

Contents

About these reading notes	2
Chapter 8: Scheduling: The Multi-Level Feedback Queue	2
8.1 MLFQ: Basic rules	2
8.2 Attempt #1: How to change priority	3
Example 1: A single long-running job	3
Example 2 and Example 3	3
Problems with our current MFLQ	3
8.3 Attempt #2: The Priority Boost	4
8.4 Attempt #3: Better accounting	4
8.5 Tuning MLFQ and other issues	4

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 8: Scheduling: The Multi-Level Feedback Queue

This idea was first described in 1962!

8.1 MLFQ: Basic rules

So we've got a list of queues (yo dawg I herd you like data structures), and *each* queue has a priority level. How would you even begin to arrive at deciding what priority level a job/process gets?

On Linux and macOS, *one* measurement of priority is called the “nice” level of the process. You can run this command on aviary or rodents:

```
ps -eo nice,comm
```

That will print out a list of the currently running processes and how “nice” they are. A lower value of “nice” implies that it's a higher priority (it's *meaner*?), so negative values for nice have a very high priority, and positive values means it's very nice (lower priority). Processes are launched with a nice value of 0 by default, but you can **manually** change the nice level of a program (at least on launch) using the `ni ce` command (see `man ni ce`). You'll see this near the end of the chapter, but niceness is “advice” about priority that the scheduler can include in its decisions about how to assign priority to a process.

The authors say here that “interactive” commands (ones that repeatedly get blocked waiting for I/O on an input device) get higher priority than those that don’t. How does the OS figure this out? What kinds of statistics might the OS need to keep about a process to know this? *When* might the OS be able to collect those statistics?

8.2 Attempt #1: How to change priority

Job priority can change over time. Think again about design: do you need to store priority as a property of the process?

The answer is no, you don’t. So, uh, OK, *where is it, then?*

Side note: Look back at `struct task_struct` (here: <https://github.com/torvalds/linux/blob/master/include/linux/sched.h>), the only places where the word priority exists in this PCB is in terms of “real-time” processing (this is very much out of scope for this course).

This approach only moves processes *down* in terms of priority (everyone immediately has highest priority, then your priority is lowered if you don’t seem to have any interactive looking tasks). This goes back to the last section, how does an OS figure out whether or not a process is doing “interactive” tasks? (hint: when does a process manually give up control to the OS?)

Example 1: A single long-running job

In example 1 (on pgs 3–4), we go from 8 queues to 3. How do you think we decide how many queues to have in the list?

Example 2 and Example 3

Beyond that, the examples themselves are (I think) very good visual ideas of how the MLFQ works. Take the time to actually step through these examples!

Problems with our current MFLQ

The authors tell you to stop and “think as deviously as you can”. Do it. Stop here (at the bottom of pg 5) and actually try to think about how you might break MFLQ with the rules that have been set up.

Is starvation a problem in this situation? Seriously, decide for yourself here: would you rather have a system that’s responding to your input more quickly, or would you rather have a system that’s making meaningful progress in all processes? Does this change if you’re running, for example, a web server

that doesn't *have* “interactive” features? (e.g., there's no GUI running on the physical system, there's no monitor attached to it, there's no keyboard or mouse attached to it)

Is this issue of a “smart user” gaming the scheduler a real problem? Can you think of any malicious kinds of jobs where a “smart user” would want to take over the CPU in this way?

“a program may *change its behaviour* over time;”

What kind of a program might do this? What kind of a program would be highly interactive, then not? What kind of a program would be not interactive at all, then suddenly interactive?

8.3 Attempt #2: The Priority Boost

I feel like this solution would make for “bursty” looking behaviour — everything would suddenly get the same level of priority, then slowly get shuffled down again. If the figures the authors were showing here were longer, I feel like you'd start to see a pattern with something that looks like “saw teeth”. The authors don't show it this way, though. I actually don't get figure 8.6, the figure on the right doesn't actually match to me what they describe (no process is ever boosted to the top queue, but instead the “gamed” process just moves down).

8.4 Attempt #3: Better accounting

Now we're adding time tracking: just how much time has this process had on the CPU? We've now got a few different pieces of information here: priority, interactivity in terms of I/O, and how much time the process has spent on the processor. Can you think of any other measurements that you can make about a process as its running that might help inform scheduling in the future?

8.5 Tuning MLFQ and other issues

This is mainly about being able to vary the settings of a real MLFQ scheduler, but also telling us a little bit about how MLFQs are real and used in real operating systems.

Take a tiny bit of time to figure out how you configure a scheduler in, for example, Linux.