CSC 345 – Project #3: User Program and Scheduler

Due: Apr 9, 2017, before midnight

Objective:
Understand how xv6 creates and schedules processes. Design and conduct experiments on several different scheduling algorithms.

Due Date:
Apr 9, 2017, 11:59pm

Project Details:
This project is mostly about understanding how `proc.c` is implemented. Particularly, scheduler() and sched() are the most important ones. xv6, implements round-robin scheduling algorithm since it is dependent on timer interrupt fired every 100 milliseconds. In this project, we want to observe how xv6 performance changes if we replace this scheduler to other schedulers.

Program requirements (100 pts):

- (25 points) Read the Chapter 5 of xv6-book.pdf. Explain, in your own words, how xv6's round-robin scheduling algorithm is implemented and works. You need to provide detailed information, e.g., how timer interrupt works in terms of scheduling, how context switching, including user and kernel virtual memory space switching process, etc. Quality and clarity of explanation will also be assessed, i.e., pictorial description is preferable.
- (15 points) Implement Unix system command [ps] for xv6. Note that [Ctrl]+[P] will trigger keyboard interrupt and print out list of currently running processes.
- (15 points) Implement FCFS version of scheduler()
- (15 points) Implement one type of Priority-based version of scheduler(). Either preemptive or non-preemptive is okay.
- (25 points) Design & conduct comparative experiments for the three scheduling algorithms. Provide some statistical analysis of your result. Like the first item, quality and clarity will be assessed, e.g., charts and graphs explaining results are preferable. Provide a user program running inside xv6 that can demonstrate your experimental setup.
- (5 points) In a separate page of your report, discuss issues you encountered while implementing and conducting the experiments, and how you resolved (or will try to resolve) them.

Extra credit:

- (75 points) Design and implement a new scheduler for xv6 that performs ***better*** than built-in round-robin scheduler. In the report, you should provide either (mathematically rigorous) theoretical proof *or* extensive and scientific, empirical results to justify your claim.
- (75 points) Design and implement Linux Complete Fair Scheduler (CFS) for xv6. Fully comment your code. Provide a short (1 page) document describing your implementation. You need to provide a xv6 user program that demonstrates the correctness of your implementation. Only fully-working CFS implementation will be granted credits.

Formatting Requirements

- Comment the part of your modified code.
- In a separate file (**report.pdf**), clearly describe which requirements you implemented, including your claim on extra credit (if any).
- In another separate file (**discussion.pdf**), attach your discussion log including date/time.
- Prepare separate copy of xv6 for each scheduler you implement. Put them in a separate folder, e.g.,

  xv6-project3-RR/ (original xv6)
  xv6-project3-FCFS/
  xv6-project3-Priority/

  and put your implementations inside.

What to turn in

- Zip the folder containing your source file(s) **and Makefile**, then submit it through Canvas by the deadline. There will be a penalty for not providing any working Makefile.

Hints

- For this project, implementation should not be that much once you understand materials in Chapter 5.
- If you haven't, it might be helpful to read the first few chapters of the xv6-book.pdf regarding timer interrupt and lock part (Chapter 3, Chapter 4).
- It may not be a good idea to completely remove timer interrupt when implementing FCFS or non-preemptive priority schedulers because some important user processes contain intentionally designed infinite loops (e.g. shell) or they need frequent, intermittent CPU usage (e.g. console device). Thus, it might be a better strategy to add some check routine while traversing through the process table, to either sleep / wake up only those belong to user level processes (except the shell). Note that you can always obtain the name of the process from the process structure.