Virtual Memory: Load Control & Performance

Steve Goddard
goddard@cse.unl.edu

http://www.cse.unl.edu/~goddard/Courses/CSCE451

Virtual Memory Management

Fundamental Issues

- Placement strategies
- Load control strategies
- Replacement strategies
- When & how much of a process's virtual memory to load into physical memory, or how and when to set the multiprogramming level
Load Control
Fundamental tradeoff

- High multiprogramming level
  » \( MPL = \frac{f}{k} \)
  ✦ \( f \) is number of page frames
  ✦ \( k \) is the minimum number of pages required for a process to execute

- Low paging overhead
  » 1 process

Issues
- What criterion should be used to determine when to increase or decrease the \( MPL \)?
- Which task should be swapped out if the \( MPL \) must be reduced?

Load Control
How not to do it!

- Operating system increases \( MPL \)
  » New processes fault, taking memory away from existing processes

- CPU utilization falls
  » CPU utilization goes to 0, so...

- System is thrashing — spending all of its time paging
Load Control

Thrashing

- Can be ameliorated by *local* page replacement
- Better criteria for load control
  - Adjust MPL so that
    - mean time between page faults = page fault service time
    - \( \sum |w_S| = \text{size of memory} \)
- Which process should be swapped out?

Design of Paging Systems

Key design issues

- Global v. local page replacement
  - Global
    - gives better system throughput by admitting a higher MPL level
    - can lead to thrashing (programs cannot determine their page fault rate)
  - Local
    - opposite of above
- Prepping
  - Load many pages at once
  - A “win” if \( \text{prepping cost} < \text{cost of handling separate faults} \)
  - If \( p \) is the number of pages prepped and \( \alpha \) is the fraction of pages used then you potentially save \( \alpha p \) page faults
  - but “lose” the cost of \( (1 - \alpha)p \) page transfers
Design of Paging Systems

Choice of page size

◆ Small pages
  + less fragmentation, better memory utilization
    – large page tables, higher fault handling overhead
      Example: a 32-bit virtual address space with 512 byte pages
      Page table has $2^{32-9} = 8,388,608$ entries, requiring 16-32 MB of memory per process!

◆ Which page size...
  » maximizes disk performance?
  » minimizes page fault rate?
  » is motivated by good locality?

I/O Interlock

◆ To support I/O there is often a lock bit for each page table entry

◆ Example — DMA
  » Assume global page replacement
  » A process blocked on an I/O operation appears to be an ideal candidate for replacement
  » If replaced, however, I/O operation can corrupt system

◆ Solution: either
  » Lock pages in memory
  » Perform all I/O into and out of system space