Slide 1

Administrivia

- Homework 5 on the Web. Due next Monday.

Slide 2

Minute Essay From Last Lecture

- One person said keeping table in registers might be less secure. I'm skeptical — ?
- If the table is in memory, is it enough to know the starting point? (If entries are fixed size?)
- (Keep in mind that each process has its own page table.)

Slide 3

Paging — Recap

- Idea — divide both address spaces and memory into fixed-size blocks ("pages" and "page frames"), allow non-contiguous allocation.
- Makes for a much more flexible system but at a cost in complexity — keeping track of a process's memory requires a "page table" to be used by both hardware (MMU) and software (O/S).

Slide 4

Sidebar: Memory Management Within Processes

- What if we don't know before the program starts how much memory it will want? with very old languages, maybe not an issue, but with more modern ones it is.

  I.e., we might want to manage memory within a process's "address space" (range of possible program/virtual addresses).
- Typical scheme involves
  - Fixed-size allocation for code and any static data.
  - Two variable-size pieces ("heap" and "stack") for dynamically allocated data.
  - Notice — combined sizes of these pieces might be less than size of address space, maybe a lot less.

Slide 5

Page Table Entries

- Exactly what's in a page table entry depends partly on hardware.
- Required(?) fields — page frame number, present/absent bit.
- Optional but useful fields — bits that can be used to track usage ("referenced/modified"), bits indicating what access is allowed (e.g., read-only), etc.

Slide 6

Page Sizes and Other Details

- How big to make pages? compare extreme cases (really big, really small).
- If you know how big addresses are, what does that tell you about (maximum) sizes of physical/virtual memory?
- How big are page tables . . .

Slide 7

Page Table Size — Example

- Given a page size of 64K (2^{16}), 64-bit addresses, and 4G (2^{32}) of main memory, at least how much space is required for a page table? Assume that you want to allow each process to have the maximum address space possible with 64-bit addresses, i.e., 2^{64} bytes.
- (Hints: How many entries? How much space for each one? and no, this is not a very realistic system.)

Slide 8

Page Table Size — Example Continued

- Number of entries is $2^{64}/2^{16}$, i.e., 2^{48}.
- Size of each entry — at least enough for page frame number. There are 2^{16} of them, so we need 16 bits for that. Probably should also include a valid/invalid bit, for a total of 17 bits. Rounding up to a multiple of 8 bits (one byte), that's 3 bytes per entry.
- Total space is $2^{48} \times 3$ — bigger than main memory!! so, not realistic.

Slide 9

Performance / Feasibility Concerns

- Even with good choice of page size, serious performance implications — page table can still be big, and every memory reference involves page-table access — how to make this feasible/fast?
- (Remember that the MMU is hardware, and a bit about registers — local to the CPU, faster to access than memory but limited in number, can be general-purpose or dedicated to a particular use (e.g., the program counter).)

Slide 10

Page Tables — Performance Issues

- One possibility is to keep the whole page table for the current process in registers. Could possibly use general-purpose registers for this but likely would not. Should make for fast translation of addresses, but — is this really feasible for a large table? and what about context switches?
- Another possibility is to keep the process table in memory and just have one register (probably a special-purpose one) point to it. Cost/benefit tradeoffs here seem like the opposite of the first scheme, no?

  The big downside is slow lookup, though, and that can be improved with a "translation lookaside buffer" (TLB) — special-purpose cache.

Slide 11

Large Address Spaces

- Clearly page tables can be big. How to make this feasible?
- One approach — multilevel page tables.
- Another approach — inverted page tables (one entry per page frame).

Slide 12

Paging and Virtual Memory

- Idea — if we don't have room for all pages of all processes in main memory, keep some on disk ("pretend we have more memory than we really do").
- Or a simpler view: All address spaces live in secondary memory / swap space / backing store, and we "page in" as needed (demand paging).
- (Aside: Why are we even bothering? Can't the processor(s) access disk? Yes, but . . .)
- Making this work requires help from both hardware (MMU) and software (operating system).

Slide 13

Minute Essay

- None — sign in.