JORDAN HENSTROM – CS 324 – 9/21/2018 – SECTION 001 – HW3

1. Download hw3.tar.gz and un-tar/gzip it to a directory:
hw3.tar.gz Download

$ tar -zxvf hw3.tar.gz

2. Run "make" to build three executables: fork, exec, and pipeline. These are programs to that illustrate the system calls fork() and exec() and the combination of the two.

3. exec.

3a. Open exec.c and look at what it does. Give a description of its behavior.

It will print “Program <PROGRAM> pas pid <PID OF PROGRAM>. Sleeping\n”. then it sleeps for 30 seconds. After that if there was no command line argument given with the exec, it prints that there was no program to exec, and exits. Otherwise it says it’s running the given program and then calls execve with it and runs it. Lastly it exits the exec.

3b. Under what conditions will the final print statement be executed?

If the execve call fails to open a new program.

3c. Execute the exec program, instructing it to run the program "/bin/cat" with no arguments (i.e., no arguments to cat). Show the command-line you used to execute the program.

```
jordan@jordan-virtualbox:~/downloads/hw3/$ ./exec /bin/cat
Program “./exec” has pid 4570. Sleeping.
Running exec of “/bin/cat”
[] ← cursor is here
```

3d. Run the ps command in another window, such that 1) only information for the process whose PID output by 3c is shown; and 2) only the following fields are shown in the output:

user: the username of the user running the process
pid: the process ID of the process
ppid: the process ID of the parent process
state: the state of the process, e.g., Running, Sleep, Zombie
ucmd: the command executed

Use the man page for ps to learn how to do this. Run your ps command 1) during the initial sleep(); and then again 2) after the exec call (i.e., after the initial 30-second sleep is over). Show the commands you used and their output.

1.
```
jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
USER        PID     PPID    S       CMD

jordan      4700    4684    S       exec
```

2.
```
jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
USER        PID     PPID    S       CMD

jordan      4700    4684    S       cat
```

3e. What similarities and differences are there between the output of the two executions of ps, and what are the reasons for those similarities and differences?

The big difference is that COMMAND name in the first is exec and cat in the second. This is because while the process is sleeping, it's still in the exec command, but the second one is in the cat process which was taken over the pid of the exec process at that point.

(You may use ctrl-d to specify end-of-file and end the program you executed in 3c.)

3f. Run the exec program again, this time specifying that it execute a non-existent program. Show the output and explain it.

```
jordan@jordan-virtualbox:~/downloads/hw3/$ ./exec /bin/cat
Program "./exec" has pid 4570. Sleeping.
Running exec of "/bin/cat"
End of program "./exec"
```

4. fork

4a. Open fork.c and look at what it does. Give a description of its behavior.

Fork.c will first print stat it's starting a program and will display the pid. Then it will fork(), storing the result in a variable called pid. If it fails, then it prints that to stderr and exits. After that if it prints Section A and the pid, and sleeps for 30 seconds. Then if the variable pid is 0 (aka this fork is the child) it prints Section B and sleeps for 30 seconds, then says it's done sleeping and exits. If the pid was not 0 then it prints section C and sleeps twice for 30 seconds. It then says it's done sleeping and exits. After all of that it will print Section D and sleep for 30 seconds.

4b. Which section(s) (annotated "A", "B", "C", and "D") are run by the parent process and which by the child process?

A – parent and child
B – only child
C – only parent
D – neither

4c. Execute the fork program. Now run the ps command in another window, such that 1) only information for the processes whose PIDs (should be two) output by 4b are shown; 2) only the following fields are shown in the output:

user: the username of the user running the process
pid: the process ID of the process
ppid: the process ID of the parent process
state: the state of the process, e.g., Running, Sleep, Zombie
ucmd: the command executed

and 3) process ancestry is illustrated (with the "--forest" option):

Use the man page for ps to learn how to do this. Run your ps command 1) during the initial sleep(); and 2) after "Section B done sleeping" is printed (but while the program is still running). Show the commands you used and their output.

```
1.  jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
    USER        PID   PPID  S       CMD
    jordan      4803  4789  S       fork
    jordan      4804  4803  S       \_ fork

2.  jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
    USER        PID   PPID  S       CMD
    jordan      4803  4789  S       fork
    jordan      4804  4803  Z       \_ fork <defunct>
```

4d. What similarities and differences are there between the output of the two executions of ps, and what are the reasons for those similarities and differences?

They are showing the same two processes, but the second one shown is the child of the first. Additionally, the second one is shown as a defunct/zombie process

4e. With what single line of code could you change the output of the second ps command, such that the process with state "Z" is eliminated? Where would you put it? Add that line to fork.c, and re-make.

Added the line 'waitpid(pid, &returnstatus, 0);' at the beginning of process C. this cleans up the child so it doesn't become a zombie.

[NOTE: Part 4f initially had a problem with the expected output because of the line of code added in 4e. For that reason, I will give credit for output of reasonable attempts to complete problems 4f and 4g, even having used the initial version of my question. But even if you get credit on this assignment for these questions, please understand what the proper output is/should be, so you can answer properly on any test questions on this topic.]

4f. Modify fork.c. Modify the code, such that: 1) Section B has two 30-second sleep() calls in place of the original; 2) Section C has only a single 30-second sleep() call in place of the original; and 3) the line of code you added in 4e is temporarily commented out. Now repeat the exercises in 4c (replace "Section B done sleeping" with "Section C done sleeping" in the instructions).

```
1.  jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
    USER        PID   PPID  S       CMD
    jordan      5025  4789  S       fork
    jordan      5026  5025  S       \_ fork

2.  jordan@jordan-virtualbox:~/downloads/hw3/$ ps- --pid 4570 -o user,pid,ppid,state,ucmd
    USER        PID   PPID  S       CMD
    jordan      5026  791   S       fork
```

4g. Repeat the exercises associated with 4d.

This time the parent finished first so it's not displayed anymore. The child is now associating process 791 as it's parent. It is still running

4h. Modify fork.c: 1) comment out all calls to sleep(); 2) prior to the call to fork(), open the file "fork-output.txt" for writing (see fopen); 3) write "BEFORE FORK\n" to the file before the call to fork(),

"SECTION A\n" in section A, "SECTION B\n" in section B, and similarly for sections C and D; and 4) uncomment the line of code you added in 4e. Re-make, and execute the newly compiled fork. Use cat to show the contents of the file you created.

4i. Describe what you observe about the parent and child processes writing to the file.

They both write the section A string and their specific section (B, C respectively). However, they both write a 'BEFORE FORK'.

4j. Modify fork.c. Prior to the call to fork(), open a pipe (see man pipe). In section B: close the file descriptor corresponding to the reader end of the pipe; create a stream using the file descriptor corresponding to the writer end of the pipe (see man fdopen); and write "hello from Section B\n" to the newly opened stream (see man fputs). In section C: close the file descriptor corresponding to the writer end of the pipe; create a stream using the file descriptor corresponding to the reader end of the pipe (see man fdopen); read a line from the newly opened stream (see man fgets); and print the line read to stdout. Show the output of your program.

```
Section A; pid 5945
Section B; pid 5946
Section B
Section B done sleeping
Hello from Section B
Section C
Section C done sleeping
```

5. fork/exec()

5a. Modify fork.c. Take the body of exec.c and integrate it into fork.c, such that the child process created with the fork() call runs the execve() call with same arguments, as the *last* thing it does before exiting. Note that you shouldn't have to change the content of the code you're integrating, only where they are placed in the new file. Also, remove all the sleep() calls in the program. Re-make, and execute the following to show that it works: "echo foobar | ./fork /bin/cat". Show the output from running that.

```
Starting program: process has pid 5978
Section A; pid 5978
Section A: pid 5979
Section B
Section B done sleeping
Program "./fork" has pid 5979. Sleeping.
Running exec of "/bin/cat"
foobar
Section C
Section C done sleeping
```

5b. Modify fork.c. Immediately before the call to "exec()", duplicate the file descriptor associated with the stream you opened in 4h, such that the standard output of the child process goes to that stream instead (see man fileno and man dup2). Re-make, and execute the following to show that it works: "echo foobar | ./fork /bin/cat". Show the output from running that. Also, cat the contents of fork-output.txt.

```
1. CONSOLE::
Starting program: process has pid 9562
Section A; pid 9562
Section A; pid 9563
Section B
Section B done sleeping
```

Program “./fork” has pid 9563. Sleeping
Running exec of “/bin/cat”
Section C
Section C done sleeping

2. TEXT FILE::
foobar
BEFORE FORK
SECTION A
SECTION C

MAKE FILE ZIPPED WITH OTHER FILES