Homework 3

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 me. We will be checking for this!
NYU Poly's Policy on Academic Misconduct:
http://engineering.nyu.edu/academics/code-of-conduct/academic-misconduct
Homework 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 an exhaustive list of cases that should work.
- **TEST** your solutions, make sure they work. It's obvious when you didn't test the code.

Part 0: Setup

Same as the last homework assignment we will be using the Anubis autograder, and github for submitting work. Please use this link to get your repo:

https://classroom.github.com/a/sLWqrsgm

For this assignment, you will no longer be receiving emails. Instead you will be using a web interface to view the status of your submissions. Enter the git commit hash, and verify your netid to see the details of your submission. To find a git commit hash, you can either use the command line, or github itself. *Once you have pushed to your github repo*, you can use `git rev-parse master` to see the most recent commit hash.

```
jc@aion ‹ master › : ~/nyu/os/os3224-assignment-3
[0] % git rev-parse master
e0b313a4574f00767a1e5d67c31372fea791d575
```

Once you have your commit hash, you can navigate to the autograder site at https://nyu.cool. Enter your hash into the search bar and verify your netid to see the status of your submission. Please keep in mind that attempting to view other students submissions will be considered an act of academic dishonesty, and will be treated accordingly. As with the last homework, this is a new system in active development, so please bear with us if we need to work out any bugs.

Homework 3: Processes and Scheduling

Part 1: Performance Measurements (20 pts)

In part 2 you will implement a scheduling policy. However, before that, we will implement infrastructure that will allow us to examine how policies affect performance under different evaluation metrics.

The first step is to extend the proc struct (see proc.h). Extend the proc struct by adding the following fields to it: ctime, ttime, stime, retime and rutime. These will respectively represent the creation time, termination time and the time the process was at one of the following states: SLEEPING, READY and RUNNING. Note that these fields contain enough information to calculate the turnaround time and waiting time for each process.

After the creation of a new process the kernel will update the process' creating time. The fields for each process state) should be updated for all processes 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 which extends the wait system call and is defined as:

```
int wait_stat(int *wtime, int *rtime, int *iotime, int* status )
```

Where the three arguments are pointers to integers to which the wait_stat function will assign:
- wtime: The aggregated number of clock ticks during which the process waited (was able to run but did not get CPU)
- rtime: The aggregated number of clock ticks during which the process was running
- iotime: The aggregated number of clock ticks during which the process was waiting for I/O (was not able to run).
- status return the pid of the terminated child process or -1 upon failure.

Hints:

1. You should test your wait_stat on your own. The autograder will just check that you have implemented the syscall, and that it does not crash when running it. Your test should create some forks that simulate some computationally intensive work (30 ticks or so).

Part 2: Scheduling Policies - First Come First Serve (30 pts)

Scheduling is a basic and important facility of any operating system. The scheduler must satisfy several conflicting objectives: fast process response time, good throughput for background jobs, avoidance of process starvation, reconciliation of the needs of low-priority and high-priority processes, and so on. The set of rules used to determine when and how to select a new process to run is called a scheduling policy.

You first need to understand the current xv6 scheduling policy. Locate it in the code and try to answer the following questions to yourself: which process does the policy select 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. HINT: We have discussed this in class and you can also look at the slides.

Your task is to remove the current default policy, and replace it with a first come first serve policy. This is a non-preemptive policy. A process that get CPU runs until it no longer needs the CPU (yield/ finish/blocked).

Tests:

To test the scheduling policy, we will be using a new program called schedtest. This program will create 10 forks that essentially spinlock for 2 seconds then exit. With the default policy, we would expect the output of this program to look something like this (pids may vary):

```
$ schedtest
sched test
start 4
start 5
start 6
start 7
start 8
start 9
start 10
start 11
start 12
start 13
end 4
end 5
end 6
end 7
end 8
end 9
end 10
end 11
end 12
end 13
```

With a properly implemented first come first serve policy, each time a fork occurs the child will hold execution until it exits. This will alter the order in which the processes start and end. We would expect the output to look like this under the new first come first serve policy (pids may vary):

```
$ schedtest
sched test
start 13
end 13
start 12
end 12
start 11
end 11
start 10
end 10
start 9
end 9
start 8
end 8
start 7
end 7
start 6
end 6
start 5
end 5
start 4
end 4
```

Hints:

1. Make sure that you schedule new (forked) processes correctly.
2. You have to call the exit system call to terminate a process execution.
3. When the cpu timer goes off, where in the kernel does execution go to?

Part 3: Assembly Programing (30 pts)

Write the following python program in assembly.

```
def sumTo( n ):
    if (n <= 1):
        return 1
    return sumTo(n - 1) + n
```

Put your solution into the **sumtofunc.S** file in your repository. You will be able to test your sumto function in xv6 with the sumto command. Please leave light comments to help your grader understand your code.

Hints:

1. Use proper 32 bit calling conventions. Pass arguments on the stack, and return values through eax.
2. Don't overthink it. A correct solution should be between 20 to 25 lines.
3. You can use this website to test your code http://www.emu86.org/.

Part 4: Short Answer (20 pts)

Leave your answers to the following questions in a file called answers.txt in your repository. Answers should be short and to the point. No essays are necessary.

1. Explain the working of the shell when it executes a command and how it uses fork() and exec(). Draw a diagram and label it with process ids to show your understanding. Submit this diagram as a separate file in your repo called question1.jpg or question2.png.
2. Briefly explain how the OS switches between processes.
3. Three batch jobs, A through C, arrive at a high performance computer at the same time. They have estimated running times of 10, 6 and 2 minutes respectively. At time 3, jobs D and E arrive, which take 4 and 8 minutes respectively. For each of the following scheduling algorithms, determine the mean turnaround time. Ignore any process switching overhead.
    a. First come, first serve
    b. Shortest job first

Submit

As with the homework 2, your repo will be your submission. Push to your repo often. You have unlimited submissions until the deadline. Please use this tutorial for using git:

https://piazza.com/class/k5wsd9vc9x81es?cid=63