CS/SE 4348 Operating Systems

Project 1: Add Lottery Scheduler to xv6

Due date 20th Feb, 2019

Read the project description below carefully. Also, watch this video. (You can skip the first 10 minutes of the video as some parts may not be relevant and some parts you may already know.)

You can do this project individually or with a partner. You cannot share your work with anyone other than your project partner.

This project can be done only in **csjaws.**

Add a lottery scheduler to xv6

In this project, you'll replace the current round robin scheduler in xv6 with a **lottery scheduler**, which we discussed in the class and is also described in OSTEP book. The basic idea is simple: assign each running process a slice of the processor 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.

The objectives for this project:

- To gain further knowledge of a real kernel, xv6.
- To familiarize yourself with a scheduler.
- To change that scheduler to a new algorithm.
- To make a graph to show your project behaves appropriately.

Setting Tickets

You need to implement a new system call to set the number of tickets. The prototype of the system call is:

```
        int settickets(int)
```

This call 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 caller passes in a number less than one).

Implementing Scheduler

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. (It is also very useful to read (relevant parts of) chapter 5 on Scheduling in xv6 book.)
To change the scheduler, first study its control flow. Find out the part of the code where round robin is implemented. Replace that code with your code for lottery scheduler.

You'll need to assign tickets to a process when it is created. Specfically, you'll need to make sure a child process *inherits* the same number of tickets as its parent. Thus, if the parent has 10 tickets, and calls **fork()** to create a child process, the child should also get 10 tickets.

You'll also 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.

Getting Process Statistics

You need to implement a second system call to gather some statistics about all the running process. The prototype for the second system call is

```
int getpinfo(struct pstat *)
```

This routine returns some information about all running processes, including how many times each has been chosen to run and the process ID of each. 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. The structure `pstat` is defined below. Note, you cannot change this structure, and must use it exactly as is. This routine should return 0 if successful, and -1 otherwise (if, for example, a bad or NULL pointer is passed into the kernel).

You need to understand how to fill in the structure `pstat` in the kernel and pass the results to user space. The structure should look like what you see below, in a file called `pstat.h`. You have to include this file in appropriate .c files.

```
#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 tickets[NPROC]; // the number of tickets this process has
  int pid[NPROC];     // the PID of each process
  int ticks[NPROC];   // the number of ticks each process has accumulated
};

#endif // _PSTAT_H_
```

Good examples of how to pass arguments into the kernel are found in existing system calls. In particular, follow the path of `read()`, which will lead you to `sys_read()`, which will show you how to use `argptr()` (and related calls) to obtain a pointer that has been passed into the kernel. Note how careful the kernel is with pointers passed from user space -- they are a security threat(!), and thus must be checked very carefully before usage.

Graph

Beyond the usual code, you'll have to make a graph for this assignment. The graph should show the number of time slices a set of three processes receives over time, where the processes have a 3:2:1 ratio of tickets (e.g., process A might have 30 tickets, process B 20, and process C 10). The graph is likely to be pretty boring, but should clearly show that your lottery scheduler works as desired.

xv6 Source Code

The xv6 source code for this project is /cs4348-xv6/src/xv6.tar.gz. You do not need any of the code that you implemented in exercise 1. Copy this file to your local working directory for this project and extract the source code tree using the command

```
tar -zxvf xv6.tar.gz
```

Testing

Sample programs to test your implementation is available in the directory `/cs4348-xv6/src/testscripts/p1/`. Copy these programs to your local …/xv6/user/ directory. To run these test programs you need to compile them and rebuild the xv6 kernel with them.

Submission

Copy your entire source code tree under xv6 to the directory /cs4348-xv6/xxxyyyyyy/p1 (You need to create p1 under /cs4348-xv6/xxxyyyyyy). Then change to this directory and ensure that your test programs still work. Finally, run ‘make clean’ to remove the *.o files and the kernel image files.

If you have worked with a partner, only one of you need to submit the files. But, both of you should create a text file named PARTNER in /cs4348-xv6/xxxyyyyy/p1 and save your partner’s name and netid in the file.

Also, submit your graph as pdf file in p1 directory.

Grading

Make sure you have provided adequate comments for someone else understand the code you have added/modified. 10% of the total points will be based on how well you have commented.
You may be asked to demonstrate your work to the TA. If you are not able to explain how your code works, then you may not get any points even though your code may work.