Operating System

CS 370

Spring 2019

Project 4.2

XV6 Scheduler

Due Date: 11:59 pm, 13th March 2019

Must be submitted before deadline

Total Points: 100

Teaching Assistant:
Office Hours:
Graduate Assistant: Pradip Maharjan
Office Hours: MoWe 12 PM - 2 PM
Teaching Assistant: Luis Maya-Aranda
Office Hours: MoWe: 1 PM - 4 PM
TuTh: 10 AM - 12 PM

Course Web Page: http://osserver.cs.unlv.edu/moodle/

Objective: Become familiar with the **xv6** Teaching Operating System, and scheduling algorithms

Introduction:

Lottery scheduling is a probabilistic scheduling algorithm for processes in an operating system. Processes are each assigned some number of lottery tickets, and the scheduler draws a random ticket to select the next process. The distribution of tickets need not be uniform; granting a process more tickets provides it a relative higher chance of selection. This technique can be used to approximate other scheduling algorithms, such as Shortest Job Next and Fair-share scheduling.

Lottery scheduling solves the problem of starvation. Giving each process at least one lottery ticket guarantees that it has non-zero probability of being selected at each scheduling operation.*

Resources:

The following resources provide more in-depth information regarding **xv6**. They include the **xv6** reference book, source code (pdf), and a tutorial on running and debugging.

1. **xv6** Reference Book: https://pdos.csail.mit.edu/6.828/2016/xv6/book-rev9.pdf
2. **xv6** Source Code PDF: https://pdos.csail.mit.edu/6.828/2016/xv6/xv6-rev9.pdf
3. Running and Debugging Tutorial: http://zoo.cs.yale.edu/classes/cs422/2010/lec/l2-hw

Project:

Complete the following activities.

- Review chapter 0, 1, and 3 of the **xv6** reference book
- Review project 4.1
- Implement random function
- Implement fork2 system call
- Implement a lottery scheduler

Submission

When complete, submit:

- Zip the entire **xv6** folder and submit (not qemu).
- Use "zip" to compress, not "tar" or others.
- A copy of the ***zipped xv6 folder*** via the class web page (assignment submission link) by 11:59 pm 13th March, 2019.

We will check for code similarities using MOSS. Please do not download code from the internet or distribute solutions anywhere including Github, Bitbucket and any other public code publishing platforms.

Submissions received after the due date/time will not be accepted.

** https://en.wikipedia.org/wiki/Lottery_scheduling*

1) <u>Input Specification(input.txt):</u>

The input will be given in the form of a text file named ***input.txt***. The first line in the text file will contain a number, ***n***, that represents the number of processes to fork. The file will then be followed by ***n*** lines of the form where each number is a positive integer:

arrivalTime numberOfTickets numCPUBursts CPUBurst IOBurst CPUBurst ...

Processes will be specified in order of their arrival time. The number of CPU bursts will be capped at five. Recall that the number of I/O bursts is the number of CPU bursts – 1. An I/O burst length will always be followed by a CPU burst length. CPU and I/O burst times will be between 1 and 20. A larger integer specifies a longer CPU or I/O operation while a smaller integer specifies a shorter CPU or I/O operation. For simplicity, you may assume that the input file contains no errors. Number of Tickets can be between 1 and 10000.

2) <u>Modifying proc.h file:</u>

You need to add ticket field inside proc structure to save number of tickets each process gets.

3) <u>fork2 system call:</u>

You are required to write a fork2() system call which is very similar to the one already in proc.c file. fork2() will accept one argument which is number of tickets passed when a new child process is created using fork2(). Assign the number of tickets to the new process ticket field.

Inside allocproc() function assign 10 number of tickets to new process ticket field. Every new process is created by calling allocproc() but not all processes are created by using fork() or fork2(). To guarantee that every process gets some number of tickets, we assign every process 10 tickets when it is created.

Note: Remember the files that you need to modify to add a system call like in previous project.

4) <u>Random function:</u>

You are required to write a rand() function inside proc.c file. rand() is called by scheduler to determine winning ticket number.

To make a random number generator, you will create a global struct which contains an integer state variable as well as a lock for accessing the state variable. Your rand() function will acquire the lock, then read the state value. Then calculate the new random state by your rand() algorithm and write it back to the state variable. Then release the lock. You must set the initial rand state when xv6 boots. To do this, set the starting state (called a seed value) in the pinit() function in proc.c. pinit() is called in main.c when xv6 starts. Use the cmostime() function call to get the current system time and generate a seed value from the system time.

Hint: You may use the XOR shift algorithm for random number generation or any other suitably random algorithm.

5) **<u>Scheduling Algorithm:</u>**

a) Get familiar with the default scheduler() in proc.c. It is Round Robin scheduler by default. Each CPU calls the scheduler() function after setting itself up. The scheduler function never returns. The default scheduler loops across the ptable looking for a RUNNABLE process. As soon as it finds one, it performs a process context switch to run that process. The context switch starts by setting c->proc = p and ends after switchkvm(). When the process is done running, control is returned to the scheduler function which then sets c->proc = 0, and the scheduler picks up where it left off in looping across the ptable looking for another RUNNABLE process. If this loop finishes, then it releases the ptable lock, and begins the infinite for loop again. This allows other processes to modify the ptable (such as to change another process's state to RUNNABLE or allocate a new process).

b) Change ***scheduler()*** in ***proc.c*** so that it implements a lottery scheduler.

i. Find the total number of tickets given to all processes in the RUNNABLE state.

ii. Select a winning ticket by getting a random number between 0 and the total calculated in step a. *Hint: call rand()*

iii. Loop across the ptable summing all the tickets of the RUNNABLE processes only.

iv. If the sum is greater than or equal to the winning ticket number, then perform the process context switch on this process. *Hint: do not change the process context switch code.*

v. After the process returns and the scheduler sets c->proc = 0, use ***break;*** to exit the for loop so the scheduler can return to step a. and recalculate the tickets given to all runnable processes and select a new winner.

6) **<u>Modifying benchmark.c</u>**

You need to modify the benchmark.c program from previous assignment to read the number of tickets as well from the input file. Then you need to change fork() to fork2() to pass number of tickets for each process when it is created. See 1) Input Specification.

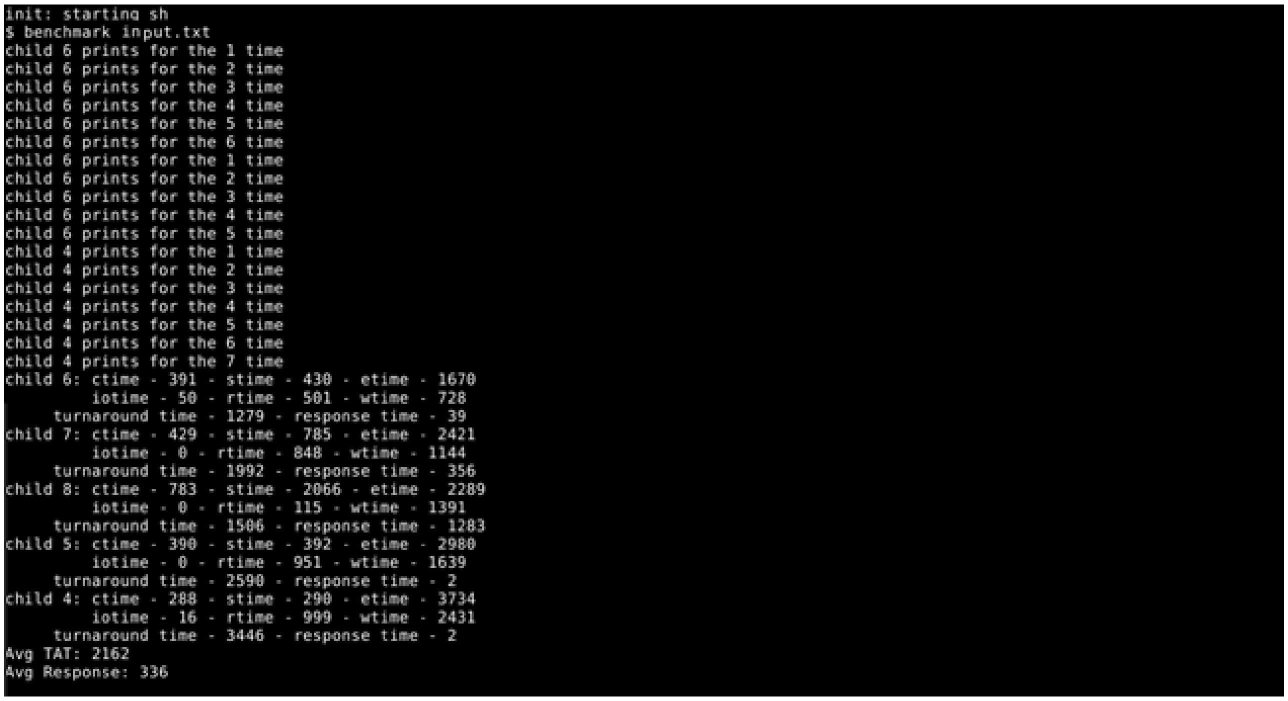
7) **<u>Testing the Scheduler:</u>**

Once all the changes have been made, build xv6 and limit it to one CPU. Run the scheduler tester from part 1 of the assignment with the same test file and compare the results to the default Round Robin scheduler. Use the queue printing that you made in assignment 4.1 to view the scheduling order of the lottery scheduler with different numbers of tickets. For example, what happens when all children have the same number of tickets or when the first process has far more tickets than all the others?

8) **Sample Input:**

```
5
0 100 2 10 7 6
100 600 1 17
100 1100 3 1 6 2 5 4
125 1600 1 15
130 2100 1 2
```

9) **Sample Output:**



```
init: starting sh
$ benchmark input.txt
child 6 prints for the 1 time
child 6 prints for the 2 time
child 6 prints for the 3 time
child 6 prints for the 4 time
child 6 prints for the 5 time
child 6 prints for the 6 time
child 6 prints for the 1 time
child 6 prints for the 2 time
child 6 prints for the 3 time
child 6 prints for the 4 time
child 6 prints for the 5 time
child 4 prints for the 1 time
child 4 prints for the 2 time
child 4 prints for the 3 time
child 4 prints for the 4 time
child 4 prints for the 5 time
child 4 prints for the 6 time
child 4 prints for the 7 time
child 6: ctime - 391 - stime - 430 - etime - 1670
         iotime - 50 - rtime - 501 - wtime - 728
     turnaround time - 1279 - response time - 39
child 7: ctime - 429 - stime - 785 - etime - 2421
         iotime - 0 - rtime - 848 - wtime - 1144
     turnaround time - 1992 - response time - 356
child 8: ctime - 783 - stime - 2066 - etime - 2289
         iotime - 0 - rtime - 115 - wtime - 1391
     turnaround time - 1506 - response time - 1283
child 5: ctime - 390 - stime - 392 - etime - 2980
         iotime - 0 - rtime - 951 - wtime - 1639
     turnaround time - 2590 - response time - 2
child 4: ctime - 288 - stime - 290 - etime - 3734
         iotime - 16 - rtime - 999 - wtime - 2431
     turnaround time - 3446 - response time - 2
Avg TAT: 2162
Avg Response: 336
```

Notice that child 4 only has 100 tickets and even though it was created first (at 288 ticks), it was the last to finish (at 3734 ticks). Child 6 has 1100 tickets and although it has the most work to do, after it was created, it was scheduled much more frequently than children 4 and 5 which preceded it. Child 6 finished before any other children, despite having the most work to do. The ticket values are essentially priority values for each process. Even though the lottery scheduler is random, the chances of a process being scheduled is based on the number of tickets a process has relative to the total number of tickets in the system. However, if two processes have the same amount of work to do, then they will both require the same amount of time slices (i.e. schedules) to complete.