Slide 1

Administrivia

- (None.)

Slide 2

Memory Management — Recap

- The problem we're solving — partition physical memory among processes.
- Two related issues — program relocation and memory protection. Whether program relocation is potentially a problem depends on the processor's instruction set and on the program — are there instructions that use absolute addresses, and does the program use them? (For MIPS, it appears that some forms of jump and load/store instructions could.)
- Both nicely solved by defining "address space" abstraction and implementing with help from hardware (MMU). Also makes it easier to move processes around in memory. (Why would you want to?)

Slide 3

Multiprogramming with Fixed/Variable Partitions — Recap

- Comparing the two schemes:
  - Similar admission scheduling issues.
  - Complexity versus flexibility, memory use also roughly similar.
- Either could be adequate for a simple batch system, maybe with the addition of swapping.

Slide 4

Swapping

- Idea — move processes into / out of main memory (when not in main memory, save on disk).

  (Aside — can we run a program directly from disk?)
- Addresses both questions from previous slide; could also provide a way to "fix" fragmentation.
- Implies another level of scheduling (what to swap in/out).
- Makes non-dynamic solutions to relocation problem much less attractive. MMU-based solution still works, though, and adds memory protection.
- Consider tradeoffs again — complexity versus flexibility, efficient use of memory.

Slide 5

Sidebar: Three-Level Scheduling

- Basic idea — break up problem of scheduling (batch) work into three parts:
  - Admissions scheduling — choose from input queue which jobs to “let into the system” (create processes for).
  - Memory scheduling — choose from among processes in system which to keep in memory, which to “swap out” to disk.
  - CPU scheduling — choose from among processes in memory which to actually run.
- Points to consider:
  - Are there advantages to limiting how many processes, how many in memory? What criteria could we use?
  - Are there advantages to the explicit three-level scheme?
  - Would this (or a variant) work for interactive systems?
  - Do all three schedulers have to be efficient?

Slide 6

Simple Memory Management — Recap

- Contiguous-allocation schemes are simple to understand, implement.
- But they’re not very flexible — process’s memory must be contiguous, swapping is all-or-nothing.
- Can we do better? yes, by relaxing one or both of those requirements — “paging”.

Slide 7

Paging

- Idea — divide both address spaces and memory into fixed-size blocks ("pages" and "page frames"), allow non-contiguous allocation.
- Consider tradeoffs yet again — complexity versus flexibility, efficient use of memory.

Slide 8

Paging — Mapping Program to Physical Addresses

- One consequence — mapping from program addresses to physical addresses is much more complicated.
- How? "page table" for each process maps pages of address space to page frames; use this to calculate physical address from program address.
  (Are there page sizes for which this is easier?)
- As with base/limit scheme, makes more sense to implement this in MMU. (Notice again interaction between hardware design and o/s design.)
- Could let page table size vary, but easier to make them all the same (i.e., each process has the same size address space), have a bit to indicate valid/invalid for each entry. Attempt to access page with invalid bit — "page fault".

Slide 9

Paging and Virtual Memory

- Idea — extend this scheme to provide “virtual memory” — keep some pages on disk. Allows us to pretend we have more memory than we really do.
- (Compare to swapping. Details later.)

Slide 10

Paging and Memory Protection

- This scheme also provides memory protection. (How?)
- We could also use it to allow processes to share memory. (How?)

Slide 11

Page Table Entries

- Exactly what's in a page table entry depends partly on hardware.
- Required(?) fields — page frame number, present/absent bit.
- Optional but useful fields — bits that can be used to track usage ("referenced/modified"), bits indicating what access is allowed (e.g., read-only), etc.

Slide 12

Minute Essay

- To do its job the MMU must have access to the current process's page table. The textbook mentions two simple schemes for doing this:
  - Keep the entire table in (processor) registers.
  - Keep the table in memory and have a particular processor register point to its starting location.
- What advantages/disadvantages can you think of for each of these? (Think about context switching between processes and also about how quickly the MMU will be able to translate each address.)

Minute Essay Answer

- The first scheme almost surely makes for faster translations, but for a large page table it will require a lot of registers, which would make context switches slow. The second scheme won't slow down context switches, but as stated it isn't going to make for fast translation.