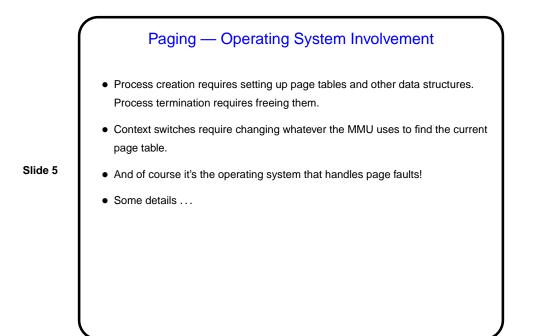
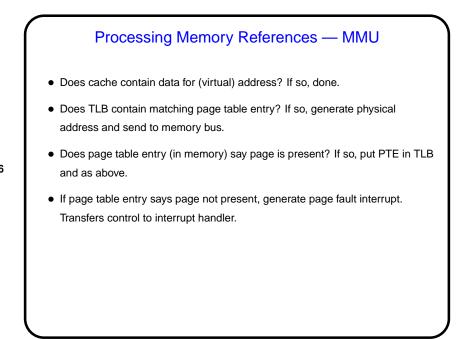


Paging — Operating System Versus MMU

• Some aspects of paging are dealt with by hardware (MMU) — translation of program addresses to physical addresses, generation of page faults, setting of *R* and *M* bits.

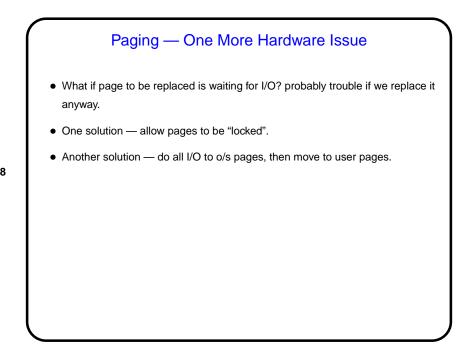
• Other aspects need o/s involvement. What/when?

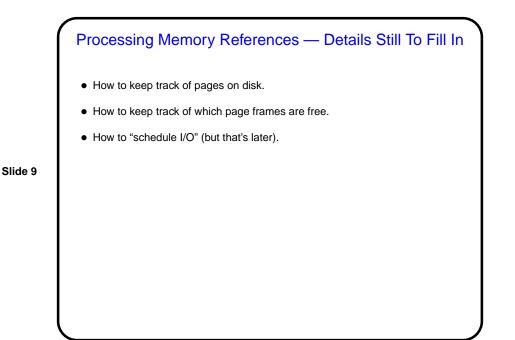






- Is page on disk or invalid (based on entry in process table, or other o/s data structure)? If invalid, error terminate process.
- Is there a free page frame? If not, choose one to steal. If it needs to be saved to disk, start I/O to do that. Update process table, PTE, etc., for "victim" process. Block 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.

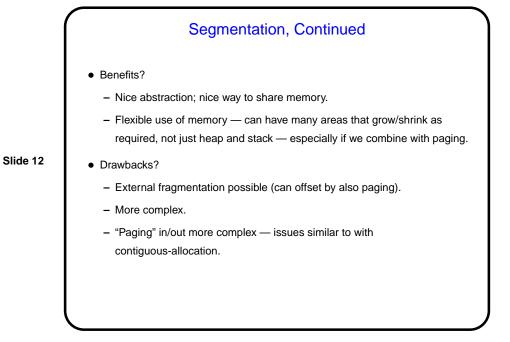


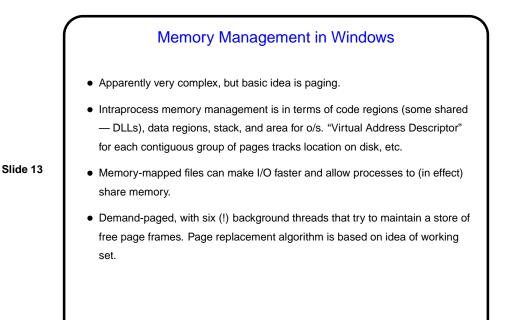


Keeping Track of Pages on Disk
To implement virtual memory, need space on disk to keep pages not in main memory. Reserve part of disk for this purpose ("swap space"); (conceptually) divide it into page-sized chunks. How to keep track of which pages are where?
One approach — give each process a contiguous 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).



- Idea make program address "two-dimensional" / separate address space into logical parts. So a virtual address has two parts, a segment and an offset.
- To map virtual address to memory location, need "segment table", like page table except each entry also requires a length/limit field. (So this is like a cross between contiguous-allocation schemes and paging.)





Memory Management in UNIX/Linux

- Very early UNIX used contiguous-allocation or segmentation with swapping. Later versions use paging. Linux uses multi-level page tables; details depend on architecture (e.g., three levels for Alpha, two for Pentium).
- Intraprocess memory management is in terms of text (code) segment, data segment, and stack segment. Linux reserves part of address space for o/s.
 For each contiguous group of pages, "vm_area_struct" tracks location on disk, etc.
 - Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
 - Demand-paged, with background process ("page daemon") that tries to maintain a store of free page frames. Page replacement algorithms are mostly variants of clock algorithm.

Minute Essay

- At least one early mainframe operating system supported virtual memory (i.e., keeping some pages on disk), but provided only a single address space that all processes shared. How does this compare with giving each process its own address space - with respect to simplicity, efficiency, anything else you can think of? (There appears to be relatively recent research into reviving this idea. Hm!)
- Anything about memory management you'd like to hear more about / have clarified?

Minute Essay Answer

• A single address space seems like it might be simpler and more efficient -could have a single page table. Also it would be easier for processes to share memory. But it seems potentially less secure - how to protect one process's memory for another? - and seems like it would make the "relocation problem" relevant again, at least in some contexts. (You may think of other considerations!)