Operating Systems Project #2 - xv6 MLFQ Scheduler

SWE3004-42 Introduction to Operating Systems - Spring 2018

Due date: Oct 19 (Fri) 11:59pm

1 Goal

In this project, you will implement a new scheduling policy MLFQ in xv6. The detail algorithm is described in chapter 8 of the OSTEP book. Note that this project will be a lot more difficult than the first project. So, you better start on this project as early as you can.

The objectives for this project are as follows:

- To gain further knowledge of a real OS kernel
- To familiarize yourself with the CPU scheduling
- To make a graph to show your scheduler behaves appropriately

2 Details

You need the setnice and getnice system calls that you implemented for project #1 to verify your scheduling policy. The top queue (numbered 0) has the highest priority and the bottom queue (numbered ~~40~~) has the lowest priority. In addition to the system calls you implemented for the first project, you need to implement `int getpinfo(struct pstat*)` to get information about how many times each process has been chosen to run. This system call should return 0 if successful, and -1 otherwise, for example, if a bad or NULL pointer is passed. The structure `pstat` is defined as follows:

```
#ifndef _PSTAT_H_
#define _PSTAT_H_

#include "param.h"

struct pstat {
  int inuse[NPROC];   // whether this slot of the process table is in use (1 or 0)
  int nice[NPROC];    // the nice value of each process
  int pid[NPROC];     // the PID of each process
  int ticks[NPROC];   // the number of ticks each process has accumulated
};

#endif // _PSTAT_H_
```

3 How to Start Project

Download xv6 source code by running the following git command. Note that you should rename your first project directory (xv6-skku) to something else such as 'proj1' before cloning the second project file. Otherwise, you may overwrite the first project.

```
$ mv xv6-skku proj1
$ git clone https://github.com/jinsoox/xv6-skku.git -b pa2
```

4 Details about MLFQ

When a process is first created, it should be placed at the end of the highest priority queue. When a process uses up its time-slice, it should be downgraded to the next priority level.

The time slice for priority 0 should be 1 timer tick while the time slice for priority 1 is 2 timer ticks. I.e., for priority k, the time slice is 2^k timer ticks.

When a timer tick occurs, whichever process was currently using the CPU should be considered to have used up a timer tick's worth of CPU. (Yes, a process could game the scheduler this way by relinquishing the CPU just before the timer tick occurs; ignore this!)

If a process voluntarily releases the CPU, it should be placed at the end of queue in the same level when it becomes ready to run. That is, it should not preempt a process at the same priority. This can prevent a malicious process from gaming the scheduler, to some extent. If there's no other process in the same or higher level, the process must preempt of course.

Whenever a process is moved to a different priority level, it should be placed at the end of the corresponding queue. A round-robin scheduler should be used for processes at the lowest priority.

However, if a process uses up its time slice three times at the lowest level, i.e., 3×2^40, the process has to be boosted to the topmost level queue.

5 How to Submit

To submit your project, you must run `make tarball` command in xv6-skku directory to compress your source codes into one .tar.gz file. This .tar.gz file must be submitted on swin server using the following command.

```
$ os_submit project2 your_tar_file_name
```

Note: Please don't forget writing your ID and project number in the Makefile before creating a tarball.

For any questions, please post them in Piazza so that we can share your questions and answers with other students.

6 You must work on In-Ui-Ye-Ji Cluster

It seems some students have implemented the first project in their own virtual machine but copy-and-pasted the modified source codes into swin server. This is NOT acceptable for the second project. You MUST write codes in In-Ui-Ye-Ji servers because I want you to become familiar with Linux server-client environment and because I want to make sure you work on this project on your own. If you work on this server, I can check how long you use this machine, how long it takes for you to finish, how many lines of codes you write every day, what commands you run, where you connect from, and many other history. Based on this, I can figure out if it is really you who implemented the codes. If you get caught cheating, you will fail this class of course, no matter what. If you do not use this cluster, you will also get penalty for that as I will not believe it is you who wrote the codes.

One more thing, if you attempt to hack this cluster for any reason, you better not get caught. Otherwise, I have to take it all the way up to the university council to put a sanction on your academic record. I do not want all the troubles, so please don't even try.