More on Virtual Memory

CS 256/456

Dept. of Computer Science, University of Rochester

Recap of the Last Class

- **Virtual memory** – separation of user logical memory from physical memory.
  - Transparent page sharing:
    - Copy-on-write: allows for more efficient process creation.
  - Only part of the program address space needs to be in physical memory for execution:
    - Demand paging.
    - Memory-mapped I/O.
- Page replacement algorithm: the algorithm that picks the victim page.
  - FIFO, Optimal, LRU.

Implementations of Page Replacement Algorithms

- FIFO implementation.
- LRU implementations:
  - Time-of-use implementation
  - Stack implementation
- What needs to be done at each memory reference?
- What needs to be done at page loading or page replacement?

LRU Approximation Algorithms

- LRU approximation with a little help from the hardware.
- Reference bit
  - With each page associate a bit, initially = 0
  - When page is referenced, the bit is set to 1 by the hardware.
  - Replace a page whose reference bit is 0 (if one exists). We do not know the order, however.
- Second chance
  - Combining the reference bit with FIFO replacement
  - If page to be replaced (in FIFO order) has reference bit = 1. then:
    - set reference bit 0.
    - leave page in memory.
    - replace next page (in FIFO order), subject to same rules.
  - Also called CLOCK algorithm

LRU Approximation Algorithms

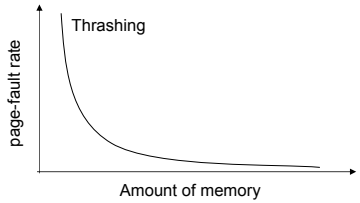
- Enhancing the reference bit algorithm:
  - it would be nice if there is more information about the reference history than a single bit.
  - with some more help from software, e.g., a memory reference counter (in page table entry and TLB)
- Maintain more reference bits in software:
  - at every N-th clock interrupt, the OS moves each hardware page reference bit (in page table entry and TLB) into a multi-bit page reference history word (in software-maintained memory).

Counting-based Page Replacement

- Least frequently used page-replacement algorithm
  - the page with smallest access count (within a period of time) is replaced
- Implementation?

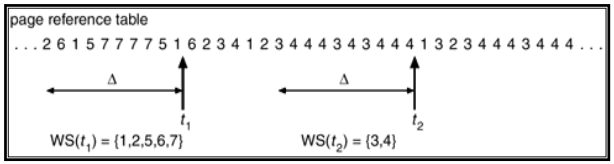
How much memory does a process need?

- Our discussion so far is "Given the amount of memory, what order should we evict pages?"
- Now we look at "How much memory does a process need?"
- If a process does not have "enough" pages, the page-fault rate is very high.
  - **Thrashing** ≡ a process is mostly busy with swapping pages.



Working-Set Model

- WSS_i (working set of Process P_i) = total number of pages referenced in the most recent Δ (working-set window).



- data access locality:
  - working set does not change or changes very slowly over time.
  - so enough memory for the working set should be good.
- How to choose Δ?

Working-Set-Based Memory Allocation

- Two components
- How much memory does a process need?
  - try to allocate enough frames for each process's working set.
  - if ΣWSS_i > m, then suspend one of the processes.
  - How to determine the working set size over a recent period Δ?
- Given the amount of memory, what order should we evict pages?
  - LRU and augment (WSClock)

Pitfall of Working-Set-Based Memory Allocation

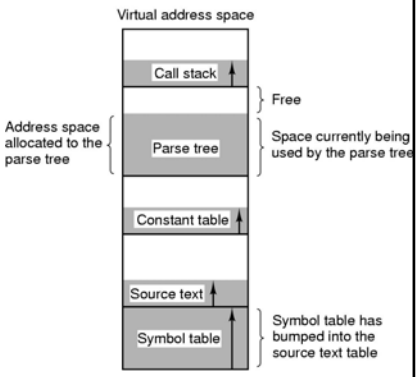
- Pitfall:
  - The working set size is not a good indicator of how much memory a process "actually" needs.
- Example:
  - Consider a process that accesses a large amount of data over time but rarely reuses any of them (e.g., sequential scan).
  - It would exhibit a large working set but different memory sizes would not significantly affect its page fault rate.

Other Memory Management Issues

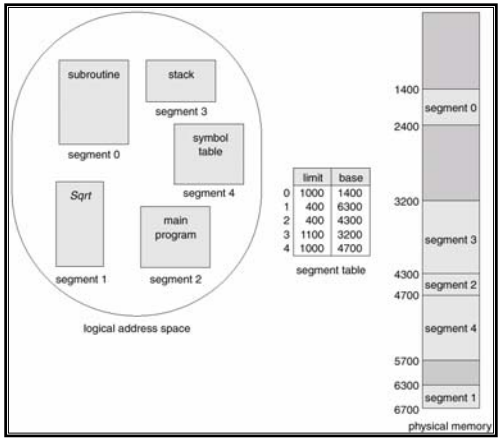
- When to swap out pages?
- Prepaging
  - swap in pages that are expected to be accessed in the future

Segmentation

- One-dimensional address space with growing pieces
- At compile time, one table may bump into another
- Segmentation:
  - generate segmented logical address at compile time
  - segmented logical address is translated into physical address at execution time

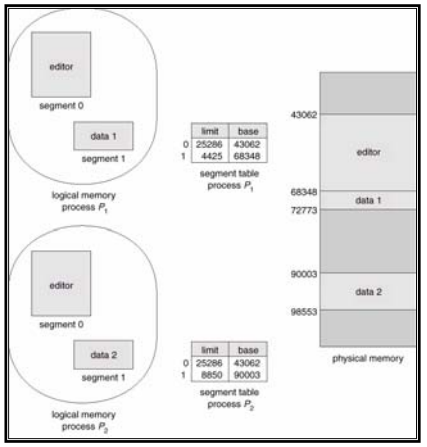


Example of Segmentation



Sharing of Segments

- Convenient sharing of libraries



Disclaimer

- Parts of the lecture slides contain original work of Abraham Silberschatz, Peter B. Galvin, Greg Gagne, Andrew S. Tanenbaum, and Gary Nutt. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).