CSE 2431 Name: **Yuhan Zhou**

AU 25

HW 1

Due: Sunday, 9-7, at 11:30 pm to the HW 1 drop box on Carmen

PLEASE NOTE CAREFULLY: **Homework is due by 11:30 pm, but there is a one-half hour "grace period," so as long as you submit before midnight (that is, by 11:59 pm), your homework will be counted as being on time; however, if you submit at midnight or after, your assignment will be counted as one day late**. Do not wait too late to submit though; try to submit at least two or three hours early, so that if you run into any unanticipated problems, you won't submit late. "Excuses" such as "I had trouble with the system when submitting," etc. will not be accepted to waive the late penalty. I do not like to be too strict, but please be professional; please do not ask me to waive the late penalty for a reason you would not ask your boss to accept late work!

All homework answers must be typed in this file; you can submit this docx file, or convert to pdf before submitting to Carmen.

The goal of CSE 2431 homework is to give students an opportunity to understand the Unix/Linux **fork()** system call, which is used to create new processes, and also to give students experience with the coelinux system along with the command line interpreter, as well as introducing students to forking processes.

You should not discuss the homework with other students; the homework must be the student's own individual work. You may use Piazza to post any questions which you have; if you reveal what you think the correct answer to a question is, or may be, be sure to leave your post private. Guidelines for Piazza and communication are given on the Syllabus.

On the CSE Linux System, under your home directory, create a directory cse2431 (use the ***mkdir*** command in coelinux) for Systems II. Now cd to your newly created cse2431 folder/directory, and create a directory/folder called **hw1** using the command line interface (use the mkdir command). You will download several C source code files from Carmen, and create several executable files, as described below.

After creating your cse2431/hw1 folder on coelinux, start a firefox web browser, by executing the following command (the $ symbol given at the beginning of each command is the Linux command line prompt; it is NOT part of the command, and should not be typed when entering the command to run):

```
$ firefox https://carmen.osu.edu/#
```

When the web browser starts, log in to Carmen, and go to Files > Homework > HW1 folder for CSE 2431.

Download the forktest1.c file to coelinux.

Download the forktest2.c file to coelinux.

Download the forktest3.c file to coelinux.

Now, you can close the firefox web browser.

PLEASE NOTE: You are required to download the programs to coelinux, run them (as processes!) there, and answer the questions based on running them in that environment. You cannot run them in your own local Linux environment, even if you use your own Linux distribution. Now, cd to your cse2431/hw1 directory. Use:

```
$ cd ~/cse2431/hw1
```

and copy the files you downloaded (they will download to your ~/Downloads directory on stdlinux) to your cse2431/hw1 directory (These commands will copy the forktest files you downloaded to your Downloads directory to your current (./) directory, but you must be sure that you used a cd command to your ~/cse2431/hw1 directory first, as directed at the beginning of this paragraph!!):

```
$ cp ~/Downloads/forktest1.c ./

$ cp ~/Downloads/forktest2.c ./

$ cp ~/Downloads/forktest3.c ./
```

IMPORTANT: When you answer the questions below, you should put your answers **in this docx file** after the **ANSWER:** for each question below, (not on coelinux, but on the system where you have this HW1 file), and when you have all your answers, save this file, and if you wish, export your file as a pdf. Also be sure that you read and understand the file **Linux-Unix fork system call.pdf** on Carmen in the Files > Homework > HW1 folder.

Next, compile all three programs on coelinux:

```
$ gcc forktest1.c -o forktest1

$ gcc forktest2.c -o forktest2

$ gcc forktest3.c -o forktest3
```

Now, run forktest1: `$ ./forktest1`

1. When the forktest1 process runs, a single child process is created (assuming no error occurs; if you get a message saying a child was not created, run the process again till you do not get this error message). Give the statement number (see the statement numbers in the comments) which is the first statement executed by the child process when it starts running after it is created.
   ANSWER: Statement 5

Now, run forktest2: `$ ./forktest2`

2. When the forktest2 process runs, a single child process is created (assuming no error occurs; if you get a message saying a child was not created, run the process again till you do not get this error message). Both the parent process and the child process print a value for the ***num*** variable which is declared in the program code. 1) Which of the two processes (***parent or child***) changes the value of *num* from the initial value, and then prints the changed value, and 2) What is the value it prints?
   ANSWERS: **1)**Child
   2)45

3. Why does the process which prints the original value of *num* not print the changed value (In other words, explain how can the process which prints the original value can still access the original value after the value is changed by the other process; note that the correct answer has ***ABSOLUTELY NOTHING*** to do with the difference between passing a parameter by value versus passing a parameter by reference, because both parent and child print the num variable with a call to printf in main)?
   ANSWER: When the fork() is called, the operating system allocates a separate address space for the child process, so any modifications made by the child are independent of the parent. They do not affect each other. Initially, however, the address spaces are identical. This behavior is based on the copy-on-write mechanism.

Now, run forktest3: $ ./forktest3

4. For this question, you need to count the number of processes created by the code when it runs, including the parent process (which is created by the shell when you run forktest3). Including the parent, how many processes are created?
   How many processes are created (including the parent) when the program forktest3 runs? [**DO NOT JUST COUNT pids!** You need to **justify** this answer with your explanation for Question 5 below.]
   ANSWER: 8

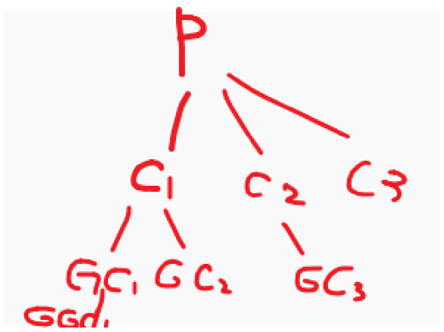
5. Explain the parent-child relationships between the processes created when the program forktest3 executes (the process which is forked by the shell to run forktest3 as a process should be called the parent process). Give a verbal (word-based) description of the relationships. Explain, as clearly as possible (and clarity counts!) how the execution of the C code in forktest3.c results in the creation of processes which have the parent-child relationships you identify. You should use these terms to describe the relationships: a child of the parent process, of course, is called a child; for a child of a child, you can use the term grandchild, and for a child of a grandchild, you can use the term great-grandchild.
   You should also number the processes of each type, starting with the first one created, or if the order is in which the processes are created cannot be determined, the numbers do not have to correspond to the order (for children, child1, child2, etc.; for grandchildren, granchild1, grandchild2, etc.; for great-grandchildren, great-grandchild1, great-grandchild2, etc.).
   Specifically, how many grandchild processes are created by child1, by child 2, etc., and how many great-grandchild processes are created by grandchild1, grandchild2, etc.? You cannot

always definitely determine the order in which the processes are created, so the numbers are not meant to refer to ordering (That is, child1 is not necessarily created before child 2, etc.)
[SAMPLE ANSWER, but *not correct for this question*]

Parent forks child1, …….
Child1 forks granchild1, …..
Child2 forks grandchild2, …..
Grandchild1 forks great-grandchild1, ….
And so on….

[See the sample answer above to know how to specify the relationships.]
ANSWER:

The first fork() creates two processes, the parent P and the child C1. The child branch executes Code1() twice, with each call performing a fork(). In the first call, C1 forks and produces GC1, and after returning both C1 and GC1 continue to the second call. In the second Code1(), C1 forks to create GC2, while GC1 forks to create GGC1, a great-grandchild. Meanwhile, the parent branch executes Code2() twice, and each call also performs a fork(). In the first call, P forks and produces C2; in the second call, both P and C2 fork, resulting in C3 and GC3. All in all, after both Code1() and Code2() complete, there are eight processes in total.



6. What is a **pid**?
ANSWER: A pid, which is called process identifier, is a unique integer assigned by the operating system to each running process. It is used by the OS to manage and distinguish processes.

7. A child process has two pids *associated with it* (Note: This **does not mean** a child process ***HAS*** two pids, but rather, it has two pids ***associated*** with it; see the getpid and getppid system calls below); what are these two pids which are associated with a child process?
ANSWER:

Its own pid, we can obtained it with getpid(). And Its parent’s pid, obtained with getppid().

8. Look at the online manual page for **ps**; that is, run the command: **$ man ps** on stdlinux. What does the ps command do?

ANSWER: It displays information about active processes, such as their process IDs, controlling terminals, CPU usage, and command names.

a. Enter and run the command: $ **ps -af**. What does ps -af do?

ANSWER: ps -af shows all processes (-a) in full-format listing (-f), including the relationship of parent and child, command arguments, and user information.

9. Look at the online manual page for the **kill** command; that is, run: *$ man kill* on stdlinux. What does **kill** do?
   ANSWER: It sends a signal to a process, usually to terminate it. In default case, it sends SIGTERM to gracefully end the process.

   a. How might the kill command be useful to a user of the system for some process which is running for the user?

   ANSWER: It allows a user to stop a process that is misbehaving, frozen, or consuming too many resources, which also means that we can do operation on the process to avoid sth.

   b. How does the kill command relate to the **ps** command?

   ANSWER: ps is used to find the pid of a process, and kill uses that pid to send a signal (like terminate) to the process. It's a good combination.

Directions for the questions below: Give a one or two sentence description, *in your own words*, of each of the following system calls:

10. fork:
    ANSWER: Creates a new process by copying the calling process. The new process is the child, and both parent and child continue execution from the point of the fork.

11. getpid:
    ANSWER: Returns the pid of the calling process.

12. getppid:
    ANSWER: Returns the pid of the parent process of the calling process.

Type your answers to the questions above into this document, save it (if you wish, you can export to pdf, but that is up to you), and submit it to the HW 1 drop box on Carmen by the due date given above.

Homework not submitted by the deadline on the due date but within 24 hours of the due date/time will be assessed a 25% late penalty of the grade for the whole homework assignment; homework is not accepted after that time (unless you have an extension from the instructor approved before the homework is due).