Slide 1

Administrivia

- Summary of grades will be mailed when I get Homework 2 graded. I based midterm grades (for the registrar) on the midterm, Homework 1, and a conservative estimate of Homework 2.

Slide 2

Memory Management, Introduction

- One job of operating system is to “manage memory” — assign sections of main memory to processes, keep track of who has what, protect processes’ memory from other processes.
- As with CPU scheduling, we’ll look at several schemes, starting with the very simple. For each scheme, think about how well it solves the problem, how it compares to others.
- As with processes, there’s a tradeoff between simplicity and providing a nice abstraction to user programs.

Slide 3

Simple Schemes — No Abstraction

- Memory (a.k.a. “RAM”) can be thought of as a very long list of numbered cells (usually bytes).
- Simplest schemes for managing it don’t try to hide that view. (Name for these come from older edition of Tanenbaum’s book.)

Slide 4

Monoprogramming

- Idea — only one user program/process at a time, no swapping or paging. Only decision to make is how much memory to devote to o/s itself, where to put it.
- Consider tradeoffs — complexity versus flexibility, efficient use of memory.
- Used in very early mainframes, MS-DOS; still used in some embedded systems.

Slide 5

Multiprogramming With Fixed Partitions

- Idea — partition memory into fixed-size “partitions” (maybe different sizes), one for each process. Possibly also add the ability to “swap” programs (write their memory to disk, read back in later).
- Limits “degree of multiprogramming” (how many processes can run concurrently).
- Probably necessitates admissions scheduling — either one input queue per partition, or one combined queue.

  If one combined queue, how to choose from it when a partition becomes available? first job that fits? largest job that fits? etc.
- Consider tradeoffs — complexity versus flexibility, efficient use of memory.
- Used in early mainframes.

Slide 6

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 7

Multiprogramming With Variable Partitions

- Idea — separate memory into partitions as before, but allow them to vary in size and number.

  I.e., “contiguous allocation” scheme.
- Like previous scheme, necessitates admissions scheduling.
- Requires that we keep track of locations and sizes of processes' partitions, free space. Notice potential for memory fragmentation.
- Consider tradeoffs — complexity versus flexibility, efficient use of memory.
- Used in early mainframes.

Slide 8

Program Relocation and Memory Protection

- At the machine-instruction level, references to memory are in terms of an absolute number. Some references are made relative to the program counter, but others may be absolute — i.e., generated when the program is translated to machine language. Compilers/assemblers can generate these only by making assumption about where program will reside in memory.
- In the very early days, all programs were loaded at address 0, so no problem. With monoprogramming, too, all programs reside at the same address, so no problem.
- What happens, though, if you want to have multiple programs in memory? compilers/assemblers can't generate correct absolute addresses, plus there's the problem of protecting each program's memory from other programs.

Slide 9

Program Relocation and Memory Protection, Continued

- One solution to the relocation problem — generate, as part of the executable, a list of locations where there's an absolute address, and modify it as the program is loaded into memory. (What implications does this have for being able to do swapping?)
- One solution to the memory-protection problem — storage-protection keys (IBM 360, an early mainframe).
- A better solution to both problems involves translating addresses "on the fly" . . .

Slide 10

Sidebar: The "Address Space" Abstraction

- Basic idea is somewhat analogous to process abstraction, in which each process has its own simulated CPU. Here, each process has its own simulated memory.
- As with processes, implementing this abstraction is part of what an operating system can/should do.
- Usually, though, o/s needs help from hardware . . .

Slide 11

Dynamic Address Translation

- Underlying idea — separate program addresses (relative to start of program's "address space") from physical addresses (memory locations), and map program addresses to physical addresses. Also try to identify out-of-bounds addresses.
- Simplest such map based on base and limit addresses (B and L):

  Program address p maps to memory location $B + p$.

  If $B + p > L$, invalid (out of bounds).

  If B and L are different for each process — solves both problems.
- Only practical way to implement — hardware "memory management unit" that logically sits between the CPU and memory.

  Simplifying, CPU references program addresses, MMU turns them into physical addresses, generates interrupt if invalid.

Slide 12

A Simple MMU

- Idea — map each process's address space to a contiguous chunk of real memory, using base and limit registers.
- Solves both the relocation and protection problems, though may not be especially fast.
- Consider tradeoffs — complexity versus flexibility.
- Used in some early mainframes and PCs.

Slide 13

Minute Essay

- How did the midterm compare to your expectations?