CSCI 1730 Breakout Lab 10

Unix System Calls

Learning Outcomes

- Design and implement programs that use system calls and signal handling (fork, execve, wait, getpid, etc.) and run concurrently.

Problem / Exercise

WARNING: DO NOT abuse the fork system call for this lab!

The purpose of this lab is to get experience using the C programming language to make UNIX system calls to fork, execve, and wait. You will write a C program that uses A SINGLE CALL TO fork to split itself into a parent process and a child process. The parent process will wait for the child process to execute a `ps` command, and once the child is finished, the parent process will execute a countdown program (countdown.c, which can be download from eLC) as shown in the examples in the Examples section.

Requirements

1. Read through the instructions and examples in this document and chapter 8 in your Advanced Programming in the UNIX Environment, Third Edition (Stevens & Rago) textbook.
2. Your program must use A SINGLE CALL to fork. Points will be deducted if you call fork more than once (do not place it inside a loop nor any other repetition statement). Abusing the fork system call may result in a conduct violation for our course, which may have heavy consequences to your final grade in our course. If the call to fork fails in your program, then your program should print (to standard output) "The call to fork failed." and terminate gracefully.
3. Your program must use two calls to execve. The first call to execve will be in the child's process code to execute `ps` (use the absolute path to `ps` on a vcf node). The second call to execve will be in the parent's process code to execute the countdown program.
4. Your program must use a single call to wait to make the parent process wait until the child process finishes before it executes its code for countdown. Thus, your program's I/O must be in the same order as the examples shown in the Examples section.
5. Your program must use two system calls to getpid: one call to get the parent's PID and one call to get the child's PID, and your program should display those PIDs as shown in the examples in the Examples section.
6. Your program CANNOT use a call to `system`. Your program must use fork, execve, and wait as aforementioned. If your program uses a call to `system`, then you will receive a grade of zero for this assignment.
7. Implement this lab in a C source file called `lab10.c`, and use our provided Makefile to create an executable called `lab10.out`. Do not modify our Makefile. Your source code must compile correctly (no warnings) without any modifications to our Makefile.
8. Your program's I/O must adhere to the instructions in the Examples section.
9. You may assume valid input for this lab, which will be a single integer greater than 0 inputted as the second command line argument.
10. Do not modify the provided countdown.c program.
11. If a function call fails in your program, then your program should print (to standard output) a reasonable error message and terminate gracefully.

Examples

After you finish implementing the `lab10.c`, the I/O from running `lab10.out` should be similar to what is shown below (PIDs and output from `ps` may differ; however, the rest of the output should match our examples). The parent's PID and child's PID must appear in the output of the `ps` command. The output of your program must be in deterministic order, as shown in all of the examples below, every time your program is run. The parent process must wait for the child process to complete before the parent process starts executing the countdown program.

```
./lab10.out 5
The parent's PID is 21663.
The parent is now forking.
The child's PID is 21664.
The child is executing ps.
  PID TTY          TIME CMD
21663 pts/495  00:00:00 lab10.out
21664 pts/495  00:00:00 ps
23550 pts/495  00:00:00 bash
The parent waited patiently for its child to complete.
The parent is executing ./countdown.out 5.
Starting the countdown!
5
4
3
2
1
Countdown finished.
```

```
./lab10.out 3
The parent's PID is 23358.
The parent is now forking.
The child's PID is 23359.
The child is executing ps.
  PID TTY          TIME CMD
23358 pts/495  00:00:00 lab10.out
23359 pts/495  00:00:00 ps
23550 pts/495  00:00:00 bash
The parent waited patiently for its child to complete.
The parent is executing ./countdown.out 3.
Starting the countdown!
3
2
1
Countdown finished.
```

```
./lab10.out 6
The parent's PID is 24026.
The parent is now forking.
The child's PID is 24027.
The child is executing ps.
  PID TTY          TIME CMD
23550 pts/495  00:00:00 bash
24026 pts/495  00:00:00 lab10.out
24027 pts/495  00:00:00 ps
The parent waited patiently for its child to complete.
The parent is executing ./countdown.out 6.
Starting the countdown!
6
5
```

```
4
3
2
1
Countdown finished.

./lab10.out 10
The parent's PID is 24605.
The parent is now forking.
The child's PID is 24606.
The child is executing ps.
  PID TTY          TIME CMD
23550 pts/495  00:00:00 bash
24605 pts/495  00:00:00 lab10.out
24606 pts/495  00:00:00 ps
The parent waited patiently for its child to complete.
The parent is executing ./countdown.out 10.
Starting the countdown!
10
9
8
7
6
5
4
3
2
1
Countdown finished.
```

1 C Program

1.1 General Requirements

1. Place the files for this assignment in a directory called `lab10`.

2. Your program's source file(s) must compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`).

3. Place your C source code for this assignment in a file named `lab10.c`. See the Examples section to see what your C program should output. Compile your C program using the provided Makefile for this assignment. Do NOT modify any commands, targets, or dependencies that are associated with compiling source code in the provided Makefile for this assignment. If there are any warnings or errors from the compiler, then you should fix them before you submit this assignment to us for grading.

4. Your program cannot contain a memory leak when executed. You are responsible for using valgrind to test your program to ensure it does not contain a memory leak, and valgrind should be run for every test case for your program. A program that contains one or more memory leaks (definitely, indirectly, or possibly lost) identified by valgrind will lose points.

5. The only include directives that you are authorized to use in your source code for this assignment are listed below.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdbool.h>
#include <sys/wait.h>
#include <unistd.h>
```

You CANNOT use any other include directives anywhere in your source code.

6. All written and verbal instructions stated by the teaching staff (lecture instructor, lab instructor(s), etc.) must be followed for this assignment. Failure to follow instructions may result in losing points.

1.2 Coding Style Requirements

1. All functions must be commented. Comments must include a brief description of what the function does, its input(s), its output, and any assumptions associated with calling it. If your function has a prototype and an implementation separated into different parts of your source code, then you only need to put the comments for that function above its prototype (there is no need to comment both the prototype and implementation; commenting the prototype is sufficient).
2. All structs, unions, and enums must be commented.
3. All global variables and static variables must be commented.
4. All identifiers must be named well to denote their functionality. Badly named identifiers are not allowed. For example, identifiers like a, aaa, b, bbb, bbbb are bad names for identifiers.
5. Every line of source code must be indented properly and consistently.

1.3 README.txt File Requirements

Make sure to include a `README.txt` file (use a .txt file extension) that includes the following information presented in a reasonably formatted way:

- Your First and Last Name (as they appear on eLC) and your 810/811#
- Instructions on how to compile and run your program. Since we are using a Makefile for this assignment, then these instructions should pretty simple based on the provided Makefile.

2 Submission

Submit your files before the due date and due time stated on eLC. Submit your files on eLC under the Assignment named Lab 10. Submit only the following files.

1. All source files required for this lab: lab10.c
2. A README.txt file filled out correctly

Do not submit any compiled code. Also, do not submit any zipped files or directories. Do not submit a Makefile. We will use our own copy of the provided Makefile to compile your program. We only need the files mentioned above with the file extensions aforementioned.

3 Breakout Lab Attendance: 3 points

Breakout lab attendance is required for this assignment, and it is worth three points. Students must physically attend the entire period of the breakout lab section that they registered for during the week of this assignment to earn these three points for attendance. If you complete the assignment before the end of your breakout lab period during the week of this assignment, then you are still required to attend the entire breakout lab period to earn attendance points, and you may use this time for independent study for the next exam in this course.

4 Grading: 7 points

If your program does not compile on odin or a vcf node using the provided Makefile and using the required compiler for this course (`gcc version 11.2.0`), then you'll receive a grade of zero on this part of the assignment. Otherwise, your program will be graded using the criteria below.

Program runs correctly with various test cases on odin	7 points
README.txt file missing or incorrect in some way	-1 point
Not submitting to the correct Assignment on eLC	-1 point
Late penalty for submitting 0 hours to 24 hours late	-2 points
One or more compiler warnings	-2 points
Not adhering to one or more coding style requirements	-2 points
Valgrind identifies a memory leak (definitely, indirectly, or possibly lost)	-2 points
Not submitting this assignment before its late period expires	-7 points
Program does not fork, execve, or wait as aforementioned	-7 points
Penalty for using a call to `system`	-7 points
Penalty for not following instructions (invalid I/O, etc.)	Penalty decided by grader

You must test, test, and retest your code to make sure it compiles and runs correctly on odin or a vcf node with any given set of valid inputs. This means that you need to create many examples on your own (that are different than the aforementioned examples) to ensure you have a correctly working program. Your program's I/O must match the examples given in the Examples section. We will only test your program with valid sets of inputs.