Slide 1

Administrivia

- (None?)

Slide 2

Paging — Recap

- Recall basic ideas of paging:
  - Divide address spaces into pages, memory into page frames; allocate memory page (frame) by page (frame).
  - Use page tables (one per process) to keep track of things.
  - Use MMU to translate program (virtual) addresses into memory locations — using page table for current process. Generate “page fault” interrupt if impossible.
- Some issues — performance, what to do about large page tables — but solvable.

Slide 3

Paging and Virtual Memory

- Idea — if we don't have room for all pages of all processes in main memory, keep some on disk ("pretend we have more memory than we really do").
- Or a simpler view: All address spaces live in secondary memory / swap space / backing store, and we "page in" as needed (demand paging).
- Making this work requires help from both hardware (MMU) and software (operating system).

Slide 4

Processing Memory References — MMU

- Does cache contain data for (virtual) address? If so, done.
- Does TLB contain matching page table entry? If so, generate physical address and send to memory bus.
- Does page table entry (in memory) say page is present? If so, put PTE in TLB and as above.
- If page table entry says page not present, generate page fault interrupt. Transfers control to interrupt handler.

Slide 5

Processing Memory References — Page Fault Interrupt Handler

- Is page on disk or invalid (based on entry in process table, or other o/s data structure)? If invalid, error — terminate process.
- Is there a free page frame? If not, choose one to steal. If it needs to be saved to disk, start I/O to do that. Update process table, PTE, etc., for “victim” process. Block process until I/O done.
- Start I/O to bring needed page in from swap space (or zero out new page). If I/O needed, block process until done.
- Update process table, etc., for process that caused the page fault, and restart it at instruction that generated page fault.

Slide 6

Processing Memory References — Details Still To Fill In

- How to keep track of pages on disk.
- How to keep track of which page frames are free.
- How to “schedule I/O” (but that’s later).
- How to choose a page frame to “steal”.

Slide 7

Keeping Track of Pages on Disk

- To implement virtual memory, need space on disk to keep pages not in main memory. Reserve part of disk for this purpose (“swap space”); (conceptually) divide it into page-sized chunks. How to keep track of which pages are where?
- One approach — give each process a contiguous piece of swap space. Advantages/disadvantages?
- Another approach — assign chunks of swap space individually. Advantages/disadvantages?
- Either way — processes must know where “their” pages are (via page table and some other data structure), operating system must know where free slots are (in memory and in swap space).

Slide 8

Finding A Free Frame — Page Replacement Algorithms

- Processing a page fault can involve finding a free page frame. Would be easy if the current set of processes aren’t taking up all of main memory, but what if they are? Must steal a page frame from someone. How to choose one?
- Several ways to make choice (as with CPU scheduling) — “page replacement algorithms”.
- “Good” algorithms are those that result in few page faults. (What happens if there are many page faults?)
- Choice usually constrained by what MMU provides (though that is influenced by what would help o/s designers).

Processing Memory References — Hardware vs. Software

Slide 9

- Some things defined by hardware architecture — structure of page table entries, how MMU finds page table.
- A very common feature — each entry has R (“referenced”) and M (“modified”) bits.

  Set by MMU on every memory reference.

  Cleared by operating system “when appropriate” — M bit when page is replaced or written to disk, R bit when? Often want to do this periodically. A good choice is “on clock interrupts” (generated at intervals by hardware, gives o/s regular opportunities to do many things — more in chapter 5).

Minute Essay

Slide 10

- In class I told a story: Once upon a time, a mainframe computer was running very slowly. The sysadmins were puzzled, until one of them noticed that one of the disk drives seemed to be very busy and asked “which disk are you using for paging?” The answer made everyone say “aha!” What was wrong (to make the system so slow)?

Minute Essay Answer

- The disk being used for paging was the one that was very busy. So, mostly likely the system was spending so much time paging (“thrashing”) that it wasn’t able to get anything else done. Usually this means that the system isn’t able to keep up with active processes’ demand for memory.