CPE 345: Operating Systems Laboratory

Lab01

Overview of OS model and Programming Tools

Submitted by: Gianna Foti

Date of Experiment:______8 January 2025_______________

Report Deadline:_______15 January 2025________________

Lab Section: ______CPE345-03______

Demonstration Deadline: _____15 January 2025___________

Introduction

In this lab, we focused on understanding Linux processes and programming tools by running and analyzing several demo programs. The primary goal was to become familiar with process creation and management in Linux using system calls such as fork(). We began by compiling and executing various C++ programs that demonstrated how processes are created and how parent and child processes interact. Through these demonstrations, we observed the behavior of processes, analyzed output patterns, and explored concepts like orphan processes and process forking. Additionally, we were required to answer questions related to these demos to reinforce our understanding of process behavior. The lab concluded with a coding assignment where we wrote a program that forked 10 child processes, each printing its process ID and serial number.

Theory Topics

1. Theory was not asked for in the lab assignment.

Part 1:

Program Description: In this section, we were to run the given code and note the behavior.

Assignment 1 Questions (Demo 2):

1. Analyze the output for the demo code above in terms of order in which output statements are printed. You may want to run the program a few times. What do you observe?
    a. Before fork() call: the variable x is initialized to 0
    b. Fork() execution: The fork() system call creates a duplicate child process. Both the parent and child processes continue executing the code following the fork().
    c. Increment of x: Both the parent and the child process independently increment `x` by 1. Since the processes have separate memory spaces, each has its own copy of `x`.

    d. Output Statement: The cout statement prints the process ID (getpid()) and the value of x for both processes.
    e. Order of Output: The output will show two lines: one from the parent and one from the child.
        i. The exact order of these lines is non-deterministic because the operating system's process scheduler decides which process (parent or child) executes first.
        ii. Sometimes the parent's output will appear first, and other times the child's output will.
    f. I observed that the output is printed twice, once by the parent process and once by the child process. I also noted that the order of execution varied with each run. This showed how processes run concurrently and are scheduled by the operating system.

Part 2:

Program Description:
In this section, we were to run the given code.

Assignment 2 Questions (Demo 3):

1. How many processes are created from the above demo code? Explain your answer.
    a. The program results in 4 processes in total.
        i. The original parent and 3 child processes which are created by the two fork() calls.
    b. Each fork() call doubles the number of processes because every existing process executes the fork().
        i. Total Number of Processes = 2^n
            1. N = number of fork() calls
            2. 2^n accounts for how each fork() doubles the number of processes

Part 3:

Program Description: In this section, we were to run the given code.

Assignment 3 Questions (Demo 4):

1. What is an orphan process? Are there any orphan processes in the above code? What is the pid of the orphan process if any are present?
    a. An orphan process is a process whose parent has terminated before the child process finishes execution.
    b. After the child process becomes an orphan, the process dispatcher adopts the child, and the child's parent process ID (PPID) becomes 1.
    c. In the above code, there is an orphan process. When the parent process terminates after sleeping for 5 seconds, the child process continues running for an additional 5 seconds. During this time, the child's PPID changes to 1, but the child's own PID remains the same.

Part 4:

Program Description: This program forks off 10 children, each of which will print out its pid and its serial number.

Process:

1. I used Demo 1 as a starting point to understand basic process forking.
2. First, I added a for loop to fork 10 child processes.
3. Second, I used an if statement to check if the process was a child (pid == 0).
4. Third, each child printed its PID and its parent's PID, then exited.
5. Lastly, I added a loop to make the parent wait for all child processes to finish.

Conclusion

In conclusion, this lab provided a comprehensive introduction to Linux process management and system programming. By running and analyzing various demo programs, we gained a deeper understanding of how processes are created, how parent and child processes interact, and how system calls like fork() function. Writing our own program to create multiple child processes further solidified our grasp of process control and the behavior of concurrent processes. Overall, I did not encounter many difficulties with the lab and was able to effectively learn the fundamental concepts of Linux process management and system-level programming.

Appendix

Table 1: Program 1

```
//Author: Gianna Foti
//Teacher: Trevor H McClain
//Course: CPE 435
//Due Date: 15 January 2025
//Description: Expand the program in Demo 1 and fork off 10 children, each of which will print out its
pid and its serial number (1,2,3,4,5...10).

using namespace std;
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>  // for wait()

int main(int argc, char *argv[])
{
   // printing the PID of the parent process
   cout << "The pid of the parent is " << getpid() << endl;

//start at 1 and loops to create 10 child processes
   for (int i = 1; i <= 10; i++) {
      pid_t c_pid = fork();

// doing a lil failsafe option here, totally optional
      if (c_pid < 0) {
         cout << "Fork failed!" << endl;
         exit(1);
      }
      // child process block
      else if (c_pid == 0) {
         //each child prints the serial number, its PID, and its parents PID
         cout << "I am child number " << i
             << ", my PID is " << getpid()
             << ", and my parent's PID is " << getppid() << endl;
         // child exits to prevent more forking
         exit(0);
      }
```

```
    }

    // parental waits for all child 10 processes to finish
    for (int i = 0; i < 10; i++) {
        // this waits for any child to terminate
        wait(NULL);
    }

    //lil ending statement that says that everything has finished
    cout << "All child processes have finished. Parent process exiting." << endl;

    return 0;
}
```