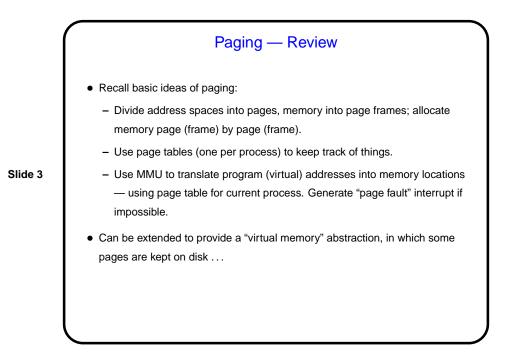


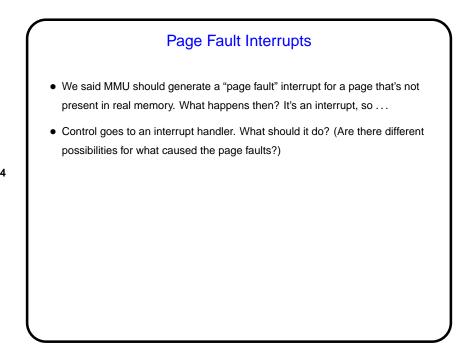
Sidebar: Memory Management Within Processes

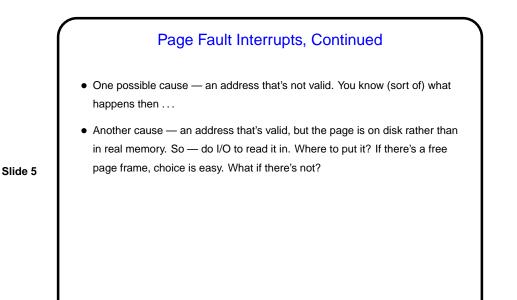
• What if we don't know before the program starts how much memory it will want? with very old languages, maybe not an issue, but with more modern ones it is.

I.e., we might want to manage memory within a process's "address space" (range of possible program/virtual addresses).

- Typical scheme involves
 - Fixed-size allocation for code and any static data.
 - Two variable-size pieces ("heap" and "stack") for dynamically allocated data.
 - Notice combined sizes of these pieces might be less than size of address space, maybe a lot less.

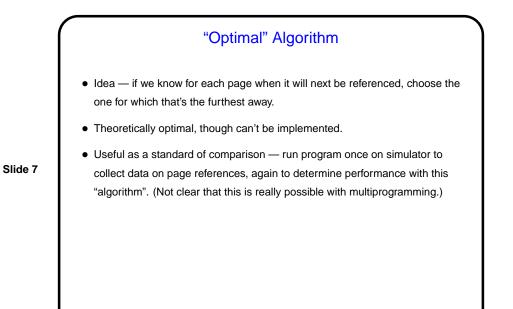




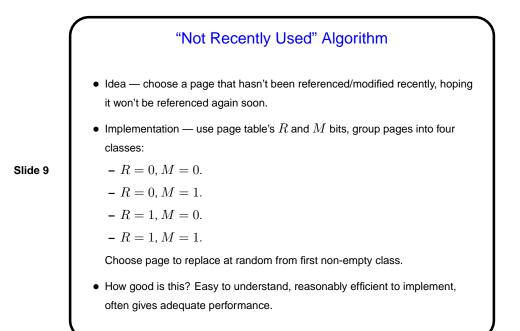


Finding A Free Frame — Page Replacement Algorithms

- 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?
- Slide 6
- 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 ...



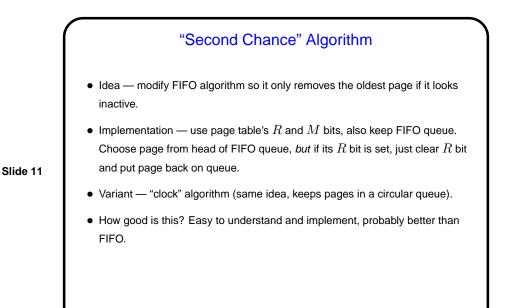
Sidebar: Page Table Entries, Revisited
Recall — many architectures' page table entries contain bits called "R (referenced) bit" and "M (modified) bit".
Idea is that these bits are set by hardware on any memory reference, and cleared by software (o/s) in some way that's useful.



"First In, First Out" Algorithm

- Idea remove page that's been there the longest.
- Implementation keep a FIFO queue of pages in memory.
- How good is this? Easy to understand and implement, no MMU support needed, but could be very non-optimal.

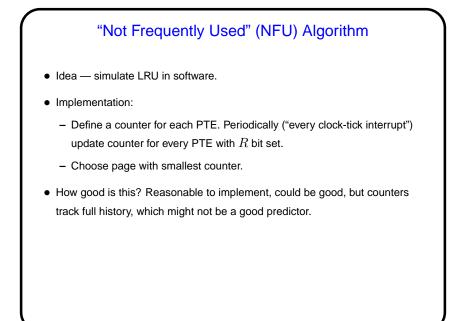
Slide 12



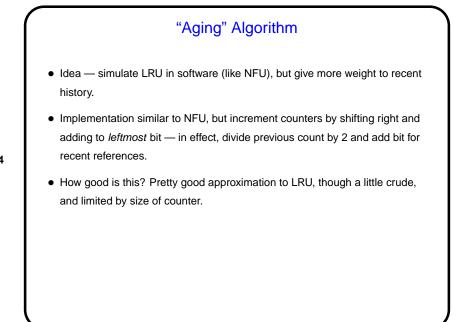
"Least Recently Used" (LRU) Algorithm
Idea — replace least-recently-used page, on the theory that pages heavily used in the recent past will be heavily used in the near future. (Usually true).
Implementation:

Full implementation requires keeping list of pages ordered by time of reference. Must update this list on every memory reference.
Only practical with special hardware — e.g.: Build 64-bit counter C, incremented after each instruction (or cycle). On every memory reference, store C's value in PTE. To find LRU page, scan page table for smallest stored value of C. (Is 64 bits enough?)

How good is this? Could be pretty good, but requires hardware we probably won't have.



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)?

Slide 15

• 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 16

• This can indeed still be a problem — only a few years ago, with the Xenas trying to run both Eclipse and a Lewis simulation.