Slide 1

Administrivia

- (None.)

Slide 2

Memory Management — Recap

- The problem we're solving — partition physical memory among programs (processes, really).
- 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

Memory Management with Contiguous Allocation

- Simplest MMU uses two registers, base and limit. This more or less implies that each process can have only one contiguous chunk of memory. (Notice here the interaction between hardware design and o/s design.)
- Key issues here are keeping track of what space is used by what, and deciding how to assign memory to processes.

Slide 4

Multiprogramming With Variable Partitions — Bitmaps

- One solution to problem of keeping track of locations/sizes of processes' memory and free-space "chunks".
- Idea — divide memory into "allocation units"; for each, one bit says whether it's free.
- Tradeoffs — simple? easy/quick to find free space of size N?
- How big should allocation units be? (What if they're really small? really big?)
- We've left something out here — how to keep track of processes' memory — where / how big. ?

Slide 5

Multiprogramming With Variable Partitions — Free List

- Another solution to problem of keeping track of locations/sizes of processes' memory and free-space "chunks".
- Idea — keep linked list with one entry for each process or free-space chunk ("hole"), sorted by address. When we allocate/free memory, possibly split/merge entries.
- Tradeoffs — simple? space requirements compared to bitmap?

Slide 6

Multiprogramming With Variable Partitions, Continued

- Another implementation issue — how to decide, when starting a process, which of the available free chunks to assign.
- Several strategies possible:
  - First fit.
  - Next fit.
  - Best fit.
  - Worst fit.
  - Quick fit.

Slide 7

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 8

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 9

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 10

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.

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".

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.

Slide 13

Paging and Memory Protection

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

Slide 14

Minute Essay

- None — sign in.