

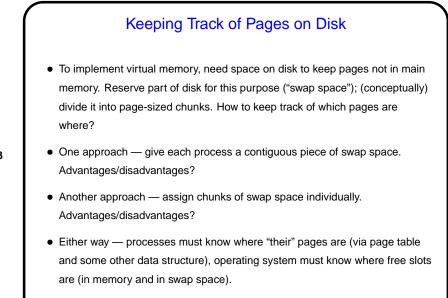
Paging — Recap
Recall basic ideas of paging:

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.

Slide 2

Use MMU to translate program (virtual) addresses into memory locations — using page table for current process. Generate "page fault" interrupt if impossible.
Last time we talked about how to resolve some potential problems (unmanageably large page tables, performance problems).
Still to discuss:

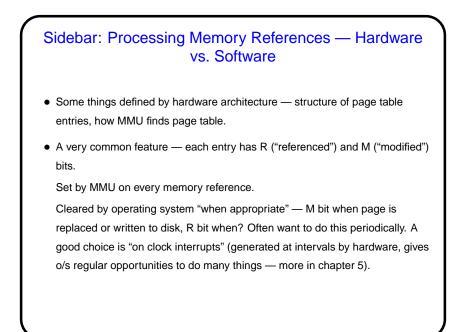
How to keep track of pages on disk.
How to choose a page frame to "steal".



Finding A Free Frame — Page Replacement Algorithms

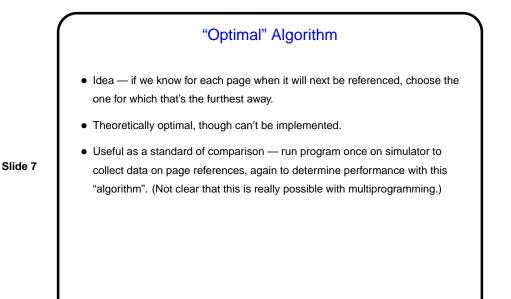
- Processing a page fault can involve finding a free page frame. Would be easy
 if the current set of processes aren't taking up all of main memory, but what if
 they are? Must steal a page frame from someone. How to choose one?
- Several ways to make choice (as with CPU scheduling) "page replacement 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).
 - Many choices ...

Slide 3



Sidebar: Page Tables, Revisited What do we need for each entry in a page table? Page frame number. Present/absent bit (was valid/invalid). Protection bit(s). "Modified since last page-in?" bit. "Referenced recently?" bit. "Okay to cache?" bit. Goal is to keep this somewhat minimal — mostly data the MMU needs. If present/absent bit says "absent", two cases — error and "page not in memory right now" — MMU should generate "page fault" interrupt, let page fault interrupt handler decide.

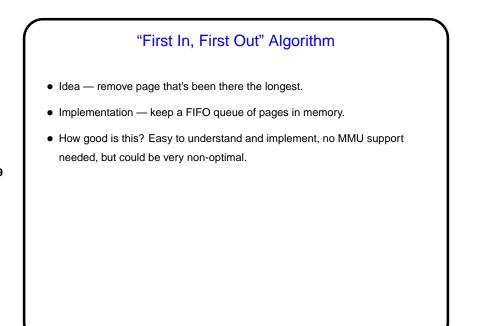
Slide 5



"Not Recently Used" Algorithm
Idea — choose a page that hasn't been referenced/modified recently, hoping it won't be referenced again soon.
Implementation — use page table's R and M bits, group pages into four classes:

R=0, M=0.
R=0, M=1.
R=1, M=0.
R=1, M=1.

Choose page to replace at random from first non-empty class.
How good is this? Easy to understand, reasonably efficient to implement, often gives adequate performance.

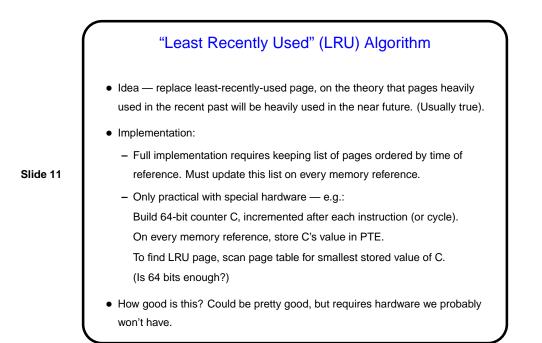


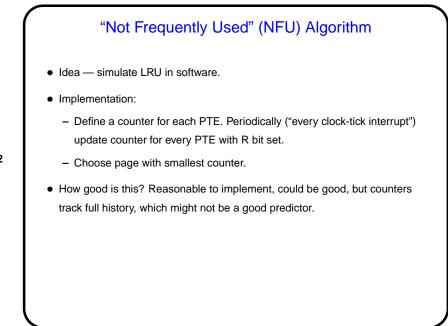
Slide 9

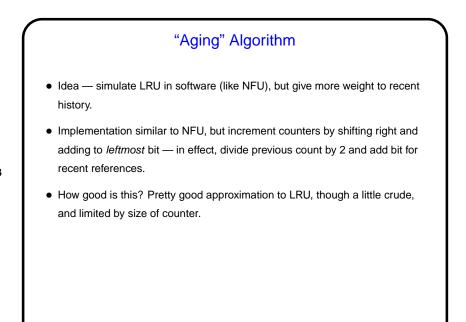
"Second Chance" Algorithm

- Idea modify FIFO algorithm so it only removes the oldest page if it looks inactive.
- Implementation use page table's R and M bits, also keep FIFO queue.
 Choose page from head of FIFO queue, *but* if its R bit is set, just clear R bit and put page back on queue.

- Variant "clock" algorithm (same idea, keeps pages in a circular queue).
- How good is this? Easy to understand and implement, probably better than FIFO.







Slide 13

Minute Essay Another story from long ago: 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)? Does anything like this still happen?

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.
- Slide 15
- This can indeed still be a problem.