Homework 3: Lottery Scheduling

Due Date: April 20, 11:55 pm

Academic Honesty

Aside from the narrow exception for collaboration on homework, all work submitted in this course must be your own. Cheating and plagiarism will not be tolerated. If you have any questions about a specific case, please ask the instructor or the TAs. Also, we will be testing for plagiarism.

NYU's Policy on Academic Misconduct:
http://engineering.nyu.edu/academics/code-of-conduct/academic-misconduct

Notes

General Notes

- Read the assignment carefully, including what files to include.
- Don't assume limitations unless they are explicitly stated.
- Treat provided examples as just that, not exhaustive list of cases that should work.
- When in doubt regarding what needs to be done, ask. Another option is test it in the real UNIX operating system. Does it behaves the same way?
- **TEST** your solutions, make sure they work. It's obvious when you didn't test the code.

Homework #3 Notes

In this assignment we will go back to using xv6.

Start by getting a modified version of the xv6 source code from GitHub.

```
$ git clone https://github.com/gussand/xv6-public.git
Cloning into 'xv6-public'...
remote: Counting objects: 4484, done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 4484 (delta 11), reused 0 (delta 0), pack-reused 4448
Receiving objects: 100% (4484/4484), 11.69 MiB | 3.79 MiB/s, done.
Resolving deltas: 100% (1779/1779), done.
Checking connectivity... done.
```

Then, switch to the homework branch.

```
$ cd xv6-public
$ git checkout hw4
Branch hw4 set up to track remote branch hw4 from origin.
Switched to a new branch 'hw4'
```

The modified version has some new features:

1. It contains a random number generator. You can use it by including rand.h in a source file and then calling the function

```
long random_at_most(long max)
```

to get a random number between 0 and *max* **inclusively**.

2. It contains a random number generator. You can use it by including rand.h in a source file and then calling the

```
int gettime(struct rtcdate *date)
```

which retrieves the current date and time from the CMOS RTC and stores it in an *rtcdate* structure (you can see how struct rtcdate is defined by looking at *date.h*).

3. It contains a program designed to test the lottery scheduler you will develop, in the source file *lotterytest.c.*

4. For simplicity in testing, you should run the tests for this assignment in QEMU with just one CPU. You can do this by invoking the Makefile as:

```
make qemu CPUS=1
```

Lottery Scheduling

In this assignment, you will implement and test lottery scheduling, a randomized algorithm that allows processes to receive a proportional share of the CPU without explicitly tracking how long each process has been run.

Specifically, you should modify xv6 so that:

1. Each `*struct proc*` has an additional field, `*tickets*`, that tracks how many tickets it has.
2. New processes are assigned 20 lottery tickets when they are created.
3. When the scheduler runs, it picks a random number between 0 and the total number of tickets. It then uses the algorithm described in class to loop over runnable processes and pick the one with the
4. User processes have a new system call, `*settickets*`, that allows a process to specify how many lottery tickets it wants. Normally this would be a bad idea, since it would let a process hog the CPU by specifying an arbitrary number of tickets – but xv6 has no security anyway, so this is not that big a deal.

Once you have implemented this, you will want to test that the scheduler works. Aside from basic tests that the system is still functioning, you should verify that allocating more tickets to a process does increase its share of the CPU allotted by the scheduler.

Included in the source distribution is a new program, `*lotterytest*`, which forks two children, sets their priorities, and runs a CPU-intensive task in each, timing the results. The test assigns 80 tickets to one and 20 to the other, so the first should be scheduled 4 times as often as the second (an 80% / 20% split).

To build `*lotterytest*`, just add it to the `UPROGS` list in the `Makefile` (taking care to use a tab character for indentation rather than 4 spaces). Note that it will not compile until you have added the `*settickets*` system call.

Given two processes started at the same time, that do the same amount of work, we expect that they will finish at the same time if they have the same number of tickets. We can predict how long they should take when the CPU is divided between them at various ratios. Suppose that we have two tasks T_1 and T_2, each of which takes 5 seconds to run. Regardless of the share of the CPU each has, they will take 10 seconds total for both to finish. Running them with an equal share of the CPU will cause them both to finish at the same time. We can calculate the expected runtime of T_1 when given a certain fraction of the CPU as:

$$R(T_1, 1.0) = 5\ seconds$$

$$R(T_1, 0.5) = 10\ seconds$$

$$R(T_1, x) = \frac{1}{x} R(T_1, 1.0)$$

So from this we can compute that if T_1 is given an 80% share of the CPU, it should finish in 6.25 seconds; T_2 will still finish in 10 seconds (remember, the scheduler is just redistributing the work to be done, so running the two tasks still takes 10 seconds no matter what happens).

By running lotterytest a few times, you should be able to verify that the process with 80 tickets finishes sooner than the process with only 20.

Submitting the Assignment

You will use git to create a patch that contains your changes. First, tell git who you are:

```
$ git config --global user.email "you@example.com"
$ git config --global user.name "Your Name"
```

Then, commit your changes:

```
$ git commit --all --message="Implement lottery scheduling"
[hw4 94b0cf7] Implement lottery scheduling
7 files changed, 60 insertions(+), 7 deletions(-)
```

If you added any new files, you will also have to use git add before you run git commit.

Finally, create a patch file. The command takes an argument that specifies what code we're comparing against to make the patch; in this case I have created a git **tag** that refers to the unmodified files called `h4.unmodified`, so you should run:

```
$ git format-patch hw4.unmodified 0001-Implement-lottery-scheduling.patch
```

The command creates a file, *0001-Implement-lottery-scheduling.patch*, containing the changes you've made.

Don't try to edit the patch file after creating it. Doing so will most likely corrupt the patch file and make it impossible to apply. Instead, change the original file, commit your changes, and run git format-patch again:

```
$ git commit --all --message="Description of your change here"
[hw4 7cc4977] Description of your change here 1 file changed,
1 insertion(+), 1 deletion(-)
$ git format-patch hw4.unmodified 0001-Implement-lottery-scheduling.patch
0002-Description-of-your-change-here.patch
```

submit your `patch(es)` on NYU Classes.