CST 334: Operating Systems
Dr. Glenn Bruns

Homework - Week 3

Please note! All homework is to be done on your own.

1. **Reading**. Read OSTEP chapters 5-7. You can skim section 5.4, but you will find it handy to read later as you continue work on your msh shell. Answer the following questions by editing hw3.txt, which is attached to the homework assignment on iLearn.

Note: I highly recommend spreading your reading out over the week. Some of the concepts take time to absorb.

1.1. What does wait() return when it succeeds?

a) 0

b) process ID of the terminated child

c) process ID of the parent

1.1. What happens if wait() is called by a process with no children?

a) wait() returns immediately

b) wait() never returns

c) the behavior of wait() is not defined in this case

1.1. What is the value returned by fork() to the newly created process?

a) -1

b) 0

c) the process ID of the parent process

1.1. What does function getpid() return?

a) the process ID of the calling process

b) the process ID of the parent of the calling process

c) a fresh (never before used) process ID

1.1. I/O operations of a processor (i.e., CPU) can be run when the processor is in:

a) user mode

b) kernel mode

1.1. The contents of the trap table are set when:

a) the OS boots

b) a system call is made

c) a trap handler runs

1.1. (T/F) The location of the trap table is stored in hardware

1.2. What does “preemptive scheduling” mean?

a) processes are run one at a time

b) all processes are eventually run

c) one process can be stopped to let another one run

1.1. (T/F) Code running in user mode is allowed to run a trap instruction.

1.2. (T/F) Round-robin tends to perform better than other scheduling algorithms with respect to average turnaround time.

2. **Scheduling simulator.** Answer question 1-3 at the end of Chapter 7 by running the python simulator `scheduler.py` provided by the authors of our text on the OSTEP site.

http://pages.cs.wisc.edu/~remzi/OSTEP/Homework/homework.html

Answer the questions below, rounding values to the nearest whole value. (i.e., if the answer is 120.75, write 121). In Question 3 at the end of Chapter 7, the job lengths are 100, 200, and 300.

Note that the simulator report turnaround and response times for individual jobs. In the questions below I ask for average values, so you need to take the averages of the values of the jobs.

2.3. Question 1, SJF scheduler, avg. response time _____

2.4. Question 1, SJF scheduler, avg. turnaround time _____

2.5. Question 1, FIFO scheduler, avg. response time _____

2.6. Question 1, FIFO scheduler, avg. turnaround time _____

2.7. Question 2, SJF scheduler, avg. response time _____

2.8. Question 2, SJF scheduler, avg. turnaround time _____

2.9. Question 2, FIFO scheduler, avg. response time _____

2.10. Question 2, FIFO scheduler, avg. turnaround time _____

2.11. Question 3, Round robin scheduler, avg. response time _____

2.12. Question 3, Round robin scheduler, avg. turnaround time _____

3. **msh**. Extend your msh code of last week’s homework as follows:

- as before, exit msh when ‘exit’ or ctrl-d is entered

● run Linux commands that are entered by the user. You must do this by using fork() and execvp() as shown in OSTEP Figure 5.3. After a command is executed, the prompt should be displayed as usual.

You will need to break the user input into “tokens”. The tokens are the strings in the input line separated by whitespace (spaces or tabs). The tokens should not include any whitespace. For example, on input

msh> wc -l

there are two tokens: “wc” and “-l”. Hint: you can use C library function ‘strtok’ to break the user input into tokens.

● Add a ‘help’ command that will simply display “enter Linux commands, or ‘exit’ to exit”

● Add a ‘today’ command that will print the current date in the format mm/dd/yyyy, where mm is month, dd is day of month, and yyyy is year. For example, 01/02/2018 is January 2, 2018. Hint: when I implemented this, I used Linux commands ‘time’ and ‘localtime’. Try ‘man 2 time’, and ‘man localtime’. It’s a little tricky. You must use system calls or C library functions to get the time, not user commands (look at the man page for 'man' if you don't know the difference).

It should be obvious, but your msh code must never (this week or any other) use the 'system' command, or any other command to execute shell commands. That would defeat the purpose of the assignment.

Here is some sample output to help you understand what your program should do:

```
$ msh
    msh> ls
    temp.txt  README.txt
    msh> help
    enter Linux commands, or ‘exit’ to exit
        msh> foo 1
        msh: foo: No such file or directory
    msh> today
    01/02/2018
    msh> exit
        $
```

Note the error message reported when ‘foo’ was entered. execvp() will return an error message if it couldn’t run the command entered by the user. How do you know this? execvp() will only return if there was a problem running the command that was passed to it. Here is the code I use to print an error message if execvp returns:

```
printf("msh: %s: %s\n", toks[0], strerror(errno));
```

Here toks is an array I used to store the “tokens” entered at the command line. strerror and errno are defined by Linux. For more information, look for errno in the man page for exec, and look at the man page for strerror.

Testing your code: On mlc104, the directory `/home/CLASSES/brunsglenn/cst334/hw/hw3` contains six test files test1.sh, test2.sh, …, test6.sh. Copy these to the directory where you developed your file msh.c. Each test should give exit status 0, like this:

```
$ ./test1.sh
$ echo $?
0
```

You need to run test1.sh first, as it will compile your code and produce binary file 'msh' that is used by the other tests. The directory also contains a Makefile. If you enter the command 'make', the target 'tests' in Makefile will run, causing each test to run. If you enter the command 'make clean', temporary files created by testing will be deleted.

4. **Peerwise**. You are expected to spend about an hour a week on Peerwise. Create a question based on the reading of part 1 above, or on something from lecture. Also, answer, rate, and comment on other students' questions.

Submitting: Submit your homework on iLearn as two files: your edited hw3.txt file for parts 1 and 2, and an msh.c file for part 3.

Grading: Part 1 is worth 20 points; 2 points per question. Part 2 is worth 20 points; 2 points per question. Part 3 is worth 60 points: 10 points for each of 5 unit tests, and 10 points for clean code (formatting, commenting, etc.).

Please use our Slack #assignments channel for questions to clarify what is being asked.