Slide 1

Administrivia

- Reminder: Homeworks 2a, 2b due Thursday.
- Reading Quiz 4 posted; due next Wednesday. Chapters 12 through 16.
- Homework 1a graded.

Slide 2

Paging — Motivation

- Simplest way to space-share memory — give each process a contiguous chunk, all chunks the same size — simple but not very satisfactory.
- Key problem is that it's good to have as large an address space as possible, with the assumption that most processes will use a small part of that (but which part and how much might vary).
- How to implement that? Segmentation is one way, preferably "fine-grained" for more flexibility. But with segmentation you have the problem of managing varying-size chunks.
- Paging tries to get best of both worlds — simplicity of allocating memory in fixed-size chunks, ability to efficiently represent "sparse" address spaces.

Slide 3

Paging — Key Idea

- Key idea is to pick a convenient size N (often a power of two), and:
- Divide each process's address space into N-byte "pages".
- Divide physical memory into N-byte "page frames".
- Place pages in page frames wherever they'll fit. No need for a process's pages to be contiguous or in order.
- Now the problem of keeping track of what's free is simpler again — list of page frames.
- Store map from page number to page frame number in "page table".
- Textbook Figure 18.2 shows doing this for one process, but idea works for more than one — distinct page table for each process.

Slide 4

Paging — Address Translation

- By dint of drawing a few pictures, probably not hard to convince yourself that to translate virtual address A to physical address P you need to:
- Divide A by page size N, giving quotient Q and remainder R. Quotient is page number, which we look up in page table to find page frame number Q'. Remainder is offset. Multiply Q' by N and add R and you have the physical address.
- Division is slow, however, and if our goal is efficiency?
- Recall however that division *by a power of 2* is quite fast! Which is why page sizes usually are powers of 2.

Slide 5

Sidebar: How Much Memory Can We Address?

- Note that size of address spaces is constrained by size of address — in a "32-bit" system, addresses are 32 bits, which means the largest address space is 2^{32} bytes, or 4GB.
- Similarly, number of bits available for physical address constrains size of physical memory. May be the same as number of bits for virtual addresses, but might be smaller. (Web search suggests that addresses are limited to 48 bits on some "64-bit" systems. But as textbook points out, maximum address space with 64-bit addresses almost unimaginably huge.)

Slide 6

Contents of Page Table

- Page table is a map. What's in each entry? Page number and page frame number? Not exactly . . .
- No need to store page number — implicit in index. So, page frame number, plus you want some way to indicate that this page isn't in use, so there isn't a matching page frame. Typically call this a "valid" bit.
- Other useful bits:

  Protection — can process read, write, execute from this page.

  Present — is page valid but not in memory (much more about that later).

  Referenced, modified bits — help track page usage (more about this later too).

Slide 7

Page Table Size

- Page table sizes are manageable with very tiny memories, but for anything realistic . . .
- Textbook does calculations for one example. Let's do another:

  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

Page Tables in Memory — Problems

- So one potential problem is how much space page tables take up in memory.
- Another problem is speed:
- Would be fast if we could keep the whole page table in registers, but — well, no, right?
- Logically enough to just keep address of page table in a register. But now access to any element of memory requires *two* accesses, one for a page-table entry!
- So paging is great from standpoint of supporting a nice abstraction. But can we make it acceptably efficient? (Teacher question. Yes!)

Slide 10

Minute Essay

- Questions? Is this making sense so far?