Slide 1

Administrivia

- Homeworks 2a, 2b available; due next week.

Slide 2

Sharing Memory, Recap/Review

- Time-sharing tried but not very practical beyond very early days.
- Space-sharing obvious alternative, but then do need to protect/isolate processes from each other.
- Either way need to protect the O/S from user processes.
- Solutions, as with virtualizing CPU, will need combination of hardware and O/S.

Slide 3

Sharing Memory — Relocation

- Simplest way is to give each process a contiguous chunk of memory. This can work without any virtualization *if* we deal with "relocation problem" — programs are compiled/linked assuming particular starting locations, which can't be right for *all* programs.
- Textbook shows an example. If the x86 throws you, similar MIPS code might be:

```
lw $t0, 0($s0)
addi $t0, $t0, 3
sw $t0, 0($s0)
```

  with all instructions 4 bytes in length rather than varying lengths.

Slide 4

Relocation, Continued

- One way to cope is "static relocation". Alluded to in Computer Design. Idea is that when loading program into memory we patch up all references to absolute addresses.

  Obviously a bit cumbersome and means more work if we ever want to move the program to a different spot in memory.

  Also provides nothing in the way of protection/isolation.
- Another way, more flexible and with possibility of protection/isolation, is "dynamic relocation", a.k.a. address translation.

Slide 5

Address Translation

- Key idea is that programs use “virtual addresses” relative to an “address space” abstraction.
- Challenge is for O/S to map that to physical hardware.
- Will need much help from hardware; parts of the hardware involved referred to as “MMU” (memory management unit).

Slide 6

Memory Management — Contiguous Allocation

- Stay with idea of one contiguous chunk of memory per process, but allow program to continue to use addresses relative to 0.
- For each process, O/S has to keep track of “base” and “limit”/“bound”. Values kept in special registers for use by MMU, changed on switch between processes. Clearly instructions to switch them need to be privileged!
- Actual arithmetic for translation is simple, no? And while MMU is doing that, it can also be checking for out-of-bounds reference using limit/bound register. What if found? Generate hardware exception (interrupt). (This addresses the question of protection/isolation, no?)

Slide 7

Memory Management with Contiguous Allocation — Hardware

- Translate addresses, using base register.
- Check for out of bounds, using bound register, generating exception if found.
- Allow changing these registers only in supervisor/kernel mode.

Slide 8

Memory Management with Contiguous Allocation — Software

- Find space for new process during process creation. If all address spaces the same size, not too hard — think of memory as consisting of big chunks, all the same size, and keep a list of those that are free.
- Switch MMU registers when switching processes.
- Deal with any out-of-bounds exceptions generated by MMU.

Slide 9

Memory Management — Next Steps

- Early on, contiguous-allocation scheme was widely used.
- Most versions did relax the fixed-size-per-process rule, but then managing free space was far more complicated. (You can perhaps imagine?)
- So as a next step — "segmentation".

  (Are you wondering whether maybe those cryptic errors generated by C programs that misuse pointers are about to make sense?)

Slide 10

Memory Management — Segmentation

- Extension of contiguous-allocation scheme, but with more than one base/bound pair — i.e., more than one "segment".
- Idea is to have one segment per logical region of address space — one for code and fixed data, one for heap, one for stack, with one base/bound pair per segment.
- How then to translate from virtual address?

Slide 11

Segmentation — Address Translation

- One way ("explicit") — use first few bits of address as segment number, the rest for offset within segment. Simple but limits size of largest segment to $1/2^n$ (n the number of segments) of total number of addresses.
- Another way ("implicit") — decide which segment based on where address came from. I admit I'm not quite sure how this works!
- Worth noting that heap grows toward larger addresses, stack toward smaller addresses. Could take this into account (details in textbook).

Slide 12

Segmentation — Shared Memory and Protection

- O/S designers realize early on — if multiple processes running the same program, no need for all of them to have a separate copy of code in memory.
- Easy way to do that — have them all share a segment containing only code.
- Clearly only works well if processes can't change it. So need some notion of what of its memory a process can actually change — "protection bits", add-on to base/bound pairs.

Slide 13

Segmentation — Coarse-Grained Versus Fine-Grained

- Discussion so far has been in terms of small number of big segments.
- But possible to also design system (hardware and O/S) to support large number of potentially small segments. Requires more complex support — "segment table" rather than small number of base/bound pairs.
- Fine-grained segmentation has some real conceptual advantages — e.g., think about a multithreaded program, where each thread needs its own stack.

Slide 14

Segmentation — Protection/Isolation

- Easy to understand how base/bound scheme provided this. How about segmentation?
- Basically simple — processes can only work with contents of memory they can find. If it's not in a process's segment table, it can't find it! That combined with usual(?) checking of bound/limit value provides what we want — with a provision for partially sharing what we *do* want to share.

Slide 15

Segmentation — Limitations

- Fine-grained segmentation in particular provides a nice abstraction for programmers.
- *But* now have the problem of managing not big fixed-size chunks (as with simple contiguous-allocation scheme) but chunks of varying sizes.
- Previously had to worry about wasted space within address space ("internal fragmentation"). Now have to worry about having plenty of free space total but none in big-enough chunks ("external fragmentation").
- Could address external fragmentation via periodic "compaction" but that has minuses too.

Slide 16

So Many Solutions?

- Textbook makes an interesting point:
- If there seem to be a lot of solutions, maybe that's a tip that there's no one best. "Hm!"?

Slide 17

Minute Essay

- Questions? (Though you may not have any until I start asking you to do assignments :-).)