

PROGRAMMING ASSIGNMENT #1

CS 480-2

due Tues., Oct. 28, 2008, 6:30 PM

Write a Unix program in C which uses `fork` to start a child process.

In the parent process, print the message:

```
Parent: My PID is xxxx, my parent's PID is yyyy and my child's PID
        is zzzz.
```

Then invoke the command `ps -fp` from your program to show that your output is correct.

Perform the same commands in the child process, replacing the word “parent” by “child.”

After printing this information, the child process should sleep for 1 second, then print the following message and exit:

```
Child: Child is awake.
```

The parent process should wait for the child process to complete, then print the following message and exit:

```
Parent: Child finished.
```

Note that each message is labeled according to the process that printed it, parent or child. When multiple messages are printed from the same process, they should appear in the order in which they are printed. Each message should be single-spaced with a blank space between messages.

The interlacing of messages printed by different processes will depend on the timing of your program (e.g., the quantum size of the round-robin algorithm) and does not have to match the sample output below. For example, the messages could appear in the order child, parent, parent, child.

A sample output:

```
child : My PID is 15317, my parent's PID is 15316 and my child's PID is 0.
parent: My PID is 15316, my parent's PID is 15187 and my child's PID is 15317.
child : Issuing command: /bin/ps -fp 15317,15316
child:
  UID  PID  PPID  C   STIME TTY      TIME CMD
freedman 15317 15316  0 20:24:54 pts/3    0:00 ./fork4.o
freedman 15316 15187  0 20:24:54 pts/3    0:00 ./fork4.o
parent: Issuing command: /bin/ps -fp 15317,15316
parent:
  UID  PID  PPID  C   STIME TTY      TIME CMD
freedman 15317 15316  0 20:24:54 pts/3    0:00 ./fork4.o
freedman 15316 15187  0 20:24:54 pts/3    0:00 ./fork4.o
child : Child is awake
parent: Child finished
```

Administration:

Turn in a printed listing of your program and the output. Also mail a copy of the source program to me as an email attachment with the name `xxxx.c`, where `xxxx` = the first four letters of your last name. The subject line of your email should be “480 program 1”.

In order for your program to be on time, you must turn in the program listing and output by class time on the due date, and the email copy must be received by then as well. A listing that does not match the email copy or output that was not generated by your program may be considered evidence of cheating. Points will be deducted for late programs as specified in the syllabus, i.e., 1 point per hour.

For full credit, you must follow the above rules and also the formatting rules for homework, including the rules on names, stapling, and text size. Code and output should be printed in a non-proportional font, e.g. Courier.

Programs must be consistently indented and commented so that a reasonable person can understand them, including the use of consistent and descriptive variable names.

As specified in the syllabus, programs must run on `turing.cs.niu.edu`.

Function notes:

You can find your own PID with the `getpid()` function and your parent's PID with the `getppid()` function.

You can invoke a command, e.g. `ps`, using the `system` system call. If you don't get the output you expect, try running the same `ps` command yourself from the shell; the fault may be in the `ps` command rather than in the call to `system`.

You can use the `man` command to learn about the arguments for any command and the libraries it needs.

Interleaving notes:

You need to ensure that output from each process comes out in sequence and that individual lines of output from one process are not interleaved with lines from the other process.

When we discussed the “readers and writers” problem in class, we did not need to discuss the size of the data structures the readers and writers were using. With batch jobs printing to a print spooler, the chunks are usually jobs (i.e., complete processes). But what size chunks does the Unix internal print process use, individual characters, lines or system calls? In fact it uses a bufferful of data.

Therefore to ensure that the output of each process comes out in sequence, you need to use unbuffered output. Otherwise output produced by `system`, which starts a separate subtask, could come out while previously printed output is still in a buffer. Use `setbuf(stdout, NULL)` at the beginning of your program to get unbuffered output.

With unbuffered output, the system will interleave print output at the line level instead, using the newline character (`\n`) to define a line. Therefore, the second step is to make sure that each output contains only one line. (This will work as long as the output of an individual print command does not exceed the buffer size.) You can add “`| cat`” after the `ps` command to convert its output to one line. You can also use `sprintf` and `echo` to convert multiple outputs to a single line.

This is a cheap solution, suitable for homework or for a one-time program, but not good enough for a real environment. There are a number of other cheap solutions available, such as making one of the processes wait a second until the other one has printed. In a later program you will see a more robust solution.