Project #2

xv6 Scheduler

Objectives

There are three objectives to this assignment:

1. To familiarize yourself with a real scheduler.
2. To change that scheduler to a new algorithm.
3. To make a graph.

Overview

In this project, you will be putting a new scheduler into xv6. It is a mix of two schedulers: the **multi-level feedback queue (MLFQ)** we learned about in class and is described *in* ***Chp008 - CPU - Scheduling -MLFG*** and the **lottery scheduler** and is described in ***Chp009 - CPU - Scheduling -Proportional Share***.

The basic idea is simple: Build a simple two-level scheduler which first places jobs into the high-priority queue. When a job uses its time slice on the first queue, move it to the lower-priority queue; jobs on the lower-priority queue should run for two time slices before relinquishing the CPU.

When there is more than one job on a queue, each should run in proportion to the number of tickets it has; the more tickets a process has, the more it runs. Each time slice, a randomized lottery determines the winner of the lottery; that winning process is the one that runs for that time slice.

Details

You will need a couple of new system calls to implement this scheduler. The first is `int settickets(int num)`, which sets the number of tickets of the calling process. By default, each process should get one ticket; calling this routine makes it such that a process can raise the number of tickets it receives, and thus receive a higher proportion of CPU cycles. This routine should return 0 if successful, and -1 otherwise (if, for example, the user passes in a number less than one).

The second is `int getpinfo(struct pstat *)`. This routine returns some basic information about each running process, including how many times it has been chosen to run and its process ID, and which queue it is on (high or low). You can use this system call to build a variant of the command line program `ps`, which can then be called to see what is going on.

Beyond the usual code, you will have to **make a graph** for this assignment. The graph should show some timelines of processes running in each scheduler, including which queue they are on, and how much CPU they received when assigned some particular number of tickets. Use the graphs to prove that your scheduler is working as desired.

Tips

Most of the code for the scheduler is quite localized and can be found in `proc.c`; the associated header file, `proc.h` is also quite useful to examine. To change the scheduler, not much needs to be done; study its control flow and then try some small changes.

You will need to figure out how to generate random numbers in the kernel; some searching should lead you to a simple pseudo-random number generator, which you can then include in the kernel and use as appropriate.

You will need to understand how to fill in the structure `pstat` in the kernel and pass the results to user space. This structure looks like:

```
#ifndef _PSTAT_H_
#define _PSTAT_H_

#include "param.h"

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

#endif // _PSTAT_H_
```

The ***xv6 book*** particularly **Chapters 5** will be useful for this project.

General Advice

Start small, and get things working incrementally. For example, first get a program that simply reads in the input file, one line at a time, and prints out what it reads in. Then, slowly add features and test them as you go.

Testing is critical. One great programmer I once knew said you have to write 5-10 lines of test code for every line of code you produce; testing your code to make sure it works is crucial. Write tests to see if your code handles all the cases you think it should. Be as comprehensive as you can be. Of course, when grading your projects, we will be. Thus, it is better if you find your bugs first, before we do.

Keep old versions around. Keep copies of older versions of your program around, as you may introduce bugs and not be able to easily undo them. A simple way to do this is to keep copies around, by explicitly making copies of the file at various points during development.

Keep your source code in a private directory. An easy way to do this is to log into your account and first change directories into `private/` and then make a directory therein.

Submission

Copy all your source files (but not `.o` files, please, or binaries!) into your projects' private directory (i.e., `~/ICS462/CODE/project2/xv6-private`). A simple way to do this is to copy everything into the destination directory, then type `make` to make sure it builds, and then type `make clean` to remove unneeded files.

```
shell% cp -r . ~/ICS462/CODE/project2/xv6-private
shell% cd ~/ICS462/CODE/project2/xv6-private
shell% make
shell% make clean
```

Finally, into your `…/project2/xv6-private` directory, please make a README file. In there, describe what you did a little bit. The most important bit, at the top, however, should be the authorship of the project.

Zip your `…/project2` directory and upload it into **D2L ➔ Assessments ➔ Assignments ➔ Project #2:…** folder.

Note: you are only allowed to submit a single file (the last uploaded file).