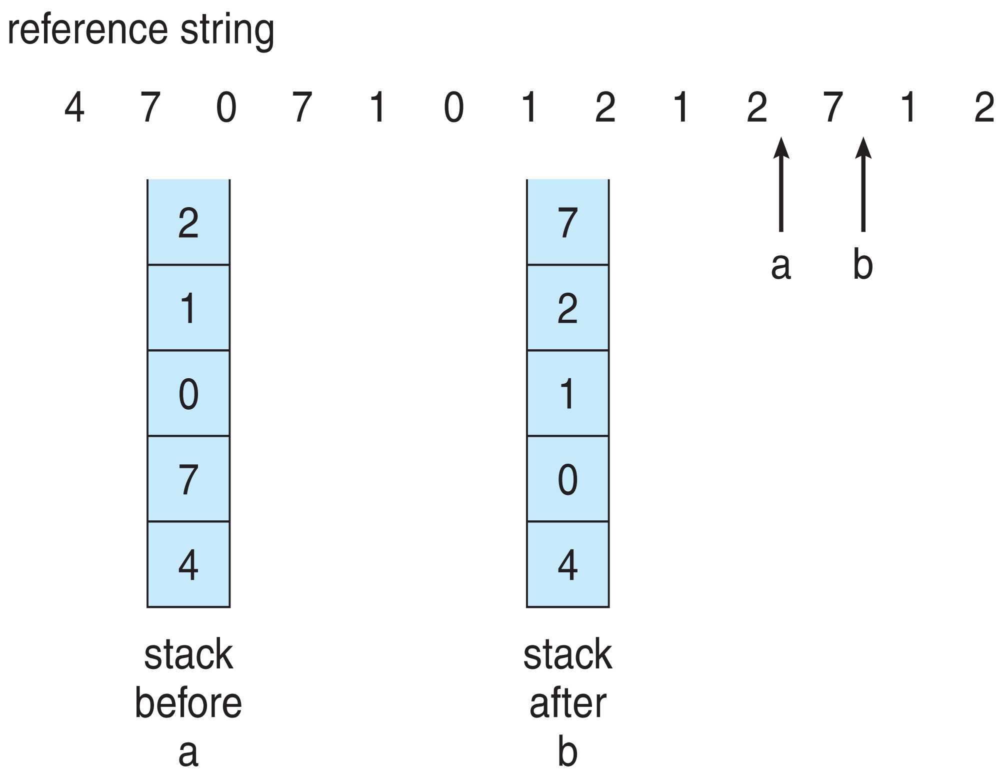
Possible implementations

- 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 changed, look at the counters to find smallest value
    - Search through table needed
- Stack implementation
  - Keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - Each update expensive
  - No search for replacement needed (bottom is least recently used)

LRU and OPT are cases of stack algorithms that don’t have Belady’s Anomaly

Colorado State University

Use Of A Stack to Record Most Recent Page References



Too slow if done in software

LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference bit**
  - With each page associate a bit, initially = 0
  - When page is referenced bit set to 1
  - Replace any with reference bit = 0 (if one exists)
    - 0 implies not used since initialization
    - We do not know the order, however.
- Advanced schemes using more bits: preserve more information about the order

Ref bit + history shift register

LRU approximation

Ref bit: indicates used

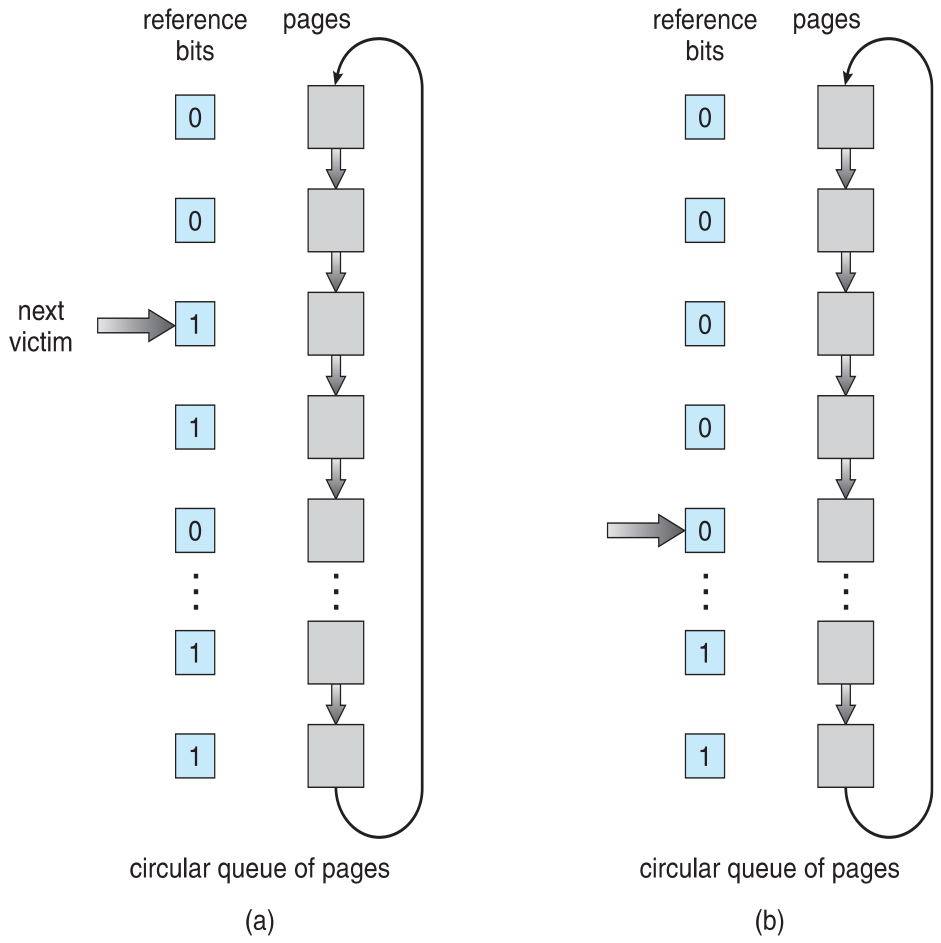
Ref Bit	Shift Register	Shift Register after OS timer interrupt
1	0000 0000	1000 0000
1	1001 0001	1100 1000
0	0110 0011	0011 0001

- Interpret 8-bit bytes as **unsigned integers**
- Page with the lowest number is the LRU page: replace. Example:
    - 00000000 : Not used in last 8 periods
    - 01100101 : Used 4 times in the last 8 periods
    - 11000100 used more recently than 01110111

LRU Approximation Algorithms

- **Second-chance algorithm**
  - Generally FIFO, plus hardware-provided reference bit
  - Avoid throwing out a heavily used page
  - **Clock** replacement (using circular queue): hand as a pointer
  - Consider next page
    - Reference bit = 0 -> replace it
    - reference bit = 1 then:
      - set reference bit 0, leave page in memory
      - consider next page, subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



- **Clock** replacement: hand as a pointer
- Consider next page
  - Reference bit = 0 -> replace it
  - reference bit = 1 then:
    - set reference bit 0, leave page in memory
    - consider next page, subject to same rules
  - (a) change to 0
  - (b) replace page

Colorado State University

Enhanced Second-Chance Algorithm

- Improve algorithm by using reference bit and modify bit (if available) in concert clean page: better candidate
- Take ordered pair (reference, modify)

1. (0, 0) neither recently used not modified – best page to replace
2. (0, 1) not recently used but modified – not quite as good, must write out before replacement
3. (1, 0) recently used but clean – probably will be used again soon
4. (1, 1) recently used and modified – probably will be used again soon and need to write out before replacement

- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
  – Might need to search circular queue several times

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
  - Not common
- **Least Frequently Used** (**LFU**) **Algorithm**: replaces page with smallest count
- **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

Clever Techniques: Page-Buffering Algorithms

- Keep a buffer (pool) of free frames, always
  - Then frame available when needed, not found at fault time
  - Read page into free frame and select victim to evict and add to free pool
  - When convenient, evict victim
- Keep list of modified pages
  - When backing store is otherwise idle, write pages there and set to non-dirty (being proactive!)
- Keep free frame previous contents intact and note what is in them
  - If referenced again before reused, no need to load contents again from disk
  - Generally useful to reduce penalty if wrong victim frame selected

Buffering and applications

- Applications often understand their memory/disk usage better than the OS
  - Provide their own buffering schemes
- If both the OS and the application were to buffer
  - Twice the I/O is being utilized for a given I/O

Allocation of Frames

Allocation of Frames

How to allocate frames to processes?

- Each process needs ***minimum*** number of frames
  - Depending on specific needs of the process
- ***Maximum*** of course is total frames in the system

- Two major allocation schemes
  - fixed allocation
  - priority allocation
- Many variations

Colorado State University

Fixed Allocation

- **Equal allocation** – For example, if there are 100 frames (after allocating frames for the OS) and 5 processes, give each process 20 frames
  - Keep some as free frame buffer pool
- **Proportional allocation** – Allocate according to the size of process
  - Dynamic as degree of multiprogramming, process sizes change

$s_i = \text{size of process } p_i$

$S = \sum s_i$

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

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

$m = 64$

$s_1 = 10$

$s_2 = 127$

$a_1 = \frac{10}{137}$ ´ 62 » 4

$a_2 = \frac{127}{137}$ ´ 62 » 57

Colorado State University

Priority Allocation

- Use a proportional allocation scheme using priorities rather than size
- If process $\boldsymbol{P_i}$ generates a page fault,
  - select for replacement one of its frames or
  - select for replacement a frame from a process with lower priority number

Global vs. Local Allocation

- **Global replacement** – process selects a replacement frame from the set of all frames; one process can take a frame from another
  - But then process execution time can vary greatly
  - But greater throughput, so more common

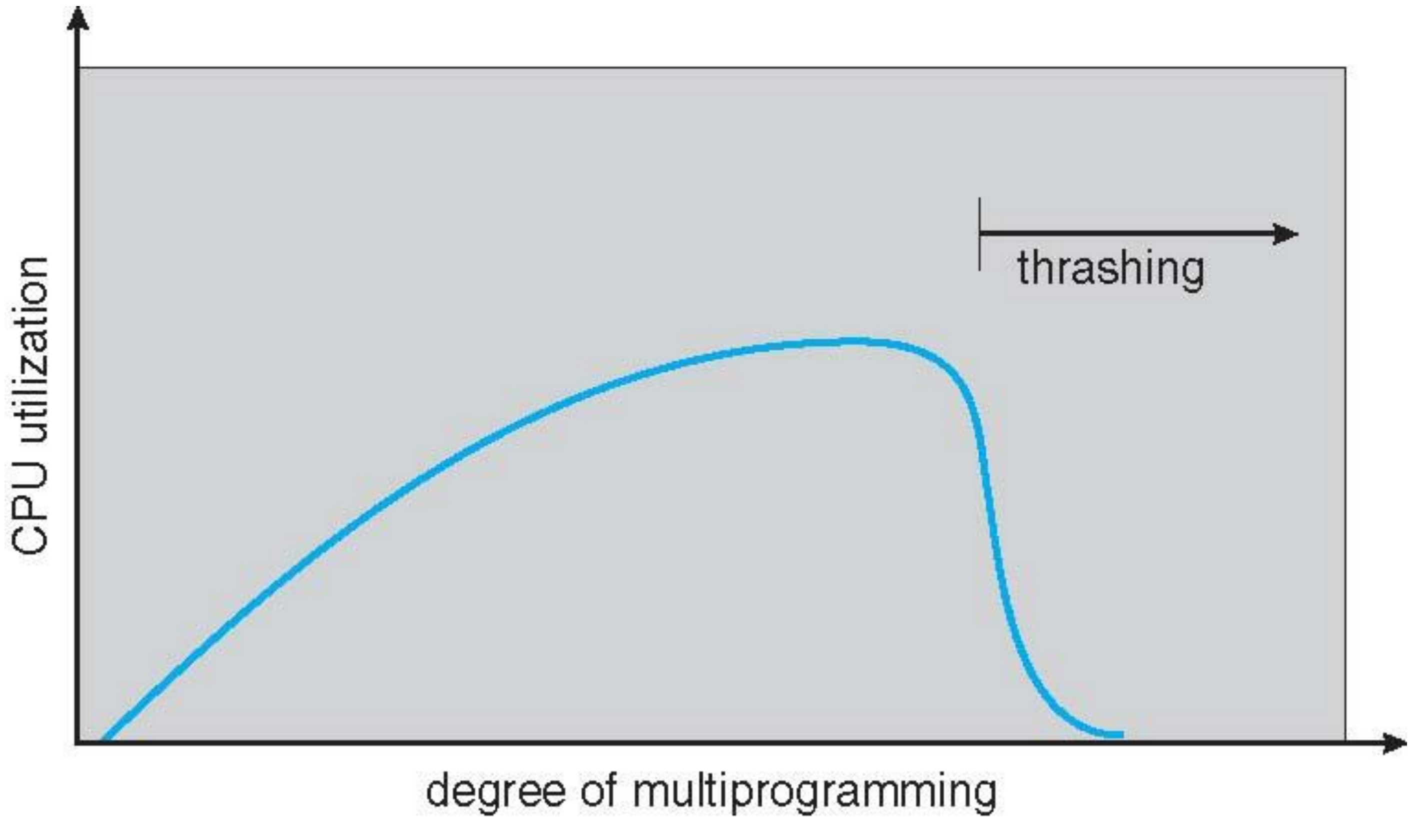
- **Local replacement** – each process selects from only its own set of allocated frames
  - More consistent per-process performance
  - But possibly underutilized memory

Thrashing

- If a process does not have "enough" pages, the page-fault rate is very high
  - Page fault to get page
  - Replace existing frame
  - But quickly need replaced frame back
  - This leads to:
    - Low CPU utilization
    - Operating system thinking 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

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
  - Limit effects by using local or priority page replacement