

Administrivia

- Reminders:
 - Homework 2 due today by 5pm. Code to be e-mailed to me at `cs4320@cs.trinity.edu`.
 - Homework 3 due Tuesday by 5pm.
 - Usually I accept homework up to a week late (syllabus says “one class period late”), at a penalty. “Not accepted past *date*” means what it says; late penalties still apply.
 - Review sheet for exam on Web. In brief — open book, open notes, chapters 1 through 3. Questions mostly similar to homework, maybe some multiple-choice, etc. Lecture slides online. No in-class review, but review session next week possible.
- A reminder from Career Services: Grad school fair tomorrow, 9:30am to 2pm on the Esplanade.

Slide 1

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.

Slide 3

Minute Essay From Last Lecture

- What questions do you have about the homeworks and exam?
(Answered in previous slide.)
- Also, a comment about a Homework 1 question:
One proposed answer to “when would you not want an operating system?” was “for better security” — well, *maybe!*

Slide 2

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 4

Multiprogramming With Fixed Partitions

- Idea — partition memory into fixed-size “partitions” (maybe different sizes), one for each process.
- 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 5

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?

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.
(We’ll consider swapping separately, unlike textbook.)
- 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 6

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 8

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 9

Relocation and Protection, Review

- The “relocation problem” — as part of compiling/linking, must turn symbolic/implicit references (e.g., to other parts of the program) into memory addresses.
 - One simple way — absolute addresses.
 - Another simple way — loader program to modify addresses as program is loaded into memory.
- Memory protection — want to protect the memory of one process from inadvertent or malicious access by other processes.
 - A simple way — divide memory into blocks, associate a key with each block, only allow access to process with matching key (or “key 0” – α/s).
- A way to solve both problems — “dynamic address translation” via MMU.

Slide 11

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 — next lecture).

Slide 10

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.
 - MMU uses, e.g., base and limit registers that change during a context switch.

Slide 12

Minute Essay

- What if anything did you find difficult about the programming part of the homework? What if anything did you find interesting/useful about it?