COMP 2560 Fall 2020

Lab 8

The purpose of this lab is to practice more signal processing.

As in Lab 7, there is one question that requires submission at the end of your lab session. The rest of the questions are due on Nov. 29, 11: 59 PM.

Please put in your name and lab section as comments on top in your source code.

1. We studied the code **pcs2.c** in class. Its main function is shown below:

```
    …..
    int main(int argc, char *argv[]){
            pid_t pid;

            if((pid=fork())>0){ //parent
                    sleep(1);
                    while(1){
                            printf("Parent is running\n");
                            kill(pid, SIGUSR1);
                            signal(SIGUSR1, action);
                            pause();
                    }
            }
            else  //child code
                    while(1){//child
                            signal(SIGUSR1, action);
                            pause();
                            printf("Child is running\n");
                            kill(getppid(), SIGUSR1);
            }
}
```

You could see the four bolded statements in the parent process and the child process, respectively, as shown in the table below:

Parent process	Child process

kill (pid, SIGUSR1)
….
pause()
	pause();
	….
	kill(getppid(), SIGUSR1);

Pay attention to the order of the four statements in the parent and child processes. In the parent process, kill(..) is called first, then pause(). However, in the child process, pause () is called first, then kill (). What if I change the order of the invocations of these four functions in parent and child processes, respectively, as shown below.

	In the Parent process	In the Child process

Original case	**kill (pid, SIGUSR1)**
	….
	pause()
		pause();
		….
		kill(getppid(), SIGUSR1);

Case 1	**kill (pid, SIGUSR1)**
	….
	pause()
		kill(getppid(), SIGUSR1);
		….
		pause();

Case 2	**pause();**
	….
	kill (pid, SIGUSR1);
		pause();
		….
		kill(getppid(), SIGUSR1);

Case 3	**pause();**
	….
	kill (pid, SIGUSR1);
		kill(getppid(), SIGUSR1);
		….
		pause();

Make changes to **pcs2.c** to demonstrate case 1, 2, and 3, nothing else changes in the original **pcs2.c** file except the calling order of the kill (…) and the pause(…) functions in the parent and child process as shown above in the table. Name your files as “pcsc1.c” (for case 1), “pcsc2,c” (for case 2), and “pcsc3.c” (for case 3).

Submit all three files (pcsc1.c, pcsc2.c, and pcsc3.c) and a text file explaining what the output is for each case and why. No video and script files submissions needed for this question.

2. Although we did not spend time in lecture to study signals **SIGCONT** and **SIGSTOP**, it is not hard at all to see how these two signals work. Read the brief descriptions on these two signals in textbook on page 317 and 320. Then follow the steps below to create your own program to observe how these two signals work.

   a) In the main function, fork a child process.

   b) In the child process, using an infinite loop to print out something. For example, you can print out the value of an int variable and each iteration increase the value of this int variable. You may also optionally sleep for a second or two in the loop to slow down the loop.

   c) In the parent process, sleep for a few seconds first, then send the **SIGSTOP** signal to the child process.

   d) In the parent process, sleep for a few more second, then send the **SIGONT** signal to the child process.

   e) Lastly, in the parent process, you can send the **SIGTERM** signal to the child process to terminate it.

   When running your program, observe what happens at step c) and d). Is the child process stopped and then continued upon the receipt of the signals?

   Submit your source code and the script files showing how you compile and run your program. Your source code should be commented at least for the above steps a) to e) so that we know how you implemented them. No video submission is needed.