Homework 3 - Processes and Scheduling

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 us. We will be checking for this!

NYU Tandon's Policy on Academic Misconduct:
https://engineering.nyu.edu/campus-and-community/student-life/office-student-affairs/policies/student-code-conduct#chapter-id-34265

Accepting the Assignment and Setting Up a Repo (Mandatory Step)

1. Head to **https://anubis.osiris.services/**
2. Press "LogIn" on the side bar, and log in with your NYU ID
3. Enter GitHub username (If not done yet)
4. Go to "Courses", and press "Join Class" with code **iloveos** to be registered to the class
5. Go to "Assignments", and accept the assignment
    a. Press "Create Assignment Repo" (This will create a repository on Github)

Working on the Assignment

For this assignment we will offer you **two options** for a platform to work on, or you can use your own platform. It's up to you what you use. It's **your responsibility** to turn it on time no matter what platform you choose. **To make this 100% clear**. Whether you choose **Anubis or Vagrant you are the only one responsible for turning it on time.** If Anubis or Vagrant doesn't work for you 5 minutes, 5 hours or 5 days before the deadline, it's not the Professor's responsibility or the TAs responsibility, it's **your responsibility.**

Using Anubis IDE (Option 1)

1. Select “Anubis Cloud IDE” from the Assignment
2. On the top menu bar, select “Terminal” and “New Terminal” to bring up a terminal
3. Complete your assignment in the project folder, and commit your changes for submission/grading

Notes:

- With auto save on, the IDE automatically commits your changes to GitHub every couple of minutes. Or type “autosave” in the shell to save your changes
    - Even with this option on, you are still **responsible** for saving your changes
- If you have any questions, don’t hesitate to post on Forum or email any of the TAs

Using Vagrant (Option 2)

1. Follow the instructions in the repo: **https://github.com/wabscale/cs3224-vm**
2. Pull the assignment repository from Github
3. Complete your assignment in the project folder, and commit your changes for submission/grading

Using your own environment (Option 3)

1. As always you are free to use your own linux environment
    a. Assignments are graded based on the Anubis IDE environment
2. Pull the assignment repository from Github
3. Complete your assignment in the project folder, and commit your changes for submission/grading

You must accept the assignment using Anubis for any of the options listed above.

Part 1 - Performance Measurement - 40 Points

In part 3 you will implement a scheduling policy. Before that, we will implement the infrastructure that will allow us to examine how policies affect performance under different evaluation metrics.

The first step is to extend the *proc struct* in proc.h by adding the following fields of type *int*: creation_time, termination_time, sleep_time, ready_time, and running_time. These will respectively represent the creation time, termination time, and the total time that the process was at one of the following states: SLEEP, READY and RUNNING.

After the creation of a new process, the kernel needs to update the process' creation time. The fields for each process state should be updated whenever a clock tick occurs (you can assume that the process state is SLEEPING only when the process is waiting for I/O). Finally, make sure you are careful when marking the termination time of the process (note: a process may stay in the ZOMBIE state for an arbitrary length of time. Naturally this should not affect the process' turnaround time, wait time, etc)

Since all this information is retained by the kernel we need to find a way to present it to the user. To do this, create a new system call, *wait_stat()*, that has the following signature for the user to call.

int wait_stat(int ctime, int* ttime, int *retime, int *rutime, int *stime)*

The arguments are pointers to integers to which the wait_stat function will assign:

- ctime: the time that the process was created
- ttime: the time that the process was terminated
- retime: the total ready time
- rutime: the total running time
- stime: the total sleep time
- Return value: the pid of the terminated process or -1 upon failure.

Helpful Notes…

- Use *argptr()* to fetch the arguments from the stack for system calls
- *wait_stat()* is similar to the *wait()* system call
- Time is measured in ticks

Part 2 - Sanity Test: Does the result make sense? - 20 Points

In this section you will add/populate the user program, sanity.c, in XV6. This program will start and immediately fork 20 child processes. Every child process runs a time-consuming function. The function is provided, and feel free to change the argument to make it run faster when testing, but make sure it is the given parameter when submitting. After the child process finishes its computation it will exit. The main process calls the *wait_stat()* function and waits until all of its children exit. For every finished child, the main process will print its pid, creation time, termination time, ready time, run time, and sleep time. Lastly, the main process prints the average wait time, average running time, average sleep time, and average turnaround time of all the processes.

Per Process Format: *printf(1, "PID[%d] Creation[%d] Termination[%d] Ready Time[%d] Running Time[%d] Sleep Time[%d]\n");*

Main Process Format: *printf(1, "Average Wait Time[%d] Average Running Time[%d] Average Sleep Time[%d] Average Turnaround Time[%d]\n");*

Part 3 - Scheduling Policy: Lottery - 30 Points

Scheduling is an important facility for any operating system. The scheduler must satisfy and optimize several conflicting objectives (discussed in class) by determining when and how a process should run.

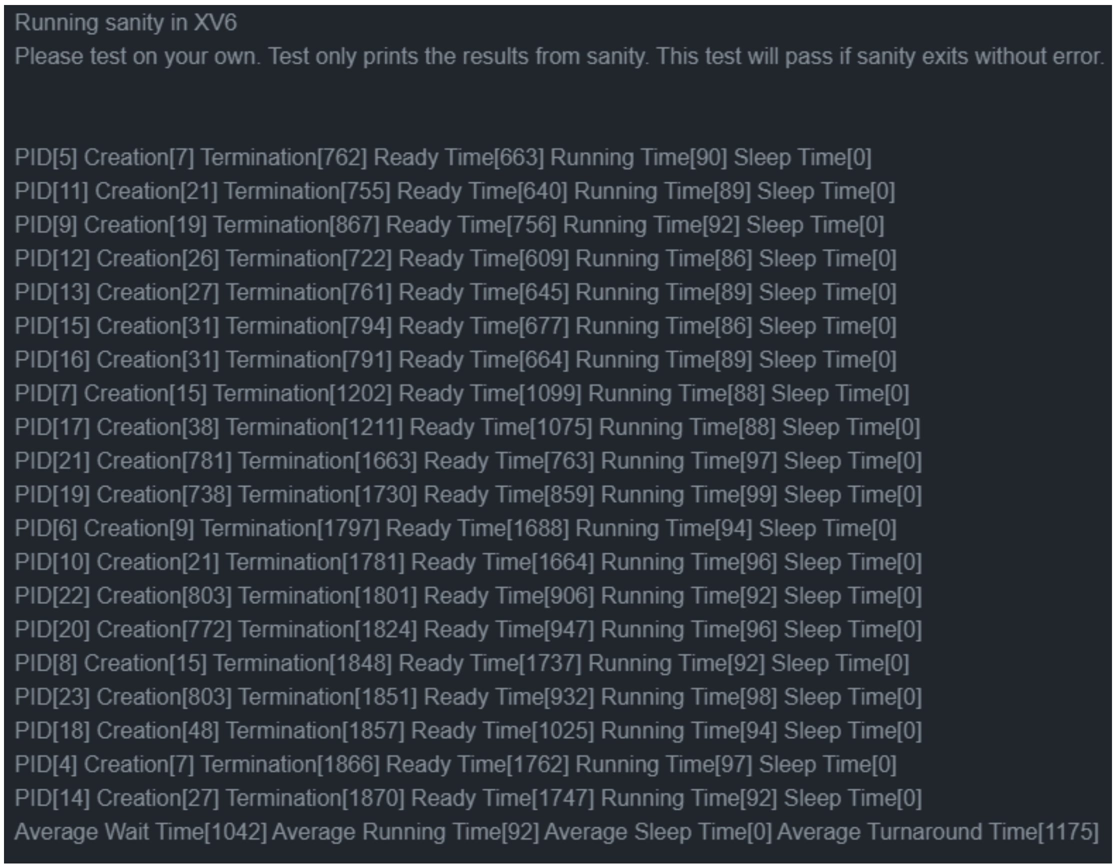
First, you need to understand the current xv6 scheduling policy. Locate it in the XV6 source code and try to answer the following questions to yourself: which process does the policy selects for running, what happens when a process returns from I/O, what happens when a new process is created, and when/how often does the scheduling take place. Your task is to remove the current scheduling policy and replace it with the *Lottery Scheduling Policy*. Processes need a new field, tickets, which can be any number from 1-999. The winning ticket is a random number between 1 and the total tickets held by the runnable processes. Tickets should be set every time the scheduler needs to pick a new process to run. This is a preemptive policy, which means the process that gets the CPU can only run for a period of time before it is preempted.

To make the scheduling algorithm as deterministic as possible for grading, an array of 256 integers is provided in “proc.c”. Everytime a random integer is needed, get the first integer that is not used from the array. If all 256 integers are used, then start from the beginning of the array again. Assuming a function called *get_random()* does the above, then to generate the tickets for the processes ...

- *process->tickets* = *get_random() % 999* + *1;* //A number between 1-999
- *winner* = *get_random() % get_total_tickets();* //A number within total tickets
    - *get_total_tickets()* can be another helper function that counts the total number of tickets held by the runnable processes.

Important Note …

The provided random integers only minimize the non-deterministic elements, as a result, Anubis will not be able to grade your assignment. A sample output from Sanity for Lottery Scheduling is shown below. Your results may not be the same, and that is fine because the results will be different based on the server that runs the code.



```
Running sanity in XV6
Please test on your own. Test only prints the results from sanity. This test will pass if sanity exits without error.


PID[5] Creation[7] Termination[762] Ready Time[663] Running Time[90] Sleep Time[0]
PID[11] Creation[21] Termination[755] Ready Time[640] Running Time[89] Sleep Time[0]
PID[9] Creation[19] Termination[867] Ready Time[756] Running Time[92] Sleep Time[0]
PID[12] Creation[26] Termination[722] Ready Time[609] Running Time[86] Sleep Time[0]
PID[13] Creation[27] Termination[761] Ready Time[645] Running Time[89] Sleep Time[0]
PID[15] Creation[31] Termination[794] Ready Time[677] Running Time[86] Sleep Time[0]
PID[16] Creation[31] Termination[791] Ready Time[664] Running Time[89] Sleep Time[0]
PID[7] Creation[15] Termination[1202] Ready Time[1099] Running Time[88] Sleep Time[0]
PID[17] Creation[38] Termination[1211] Ready Time[1075] Running Time[88] Sleep Time[0]
PID[21] Creation[781] Termination[1663] Ready Time[763] Running Time[97] Sleep Time[0]
PID[19] Creation[738] Termination[1730] Ready Time[859] Running Time[99] Sleep Time[0]
PID[6] Creation[9] Termination[1797] Ready Time[1688] Running Time[94] Sleep Time[0]
PID[10] Creation[21] Termination[1781] Ready Time[1664] Running Time[96] Sleep Time[0]
PID[22] Creation[803] Termination[1801] Ready Time[906] Running Time[92] Sleep Time[0]
PID[20] Creation[772] Termination[1824] Ready Time[947] Running Time[96] Sleep Time[0]
PID[8] Creation[15] Termination[1848] Ready Time[1737] Running Time[92] Sleep Time[0]
PID[23] Creation[803] Termination[1851] Ready Time[932] Running Time[98] Sleep Time[0]
PID[18] Creation[48] Termination[1857] Ready Time[1025] Running Time[94] Sleep Time[0]
PID[4] Creation[7] Termination[1866] Ready Time[1762] Running Time[97] Sleep Time[0]
PID[14] Creation[27] Termination[1870] Ready Time[1747] Running Time[92] Sleep Time[0]
Average Wait Time[1042] Average Running Time[92] Average Sleep Time[0] Average Turnaround Time[1175]
```

Short Answers: Answer on Anubis - 10 Pts

1. What is a major difference between threads and processes?
2. Explain the working of the shell when it executes a command: how it uses fork() and exec().
3. Briefly explain how the OS switches between processes.

Rubric

We ask that you thoroughly comment your code and compile your code without errors to receive full credit for the assignment.

Part 1 - Performance Measurement - 40 Pts

Part 2 - Sanity Test - 20 Pts

Part 3 - Scheduling Policy: Lottery - 30 Pts

Short Answers - 10 Pts