Administrivia

- (None.)

Slide 1

Paging — Miscellaneous Issues, Recap

- Demand paging versus prepaging.
- Global versus local allocation.
- Sharing pages.

Slide 2

Slide 3

One More Design Issue

- Page replacement algorithms as discussed all seem based on the idea that we let memory fill up, and then "steal" page frames as needed. Is that really the best way . . .
- An alternative — background process ("paging daemon") that tries to keep a supply of free page frames, or at least ones that can be stolen without needing to write out their contents. Can use algorithms similar to page replacement algorithms to do this.

Slide 4

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?

Slide 5

Paging — Operating System Involvement

- Process creation requires setting up page tables and other data structures. Process termination requires freeing them.
- Context switches require changing whatever the MMU uses to find the current page table.
- And of course it's the operating system that handles page faults!
- Some details . . .

Slide 6

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 7

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 8

Paging — One More Hardware Issue

- What if page to be replaced is waiting for I/O? probably trouble if we replace it anyway.
- One solution — allow pages to be “locked”.
- Another solution — do all I/O to o/s pages, then move to user pages.

Slide 9

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).

Slide 10

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 11

One More Memory Management Strategy — Segmentation

- 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.)

Slide 12

Segmentation, Continued

- Benefits?
  - Nice abstraction; nice way to share memory.
  - Flexible use of memory — can have many areas that grow/shrink as required, not just heap and stack — especially if we combine with paging.
- Drawbacks?
  - External fragmentation possible (can offset by also paging).
  - More complex.
  - "Paging" in/out more complex — issues similar to with contiguous-allocation.

Slide 13

Memory Management in Windows

- Apparently very complex, but basic idea is paging.
- Intraprocess memory management is in terms of code regions (some shared — DLLs), data regions, stack, and area for o/s. “Virtual Address Descriptor” for each contiguous group of pages tracks location on disk, etc.
- Memory-mapped files can make I/O faster and allow processes to (in effect) share memory.
- Demand-paged, with six (!) background threads that try to maintain a store of free page frames. Page replacement algorithm is based on idea of working set.

Slide 14

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.

Slide 15

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?

Slide 16

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!)