

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.
- Slide 6
- 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.

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.

Slide 7

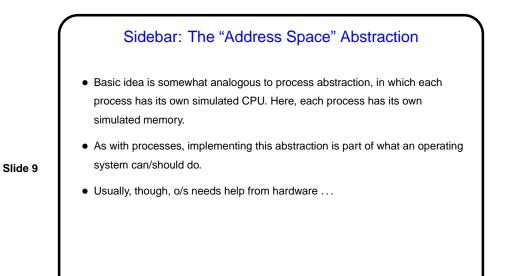
- In the very early days, all programs loaded at address 0, so no problem. With monoprogramming, too, all programs reside at the same address, so no problem.
- What happens, though, if you want to have multiple programs in memory? compilers/assemblers can't generate correct absolute addresses. Also, we want to protect each process's memory from other processes.

Program Relocation and Memory Protection, Continued

 One solution to 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?)

Slide 8

- 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" . . .



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.

Slide 10

 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

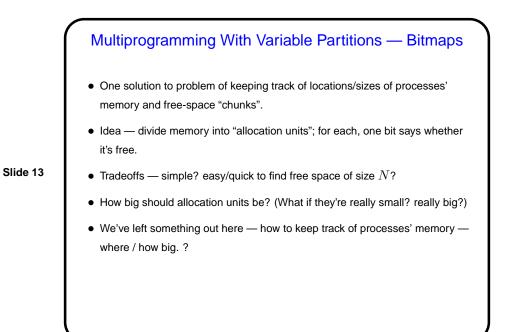
physical addresses, generates interrupt if invalid.

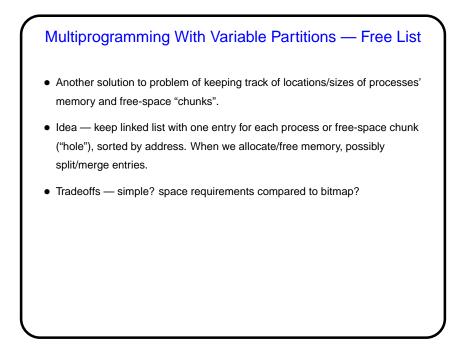
	A Simple MMU
Slide 11	 Idea — map each process's address space to a contiguous chunk of real memory, based on base and limit addresses (<i>B</i> and <i>L</i>): Program address <i>p</i> maps to memory location <i>B</i> + <i>p</i>. If <i>B</i> + <i>p</i> > <i>L</i>, invalid (out of bounds). If <i>B</i> and <i>L</i> are different for each process — solves both problems. Turn this into hardware (MMU) by 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.

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 12





Slide 14

