ECE254: Operating Systems and Systems Programming Spring 2015

Lecture 26 — Scheduling Algorithms

Jeff Zarnett

Scheduling Algorithms

Earlier we saw one example of a simple scheduling algorithm: always choose the highest priority (non-blocked) task, and execute it. There are many other choices we could make. We will examine the following options:

1. Highest Priority, Period
2. First-Come-First-Served
3. Round Robin
4. Shortest Process Next
5. Smallest Remaining Time
6. Highest Response Ratio Next
7. Multilevel Queue
8. Guaranteed Scheduling
9. Lottery
10. Feedback

The OS may maintain some data about each process to be used in making decisions about scheduling. They are, broadly [Sta14]:

1. The time spent waiting to run.
2. The time spent executing.
3. The total time of execution.

The first two criteria in the list can easily be measured, but the their one must be estimated or supplied by the user. Old batch systems did sometimes ask users to provide an estimate of how long they thought a task was likely to take. If the user underestimated, then execution would be halted at the time they said, even if his or her operation was unfinished. But the higher the estimate, the lower the priority the task received; if too high the user task might never be scheduled to run at all. Operating systems do not ask the users in desktop systems to estimate how long a process will run, but supercomputers and other batch jobs may still consider this as a criterion.

Highest Priority, Period

Implementing this is not difficult; we could have some priority queues (one for each priority level). If a task is not blocked, put it in its appropriate priority queue. If the process's priority is changed (manually or otherwise), move it to its new home. We might have a priority heap, or just one big linked list or array that we keep sorted by priority. Many options.

The flaw in this has already been identified: it is vulnerable to starvation. A process of relatively low priority may never get the chance to run, because there is always something better to do right now. Humans presumably

encounter situations like this all the time. Software projects may have bug reports open for years on end because they are never important enough to be addressed. In my personal life, there are books I'd like to read (the example "Der Deutsch-Französische Krieg" springs to mind), but I never get to them because something else always comes up... work to do, another book to read, a social event...

In some systems this is a desirable property. It does not fulfill all short term scheduling criteria, notably response time and fairness. It may be suitable for life-and-safety-critical systems, such as the process to control a robot arm, to prevent a situation where the robot arm goes through the wall and the building falls down and you're dead.

References

[Sta14] William Stallings. *Operating Systems Internals and Design Principles (8th Edition)*. Prentice Hall, 2014.