EECE.4810/EECE.5730: Operating Systems

Spring 2017

Programming Project 1

Due **3:15 PM**, **Wednesday, 4/26/17**

1. Introduction

This project reviews fundamentals of multiprocessing covered earlier in the semester. You will write a program that generates multiple processes using the UNIX `fork()` function, handles their return using `wait()`, and generates output from both parent and child processes to demonstrate the correctness of your approach.

This assignment is worth a total of 75 points. The given grading rubric in Section 3 applies to students in both EECE.4810 and EECE.5730.

2. Project Submission

Your submission must meet the following requirements:

- You should submit a single file named `OS_program1.c`.
- Programs must be submitted via e-mail sent to both Dr. Geiger (Michael_Geiger@uml.edu) and Peter Mack (Peter_Mack1@uml.edu)
- You may work in pairs (groups of 2) on this assignment. If you work in a group, please list the names of both group members in a header comment at the top of your submission, as well as in the email you use to submit your code.

3. Specification and Grading Rubric

As noted above, the program should use `fork()` to start each new process and `wait()` to collect the return status (and possibly check the PID) of each. *At present, the assignment does not contain any sample output, but I will update it shortly with outputs to demonstrate each of the cases in the rubric below.*

Your assignment will be graded according to the rubric on the following page; partial credit may be given if you successfully complete part of a given objective. Please note that you should not submit separate files for each objective—your sole submission will be judged on how many of the tasks below it accomplishes successfully. The rubric may also be used as an outline for developing your program—for example, first write a program that accomplishes objective A, then modify it to accomplish objective B, and so on.

In the rubric, objective A is the base case—you cannot complete any of the other objectives without completing that one, and the number of points listed with that objective is essentially the minimum you can earn with a working program. For each additional objective, the number of points is an additional number

3. Specification and Grading Rubric (continued)

Objectives and grading rubric:

A. (10 points) Your program creates a single child process, printing at least one message from both the parent and child process indicating the PIDs of those processes. Your parent process should wait for the child to terminate and print a message once the child has completed.

B. (+5 points) Your program creates multiple child processes without using a loop, printing messages at the start and end of each process as described in part A.

C. (+10 points) Your program uses a loop to create ten (10) child processes, printing messages at the start and end of each process as described in part A.

D. (+5 points) Your program is almost identical to part C, but the number of child processes is based on a command line argument passed to your executable. (For example, if your executable is named "`proj1`", executing the command `./proj1 6` will run a version of your program that creates 6 child processes. Assume the maximum number of child processes is 25.

E. (+10 points) Your program is almost identical to part D, but the program is able to discern when each of its child processes completes and print an appropriate message. (For example, when the first child process completes, print a message saying, "`Child 1 (PID` *xxxxx*`) finished`", where *xxxxx* would be replaced by the actual PID.

F. (+5 points) Your program is almost identical to Part E, but each child process starts a new program, replacing the address space of the parent process with that of the new program. For this part, all child processes should start the same program.

G. (+5 points) Your program is almost identical to Part F, but each child process starts one new program from a set of five possible new programs. Source code (and executables, if the web server allows) for the new programs will be posted shortly.

4. Hints

This section will be expanded in the coming days with more useful information; at this point, my primary goal is to allow you to start the assignment as soon as possible!

Useful functions: The multiprocess examples covered in Lectures 2 and 3 should serve as a starting point for your program. The following additional functions may be useful:

- `pid_t getpid()`: Returns the process ID of the currently running process.
- `int atoi(char *str)`: Converts `str` to the corresponding integer value—for example, `atoi("33") = 33`.
- `int sprintf(char *str, const char *format, … )`: Prints the string specified by `format` and the following arguments to the string `str`. For example, if `x = 7`, `sprintf(s, "x = %d", x)` writes the string `"x = 7"` to the string `s`.

Makefiles: On the Linux machines in Ball 410, you should use the C compiler GCC to compile your code. Repeated compilation is most easily done using the `make` utility, which requires you to write a makefile for your code. Reference material for writing makefiles is easily found online; I found the following website to be a good basic makefile introduction, which is all you should need for this assignment:

http://www.cs.colby.edu/maxwell/courses/tutorials/maketutor/