Group Assignment

September 4, 2019

- You need to submit your assignment in zip file. Zip file will contains your codes and README file.Only one group member need to submit the assignment.Please name your file as *rollno*_(member in your group) e.g., *IIT*2019001_4.

Implementing priority based round robin scheduler in XV6

The current scheduler in xv6 is a round robin scheduler. Please look at lines 335-337 in `scheduler()` function in proc.c code in xv6 to verify that xv6 runs a round robin scheduler. In this assignment, you will modify the scheduler to take into account user-defined process priorities. Please follow the steps below.

- Add new system calls in xv6 to get and 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. Add a system call getprio to read back the priority set, in order to verify that it has worked. In an earlier lab assignment you have already been told how to add a system call in xv6. However, in this particular system call, you have to add a variable in process control block to store the priority of that process. To do the same, look in to `struct proc {...}` in proc.h. You can add an unsigned integer variable in that structure to store priorities. Do remember that now every process will get some priority, hence, you should give a default priority of some fixed integer to every process when a new process is allocated in memory. You can do that by modifying the `allocproc` function in `proc.c`. Another important thing here is that we are only passing a priority value as a parameter to setprio(n), hence, you need to get the process id in the syscall code of setprio() function. To do that you can use `struct proc *curproc = myproc();`.
- Modify the xv6 round robin 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.

- 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.

- 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.

Note: 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.