Chenxi Hu Biyang Fu CPSC 6240

CPSC 4240/6240 **Try It 2** **50 points**

Due date: 2/24/2020 before 9pm.

Instructions:

This homework is done in *teams of two,* or *individually*, your choice. If you are working with another student, that student must be from the same level as you are (4240 or 6240).

You are allowed to use search engines, textbook/s, lecture notes, and any other sources you wish. But you are *not* allowed to copy paste from Internet, or help other teams with their work, either by giving them hints or solutions.

You will take a number of screenshots. All screenshots should be clearly legible and illustrate without a doubt what you are doing. You can open them in an image editor of your choice and trim off the parts you do not need, just to make images smaller. Insert them when answering the question, do not submit them separately as image files. Please make space between the questions and type your answers and insert screenshots here. Please do not type in red.

What and how to submit

Please create separate folders for each exercise and name them ex1, ex2, ex3, etc. All your bash scripts and C programs should be submitted, but please remove all executable files. Then place all exercise folders in one folder named *tryit2* and package/zip the folder using tar and gzip utilities. Submit your tryit2.tar.gz file to canvas before the due date. Your writeup should reside inside the tryit2 folder in .doc or .pdf format.

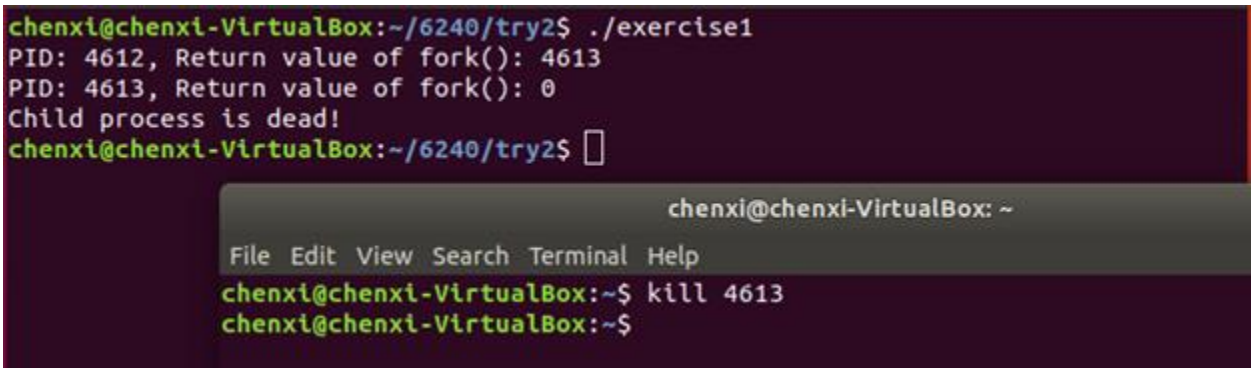
If working with one other student, please include both of your names on top of that doc or pdf file, as well as your course # 4240 or 6240. No name –> no credit.

Grading and Points

Every question indicates how many points it is worth. 4000-level and 6000-level are graded differently, with points indicated as (x/y), where x is 4240 and y is 6240.

Exercises

1. In this exercise you will create a small program using C programming language (which is a prerequisite skill for this course). Name your program *exercise1.c*. In this program you will create a child process. Please remember that this is done with fork(). Review the system call fork(). What does it return? Compile your program and let it run it in the background. Now kill the child process. You can get the child pid and kill the child in your C program using the kill system call. Look it up. How does parent find out if the child process is dead? Upon the death of the child the parent process has to print a message: “Child process is dead!” (10/ 10)



1) fork() returns 0 in the child process, and returns the PID of the child in the parent process.

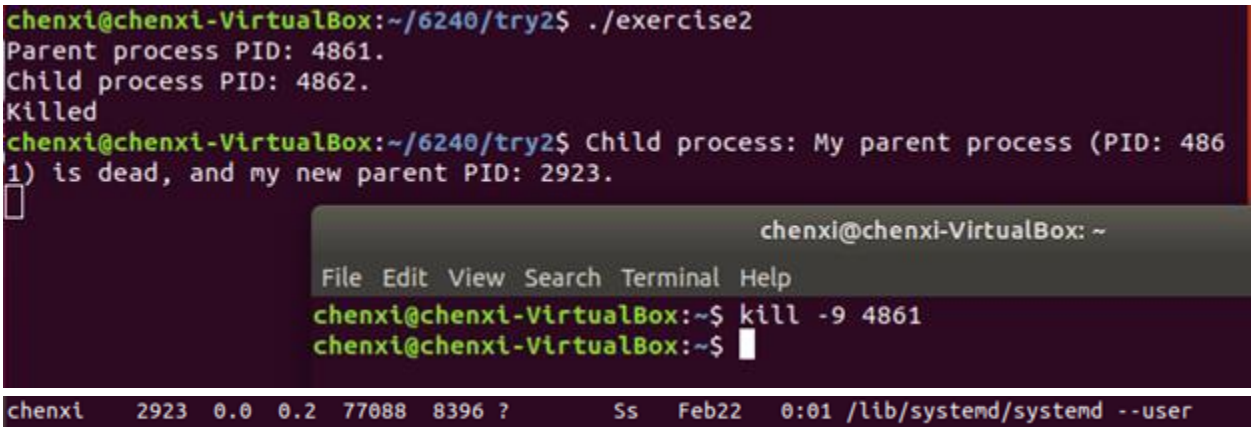
2) The parent process can find out the death of its child process by using the wait().

2. Now create a copy of the first program and modify it. Name the new version *exercise2.c.* In this version you will kill the parent (use the kill system call in your program, research how to use it). How does the child find out that/if the parent is dead? After the parent is dead, what is the parent pid of the child. Who is the child's new parent, if any? Show a screenshot that shows your child running and the new parent's id. (10/8)

1) The child process can find out the death of its parent process by check the change of the PPID.

2) After the parent is dead, the parent pid is 2923.

3) The systemd process is the child's new parent.



3. In this exercise you will write another C program named exercise3.c. You will create a process that spawns a child. Try to send the process different signals to terminate it. Your signal handler should catch the signals, but continue executing, and for each caught signal print the message

"Signal xxx was caught", where xxx is the signal that was caught. How would you terminate this program? Please include some screenshots to demonstrate what is going on. (10/8)

To terminate the program, just send the QUIT/KILL signal.

```
chenxi@chenxi-VirtualBox:~/6240/try2$ ./exercise4
Parent: Sending SIGHUP.
Signal HUP was caught.
Parent: Sending SIGINT.
Signal INT was caught.
Parent: Sending SIGSEGV.
Signal SEGV was caught.
Parent: Sending SIGBUS.
Signal BUS was caught.
Parent: Sending SIGQUIT.
chenxi@chenxi-VirtualBox:~/6240/try2$
```

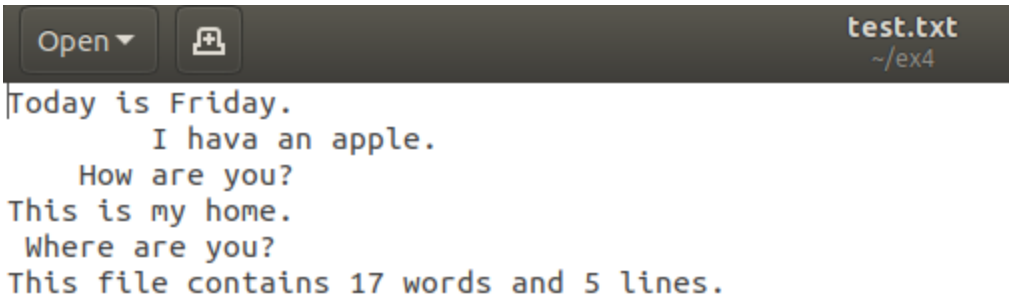
4. In this exercise you will write a file processing bash script.

- First create some text file that has 5 lines of [any] text in it.
- Your script should take two command line arguments. The first one is the name of input file, and the second is the name of output file.
- You will use your text file as input, but do not hard code the name of the file in the script, as I will be testing it with my file. Do all the necessary sanity checks to ensure that the user indeed passed two parameters and give an error message, if he did not.
- Read all the lines from input file and echo them to the screen. Also, print all of these lines into an output file that user specified as a second command line argument. The last line printed both to the stdout and to the file should be the number of words and number of lines in that file. Example is below. (10/8)

Sample output file contents:

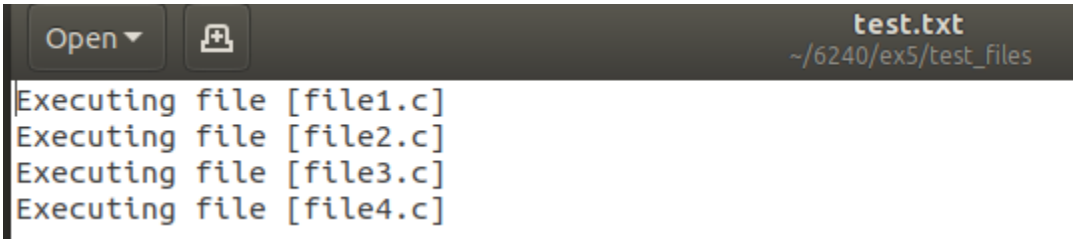
```
I shot an arrow into the air
It fell on earth, I know not where.

This file contains 15 words and 2 lines.
```

```
biyangfu@biyangfu-VirtualBox:~/ex4$ vim ex4.sh
biyangfu@biyangfu-VirtualBox:~/ex4$ chmod +x ex4.sh
biyangfu@biyangfu-VirtualBox:~/ex4$ ./ex4.sh test1.txt test.txt
Today is Friday.
        I hava an apple.
    How are you?
This is my home.
 Where are you?
This file contains 17 words and 5 lines.
biyangfu@biyangfu-VirtualBox:~/ex4$
```



5. In this exercise you will write a bash script.
   - Create a directory named *test_files* with 3 files written in C (file1.c, file2.c, file3.c) and a subdirectory named *more_files* with a C file in it as well (file4.c). Your script will reside in the *test_files* directory, along with the exercise writeup.
   - Each of the C files should print a statement: "Executing file [filename]", so it is clear which file is being executed.
   - In your script you will (probably use a loop) to look at each file and directory, compile every C file, ignore every non-C file, and step into the subdirectory and compile the C file inside it as well.
   - Because by default C executable file is called a.out, you will need to rename each of the executables to file1, file2, file3, and file4, correspondingly, for them not to overwrite each other.
   - You will then send the output from all the executables into a file that user specified as the second command line parameter.
   - Place *test_files* directory with all the contents (delete executable files first to keep submission size reasonable) into the tryit2 folder. When you are done with all exercises, you will create a gzipped tar ball to submit it. (10/8)

```
biyangfu@biyangfu-VirtualBox:~/6240/ex5/test_files$ vim ex5.sh
biyangfu@biyangfu-VirtualBox:~/6240/ex5/test_files$ chmod +x ex5.sh
biyangfu@biyangfu-VirtualBox:~/6240/ex5/test_files$ ./ex5.sh test.txt
Executing file [file1.c]
Executing file [file2.c]
Executing file [file3.c]
Executing file [file4.c]
biyangfu@biyangfu-VirtualBox:~/6240/ex5/test_files$
```



Graduate students

6. Research how to use exec() along with fork() and write a simple program where a parent spawns a child. The child should be executing the program that was passed to it as a command line parameter. Demonstrate that parent and child are executing different code. In parent routine you will print “executing parent routine” and in child – “executing child routine”. (__/ 6);

```
chenxi@chenxi-VirtualBox:~/6240/try2$ ./exercise6
executing child routine.
total 76
drwxr-xr-x 2 chenxi chenxi 4096 Feb 24 11:46 .
drwxr-xr-x 4 chenxi chenxi 4096 Feb 24 09:38 ..
-rwxr-xr-x 1 chenxi chenxi 8480 Feb 23 18:34 exercise1
-rw-r--r-- 1 chenxi chenxi  312 Feb 23 18:34 exercise1.c
-rwxr-xr-x 1 chenxi chenxi 8440 Feb 23 11:21 exercise2
-rw-r--r-- 1 chenxi chenxi  497 Feb 23 18:16 exercise2.c
-rw-r--r-- 1 chenxi chenxi 1624 Feb 23 19:08 exercise3.c
-rw-r--r-- 1 chenxi chenxi  760 Feb 23 22:43 exercise4.sh
-rwxr-xr-x 1 chenxi chenxi 8488 Feb 24 11:46 exercise6
-rw-r--r-- 1 chenxi chenxi  367 Feb 24 11:46 exercise6.c
-rw-r--r-- 1 chenxi chenxi   10 Feb 23 21:35 output_file
-rw-r--r-- 1 chenxi chenxi  159 Feb 23 22:13 text1.txt
-rw-r--r-- 1 chenxi chenxi  207 Feb 23 22:44 text2.txt
executing parent routine.
chenxi@chenxi-VirtualBox:~/6240/try2$
```