CCPS590 Lab 3 – Parent and Child Processes

Preamble

In this lab you will use the **`fork()`** function to spawn child processes from a parent. You will also explore the relationship between parent and child PIDs.

Lab Description

1) Compile pids1.c into executable named pids1. Inspect the code, try and figure out what's happening. The **`fork()`** function creates a child process. Both parent and child continue executing *from that point*. Each calls **`getpid()`**, each calls **`getppid()`**, each enters the **`if/else`** ladder, etc. Run pids1 to see the Process IDs of the parent and child processes (It will take a few seconds to complete, note the **`sleep()`** function call.

Open another shell window. In shell 1 run program pids1, and while pids1 is sleeping, quickly, In shell 2, run the command: **`ps -u <userid>`** where <userid> is your login. For example, I do:

```
ps -u aufkes
```

Check that the process IDs shown with the **`ps`** command in shell 2 are the same as printed by the parent and child processes running in shell 1. Note: You must be quick enough. You must do the **`ps`** in shell 2 while the processes in shell 1 are still alive.

2) Consider program pids2.c. It is an exact copy of pids1.c except that the lines…

```
pid = getpid();
ppid = getppid();
```

…are moved inside the **`if/else`** ladder. The lines are duplicated for the parent and child, so, semantically, it does not alter the program. Compile into pids2 and run.

Notice that when you run pids2, the child process often thinks its parent's PID is something *other* than what was printed by the parent. The child might print 1 for its parent's PID (if you're on an SCS moon), or it might print something else entirely.

Why would the child report its parent's ID as something different?

3) Program pids3.c is a copy of pids2.c but with all the **`sleep()`** calls removed. Compile and run. If you run it enough times, you will notice it usually behaves like pids1 (child gets correct parent PID), but sometimes it behaves like pids2 (child thinks its parent's PID is something else). It is unpredictable. Why?

An important aside: In this course, you will hear the word *unpredictable* a lot. This is ***NOT*** the same as *random*! Many things are hidden from us within the OS kernel. These are all still deterministic, even if we cannot influence or see their inner workings.

4) Copy pids3.c to pids4.c. Use the man pages (or Google) to see what the **`wait()`** system call does. Use the **`wait()`** function to guarantee pids4.c will get its birth-parent's PID by not allowing the parent to die while the child is still running. Your code in pids4.c should not include any sleep() calls whatsoever. It should only use wait() to guarantee that the child gets the correct parent PID every time.

Submission

For this lab you will submit only one file - your code pids4.c. At the top of pids4.c, include a block comment containing written answers to questions 2 and 3. Thus, your code must compile and run out of the box even with the written answers included.

Labs are to be submitted ***individually***! Make sure your code and written answers are formatted cleanly and are easy to read.