Contents

About these reading notes

These are my own personal reading notes that I took (me, Franklin) as I read the textbook. I'm providing these to you as an additional resource for you to use while you're reading chapters from the textbook. These notes do not stand alone on their own — you might be able to get the *idea* of a chapter while reading these, but you're definitely not going to **get** the chapter by reading these alone.

These notes are inconsistently all of the following:

- Me summarizing parts of the text.
- Me commenting on parts of the text.
- Me asking questions to myself about the text.
    - … and *sometimes* answering those questions.

The way that I would expect you to read or use these notes is to effectively permit me to be your inner monologue while you're reading the textbook. As you're reading chapters and sections within chapters, you can take a look at what I've written here to get an idea of how I'm thinking about this content.

Chapter 20: Paging: Smaller tables

- Before you start reading this chapter, try to think on your own about how you might change this idea of a page table to make it "smaller". It's hard to do this entirely independently of looking at the chapter, since the title of the first subsection is on the first page, so try to think about it in terms of **data structures**. What kind of a data structure could you use to implement a page table? How would you incorporate this idea of a valid bit (the page is allocated or not allocated) into that data structure?

20.1 Simple solution: bigger pages

- Logically convince yourself that this makes sense: bigger pages with the *same size address space* means **fewer** pages.
    - If you have 4KB pages on an x86 system with a virtual address space of 64GB (this is not a real number, just for thought's sake), how many pages are there? How big would that page table be?
- Internal fragmentation: it's back!
- The aside here "Multiple page sizes" tells a little bit more about that idea from chapter 19.

20.2 Hybrid approach: paging and segments

- Make sure you get what this structure is: we're not getting segments back, but instead we're getting sets of pages, one for each of code, heap, or stack.
- This sort of makes sense that the allocated pages for these segments are smaller than the entire page table; does this mean that we've got a limit on how big the segments can be now, though?
- Argh! External fragmentation! Again!

20.3 Multi-level page tables

- OK, data structures time.
- This basic idea on the bottom of page 5 *kind of* seems similar to the idea of indirect pointers in the file system. Is it a similar idea?
- Figure 20.3 does a great job of showing how a linear page table might be translated into this multi-level page table.
- Will this multi-level page table require *more* memory accesses than a linear page table?
  - See the middle of page 7 for an answer.

A detailed multi-level example

- This example *really* steps through the translation from linear page tables to multi-level page tables, so make sure you've got the basic idea of what the linear page table is doing before you try grokking the translation to multi-level page tables.
- How much of this multi-level page table stuff do you think *could* be done in hardware? How much of it has to be done in software? Does address translation happen in software or hardware?

More than two levels

- "In this example, assume we have a 30-bit virtual address space"; this is getting pretty close to reality with 64-bit systems. How big of an address space is 30 bits? How many pages are there in this virtual address space when you've got 512 byte pages as the authors are doing?
- Step through the pseudocode on page 12; how is this different from the basic linear page table? What parts of this do you recognize in terms of error states?

The translation process: remember the TLB

- You're not required to read about the TLB, but it's a cache that helps improve the performance of lookups from virtual addresses to physical addresses. You can find more about TLBs in chapter 19.

- (Oh no, we forgot about the TLB. https://www.youtube.com/watch?v=tgMyaAK-aZw)

20.4 Inverted page tables

- How exactly does this idea change how big page tables are? Does this change the virtual address space for a process?

20.5 Swapping the page tables to disk

- Swapping is coming up *real soon*, but you might have noticed: processes can have a virtual address space that's bigger than physical memory. What happens when a process *actually uses more than physical memory*? What happens when all processes together use more than available physical memory? (like what happens when you start Chrome and open 10 tabs)