CSCE 3600: Systems Programming

Recitation Assignment 6 – Parent-Child Processes

Available: Week 8 Due: Week 9

The purpose of this recitation is to write two short C programs showing the parent-child process relationship in Linux using the `fork()` system call. Specifically, we will look at how a zombie and orphan process are created as well as handled by the operating system.

1. Zombie Process

Recall that when a child terminates, but its parent is not currently waiting, the terminated child process becomes a zombie process. Use the following algorithm and specific instructions to develop a short, simple C program called **`rec06A.c`** to create a zombie process.

a. Add the include directives for the `stdlib.h` and `unistd.h` libraries.

b. Declare a variable called `pid` with the data type `pid_t` and invoke the `fork()` system call, assigning the result to `pid`.

c. Write an `if – else if – else` branching statement to handle the child process, parent process, and error case, respectively.

   i. In the child process, exit immediately.

   ii. In the parent process, sleep for 5 seconds, then issue the `system()` system call with "**ps -e -o pid,ppid,stat,user,cmd | grep $USER**". This prints out the current PID, parent PID, status, user ID, and command process information.

   iii. In the error case, call the `perror()` system call with "fork error".

Now, compile and run your program. After 5 seconds, you should see the result of the `ps` command. Specifically, look for the process with the Z+ status. This is an indication that it is a zombie process. This process should also be the child process of your parent `./a.out` executable that contains the `<defunct>` indicator.

Now, on the command line, enter the `ps -u $USER` command to see your existing processes. You should notice that this zombie process no longer exists as it has been reaped by the `init/systemd` process.

You will turn in this **`rec06A.c`** file to Canvas.

2. Orphan Process

Recall that a child process whose parent has terminated is referred to as an orphan process. This means that the child is still executing, but its parent has terminated and that some process is then needed to handle the child's exit status. Use the

following algorithm and specific instructions to develop a short, simple C program called **`rec06B.c`** to create an orphan process.

a. Add the include directives for the `stdio.h`, `stdlib.h` and `unistd.h` libraries.

b. Declare a pointer to a `char` called `name`.

c. Declare a variable called `pid` with the data type `pid_t` and invoke the `fork()` system call, assigning the result to `pid`.

d. Write an `if-else if-else` branching statement to handle the child process, parent process, and error case, respectively.

   i. In the child process, print "child: %d started\n" using the `getpid()` system call for the %d. Then, print "child: parent = %d\n" using the `getppid()` system call for the %d. Now, print another child status (use "child: …") to indicate the child is about to go to sleep. Then, sleep for 20 seconds. Finally, print a child status to indicate that the child just woke up.

   ii. In the parent process, print "parent: %d started\n" using the `getpid()` system call for the %d. Then print "parent: parent = %d\n" using the `getppid()` system call for the %d.

   iii. In the error case, call the `perror()` system call with "fork error".

e. Outside of the `if-else if-else` branching statement, use the ternary operator `(condition) ? (if_true) : (if_false)` to assign "child" to the `name` character pointer if `pid` is 0 and "parent" otherwise. Then, print "%s: terminating…\n" using the `name` character pointer for the %s.

Now, compile and run your program. After your program initially runs, but before the child process has terminated, enter the `ps -ef|grep $USER` command to see your existing processes. You should notice your child process is still running, but has 1 for its parent process – this means that the `init/systemd` process has taken over as the parent (since our parent terminated) and will handle the child once it has terminated.

You will turn in this **`rec06B.c`** file to Canvas.

REQUIREMENTS:

- No comments are required for this recitation assignment, except for your name at the top of each program.
- Your script will be graded based largely on whether it works correctly on the CSE machines (e.g., `cse01`, `cse02`, …, `cse06`), so you should make sure that your script runs on a CSE machine.

- Although this assignment is to be submitted individually (i.e., each student will submit his/her own source code), you may receive assistance from your TA and even other classmates. Please remember that you are ultimately responsible for learning and comprehending this material as the recitation assignments are given in preparation for the minor assignments, which must be completed individually.

SUBMISSION:

- You will electronically submit your programs, `rec06A.c` and `rec06B.c`, to the **Recitation 6** dropbox in Canvas by the due date and time. No late recitation assignments will be accepted.