Announcements

- Project #4 teams emailed out
- Reminder about re-grade deadline
  - Tuesday March 29^{th} (11:00 AM)
- Reading Chapter 10 (in 8^{th} Ed)

Access Large Memory

- Problem:
  - Even with Super pages, limited TLB reach
- Solution:
  - Add one extra large segment in addition to VM
  - Can be any sized contiguous region of memory
  - Can map into any part of a processes address space
  - Consists of three fields:
    - Virtual base (starting addr in virtual memory, page aligned)
    - Physical base (starting addr in physical memory, page aligned)
    - Length (in multiple of machine's page size)
  - Hardware always consults this mapping regardless of TLB

Page Replacement Algorithms

- FIFO
  - Replace the page that was brought in longest ago
  - However
    - old pages may be great pages (frequently used)
    - number of page faults may increase when one increases number of page frames (discouraging!)
      - called belady's anomaly
      - 1,2,3,4,1,2,5,1,2,3,4,5 (consider 3 vs. 4 frames)
- Optimal
  - Replace the page that will be used furthest in the future
  - Good algorithm(!) but requires knowledge of the future
  - With good compiler assistance, knowledge of the future is sometimes possible

Page Replacement Algorithms

- LRU
  – Replace the page that was actually used longest ago
  – Implementation of LRU can be a bit expensive
    - e.g. maintain a stack of nodes representing pages and put page on top of stack when the page is accessed
    - maintain a time stamp associated with each page
- Approximate LRU algorithms
  – maintain reference bit(s) which are set whenever a page is used
  – at the end of a given time period, reference bits are cleared

FIFO Example (3 frames)

- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1) fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (2,3,4) fault, replacement
  - access 1 - (3,4,1) fault, replacement
  - access 2 - (4,1,2) fault, replacement
  - access 5 - (1,2,5) fault, replacement
  - access 1- (1,2,5)
  - access 2 - (1,2,5)
  - access 3 - (2,5,3) fault, replacement
  - access 4 - (5,3,4) fault, replacement
  - access 5 - (5,3,4)
- 9 page faults

FIFO Example (4 frames)

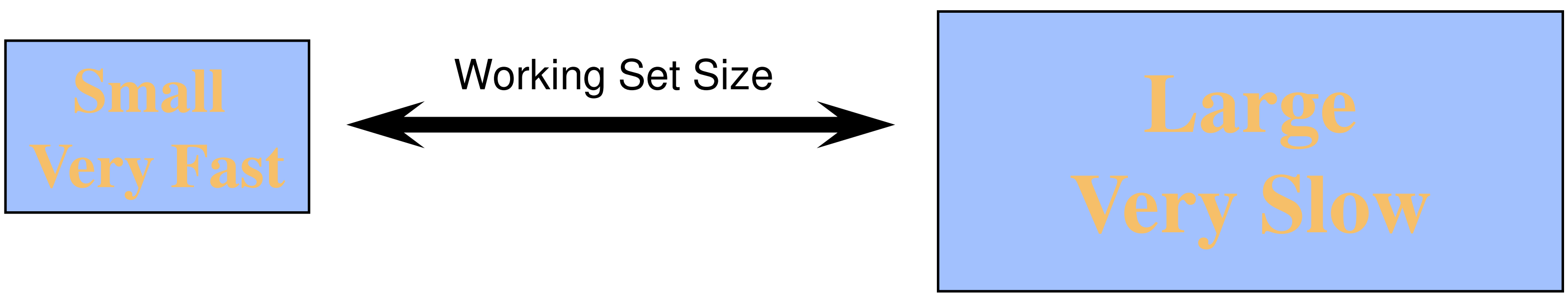
- Reference string: 1,2,3,4,1,2,5,1,2,3,4,5
  - access 1 - (1)   fault
  - access 2 - (1,2) fault
  - access 3- (1,2,3) fault
  - access 4 - (1,2,3,4) fault, replacement
  - access 1 - (1,2,3,4)
  - access 2 - (1,2,3,4)
  - access 5 - (2,3,4,5) fault, replacement
  - access 1- (3,4,5,1) fault, replacement
  - access 2 - (4,5,1,2) fault, replacement
  - access 3 - (5,1,2,3) fault, replacement
  - access 4 - (1,2,3,4) fault, replacement
  - access 5 - (2,3,4,5) fault, replacement
- 10 Page faults

Thrashing

- Virtual memory is not “free”
  - can allocate so much virtual memory that the system spends all its time getting pages
  - the situation is called thrashing
  - need to select one or more processes to swap out
- Swapping
  - write all of the memory of a process out to disk
  - don’t run the process for a period of time
  - part of medium term scheduling
- How do we know when we are thrashing?
  - check CPU utilization?
  - check paging rate?
  - Answer: need to look at both
    - low CPU utilization plus high paging rate --> thrashing

Working Sets and Page Replacement

- Programs usually display reference locality
  - temporal locality
    - repeated access to the same memory location
  - spatial locality
    - consecutive memory locations access nearby memory locations
  - memory hierarchy design relies heavily on locality reference
    - sequence of nested storage media
- Working set
  - set of pages referenced in the last delta references



Improving Heap Locality

- Malloc (or new) don't ensure locality among requests
  - Two calls to malloc could get memory on different cache lines, pages, etc.
- Option 1:
  - Malloc a large chunk of memory and parcel it out yourself
- Option 2:
  - Add a "near" hint parameter to malloc
  - Indicates that memory should be allocated near the target location
    - It's only a performance hint, and malloc can ignore it
    - Allows locality improvement without major changes

Preventing Thrashing

- Need to ensure that we can keep the working set in memory
  - if the working sets of the processes in memory exceed total page frames, then we need to swap a process out
- How do we compute the working set?
  - can approximate it using a reference bit

Implementation Issues

- How big should a page be?
  - want to trade cost of fault vs. fragmentation
    - cost of fault is: trap + seek + latency + transfer
  - Does the OS page size have to equal the HW page size?
    - no, just needs to be a multiple of it
- How does I/O relate to paging
  - if we request I/O for a process, need to lock the page
    - if not, the I/O device can overwrite the page
- Can the kernel be paged?
  - most of it can be.
  - what about the code for the page fault handler?