Lab 3

Part I

Read, understand, and summarize the following scheduling approaches: first come, first server (FIFO); shortest job first (SJF); preemptive shortest job first/shortest remaining time/shortest time-to-completion first (PSJF/STCF), round robin, lottery scheduling, multi-level feedback queue.

Discuss more models from your research. (Optional)

Put it as part of your Report I. Make use of lab to get feedback early.

Part II

Think about the following short answer questions:

What are the main tasks of the scheduler?

What is the difference between preemptive and non-preemptive scheduling?

What is the advantage of a shorter scheduling quantum?

What is the advantage of a longer scheduling quantum?

Why is feedback scheduling useful for interactive jobs?

Are these scheduling policies subject to starvation?

- Shortest Job First scheduling?
- Round Robin scheduling?
- First Come First Served scheduling?
- Fixed Priority scheduling?
- Multi-level feedback scheduling?

Part III - Stride Scheduling – Optional Reading

Lottery scheduling may not deliver the exact right proportions over short time scales. For this reason, Waldspurger invented **stride scheduling**, a deterministic fair-share scheduler [W95].

Stride scheduling: each job in the system has a stride, which is inverse in proportion to the number of tickets it has. In our example, with jobs A, B, and C, with 100, 50, and 250 tickets, respectively, we can compute the stride of each by dividing some large number by the number of tickets each process has been assigned. For example, if we divide 10,000 by each of those ticket values, we obtain the following stride values for A, B, and C: 100, 200, and 40. We call this value the **stride** of each process; every time a process runs, we will increment a counter for it (called its **pass value**) by its stride to track its global progress.

The scheduler then uses the stride and pass to determine which process should run next. The basic idea is simple: at any given time, pick the process to run that has the lowest pass value so far; when you run a process, increment its pass counter by its stride.

In the example below, we start with three processes (A, B, and C), with stride values of 100, 200, and 40, and all with pass values initially at 0. Thus, at first, any of the processes might run, as their pass values are equally low. Assume we pick A (arbitrarily; any of the processes with equal low pass values can be chosen). A runs; when finished with the time slice, we update its pass value to 100. Then we run B, whose pass value is then set to 200. Finally, we run C, whose pass value is incremented to 40. At this point, the algorithm will pick the lowest pass value, which is C's, and run it, updating its pass to 80 (C's stride is 40, as you recall). Then C will run again (still the lowest pass value), raising its pass to 120. A will run now, updating its pass to 200 (now equal to B's). Then C will run twice more, updating its pass to 160 then 200. At this point, all pass values are equal again, and the process will repeat.

Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	...

Now you might be wondering: given the precision of stride scheduling, why use lottery scheduling at all? Well, lottery scheduling has one nice property that stride scheduling does not: no global state. Imagine a new job enters in the middle of our stride scheduling example above; what should its pass value be? Should it be set to 0? If so, it will monopolize the CPU. With lottery scheduling, there is no global state per process; we simply add a new process with whatever tickets it has, update the single global variable to track how many total tickets we have, and go from there. In this way, lottery makes it much easier to incorporate new processes in a sensible manner.

Extracted from: Chapter 9, Operating Systems: Three Easy Pieces

A short version of [W95], "Stride Scheduling: Deterministic Proportional-Share Resource Management" -- read.seas.harvard.edu/~kohler/class/aosref/waldspurger95stride.pdf