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 9: Scheduling: proportional share

Thinking about the problems that the authors identified with the MLFQ, what's the **main** problem that fair-share scheduling is trying to solve?

> "every so often, hold a lottery to determine which process should get to run next; **processes that should run more often should be given more chances to win the lottery**."

That sounds an *awful lot* like "higher priority", and that sounds an *awful lot* like a "high priority queue" in a MLFQ. Can you convince yourself yet that there's a difference?

9.1 Basic concept: Tickets represent your share

Way outside the scope of this course: how much do you think the choice of source for random numbers affects the performance of this kind of a scheduler?

9.2 Ticket mechanisms

Make sure that you can distinguish here between a "user" and a "job"!

I am not convinced about the utility of this idea of currencies. The idea is similar to real-life currency: USD (US dollars) and CAD (Canadian dollars) have a different value. One US family and one Canadian family go to England, and the parents send their kids to the mall (which only accepts GBP) with their own currencies. The kids then have to go to the money changer and get GBP. Each are allocated a certain proportion of USD and CAD, and that proportion is ultimately translated into GBP. But why is this a useful mechanism? The authors don't really help us out with this.

Ticket transfer seems to make logical sense (when processes are working together, they themselves know more about what they need their peers to do than the scheduler itself could know). Can you think about any ways that ticket transfer could be gamed?

9.3 Implementation

What kind of a world are we living in when a textbook links to Stackoverflow????

I really don't care for the design of this implementation. The authors say it's simple, then go on to describe this overly complicated system that involves counting while stepping through a list. How would *you* design this policy?

9.4 An example

Does the way that fairness is calculated here make sense? We're calculating a ratio, and saying that (in this specific example) "the first job finished in half the time of the second job". This makes sense when both jobs have the same runtime. *Can* you calculate fairness in this way when jobs have different runtimes?

9.5 How to assign tickets?

9.6 Stride scheduling (note that this section is optional)

This seems very complicated despite being described as "straightforward". The important bits here are that we're keeping track of how long each process has run globally compared to other processes.

The code snippet on pg 6 does a pretty decent job of breaking down what actually happens to any individual job, combined with figure 9.3 on pg 7.

9.7 The Linux completely fair scheduler (CFS) (note that this section is optional)

> "… scheduling uses about 5% of overall datacenter CPU time."

This is one of those things that's weird to think about: The scheduler itself is just code (probably C) that someone has written, and actually runs on the CPU much in the same way that your programs do. With that in mind, it's someone's job to actually *optimize* schedulers to make them be faster, and therefore make sure that user programs (like ours) get to spend more time on the CPU.

The idea of counting how much time a process has spent running on the CPU goes almost all the way back to the beginning of this block of chapters. Thinking about the different approaches that you've seen so far (e.g., FCFS, SJF, round robin, MLFQ), which does CFS most closely resemble? Does it resemble a hybrid of any of these approaches?

Weighting (niceness)

So niceness isn't *directly* the priority of a process, but rather **advice** about the priority of a process. The advice is then mapped to a weight, then the weights are used to compute how much vruntime a process "consumes" (see equation 9.2 on pg 10).

Using red-black trees

This is a fun "I can't sleep and it's 3am and I'm bored and for some reason thinking about data structures and operating systems" kind of a problem: Actually measure the difference between using a red-black tree and a plain list to do lookups here. Of course, nobody is ever going to do think that, so nobody will ever do this, but…

Plus: that we can make real statements about the efficiency of lists ($O(n)$) and balanced trees ($O(\log(n))$) makes the effort to do this *particularly* worthless.