CS333: Operating Systems Lab

Lab 5: A New Scheduler in xv6

Goal

The goal of this lab is to understand process scheduling by building your own scheduling algorithm in xv6.

Before you start

- Download, install, and run the xv6 OS. You can use your regular desktop/laptop to run xv6; it runs on an x86 emulator called QEMU that emulates x86 hardware on your local machine.
- Learn how to add a new system call in xv6. You can follow the implementation of a simple existing system call (e.g., `uptime`) to see what files you need to change to add a new system call.
- Learn how to write your own user programs in xv6. For example, if you add a new system call, you may want to write a simple C program that calls the new function. There are several user programs as part of the xv6 source code, from which you can get an idea. Note that the xv6 OS itself does not have any text editor or compiler support, so you must write and compile the code in your host machine, and then run the executable in the xv6 QEMU emulator.

Implementing a new scheduling algorithm in xv6

The current scheduler in xv6 is a round robin scheduler. In this lab, you will modify the scheduler to take into account user-defined process priorities. Please follow the steps below.

1. Add a new system call to xv6 to set process priorities. When a process calls `setprio(n)`, the priority of the process should be set to the specified value. The priority can be any integer value, with higher values denoting more priority. You may also want to add a system call `getprio` to read back the priority set, in order to verify that it has worked.
2. Modify the xv6 scheduler to use this priority in picking the next process to schedule. How you interpret the priority and use it in making scheduling decisions is a design problem that is entirely left to you. For example, you can use the priorities as weights to implement a weighted round robin scheduler. Or, you can use the priorities to do strict priority scheduling. The only requirement is that a higher numerical priority should translate into more CPU time for the process. Your report must clearly describe the design and implementation of your scheduler, and any new kernel data structures you may have created for your implementation. Also describe the runtime complexity

of your scheduling algorithm. For example, the current round robin algorithm has a complexity of $O(1)$.

3. Make sure you handle all corner cases correctly. For example, what is the default priority of a process before its priority is set? What will happen when a process you want to schedule blocks before its quantum finishes? Also make sure your code is safe when run over multiple CPU cores. Please describe how you handle all such cases in your report.

4. Now, you will need to write test cases to test your scheduler. You must write a separate user-space test program `testmyscheduler` that runs all your tests. Your test program must fork several processes, set different priorities within the forked processes, and show that the priorities cause the child processes to behave differently. You must come up with at least **two** test cases in this test program, where your scheduler causes a different execution behavior as compared to the default round robin scheduler, and you must *quantify* and *explain* this difference in the execution time of processes. Note that your test cases must use both CPU-bound and I/O bound processes, to show that your scheduler works correctly even when processes block.

   Here is a simple test case to give you an example of what is expected. You can create two identical child processes that execute a CPU-intensive workload, and show that one finishes execution much faster than the other when its priority is increased. Further, if you implement a weighted round robin scheduler and give the higher priority process twice as many time slices as the lower priority one, you should show that it finishes execution roughly twice as fast. You should come up with such test cases that showcase your scheduling algorithm both qualitatively and quantitatively.

Submission and Grading

You may solve this assignment in groups of one or two students. You must submit a tar gzipped file, whose filename is a string of the roll numbers of your group members separated by an underscore. For example, `rollnumber1_rollnumber2.tgz`. The tar file should contain the following:

- `report.pdf` should contain a detailed description of your new scheduler: the scheduling policy, design of any new data structures, implementation details, and how you handle various corner cases. Further, you must describe your test cases in some detail, and the observations you made from them.

- A patch of your code. Your patch must include all modifications to the source code as well as to the Makefile. Do not forget to include your test program.

Evaluation of this lab will be as follows.

- We will read through your report to understand your design and testing of the scheduler.

- We will patch your code onto our codebase and run your test program. Please make sure that the output of your test program is clean enough to understand the results of the tests.

- We will read your code to understand what you have done. Please make sure your code is well-documented and readable.