Operating Systems Lab

CPE 435-01

Introduction to Processes in Linux

By: David Thornton

Lab Date: 19 January 2021

Lab Due: 25 January 2021

Demonstration Due: 25 January 2021

Introduction

This lab will introduce the student to processes in Linux.

Theory

Topic 1: Process

- A process is an instance of a program. Each process has its own area of memory, protected against modification by other processes.

Topic 2: States of processes

- The state of a process is defined in part by the current activity of that process. A process may be in one of the following states:
  - New. The process is being created.
  - Running. Instructions are being executed.
  - Waiting. The process is waiting for some event to occur (such as an I/O completion or reception of a signal).
  - Ready. The process is waiting to be assigned to a processor.
  - Terminated. The process has finished execution.

Topic 3: Orphan process

- Occurs when a parent does not invoke wait() and instead terminates.

Topic 4: Zombie process

- Occurs when a process has terminated, but the parent has not yet called wait()

Topic 5: fork()

- The fork system call creates a copy of a process that was executing.

Topic 6: exit()

- A process terminates by calling the exit system call.

Topic 7: wait()

- The execution of the current process is blocked until the child process has finished execution.

Lab Assignment

1. Write a program that does the following in the order given below:
    1. declare and initialize a variable named val (initialize to 0)
    2. call fork system call
    3. in the child process, add 2 to the value of val , and print it on the console screen along with the pid of the child in the same statement.
    4. in the parent process, add 5 to the value of val, and print it on the console screen along with the pid of the parent process in the same statement.
    5. What can you conclude about the values of the variable val?

2. Write a program, which creates a new process child1. The parent process should print its id and wait for the termination of the child1 process. The child1 process should call a function that subtracts two numbers passed as arguments, print the result on the console screen along with the pid of the process that is printing. It should then create a new child child2 process that adds two numbers. The control should then come back to the first child process child1 that now should multiply two numbers and then terminate. The waiting parent process should now resume and terminate the program.

3. Write a program, which creates n child processes and prints their process id. The n must be an even number and passed as argument, otherwise a message indicates that the number is odd and terminates the program. Please make sure that your implementation has one parent process and n-1 child processes.

4. Write a C program that demonstrates the concept of following:
    1. Orphan Process
    2. Zombie Process
    3. Sleeping Beauty Process
    4. Verify that you actually achieved all the three states mentioned above by using the terminal. Put a screenshot of the process table in your report.

Observations

Additional error handling was implemented to prevent mistakes.

Conclusion

This lab was successful in introducing me to the concept of processes
Demo link

Appendix

Appendix 1: lab02_exercise1.cpp

```
// ****************************************
// Program Title: Lab 02 - Exercise 1
// Project File: lab02_exercise1.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 01/25/2021
// ****************************************

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>

using namespace std;

#define CHILDADD 2 // given
#define PARENTADD 5 // given

int main()
{
        int val = 0;
        val = fork();

        if(val == 0)
        {
                val += CHILDADD;
                cout << "**** Child Process ****" << endl;
                cout << "Val = " << val << endl;
                cout << "Child PID = " << getpid() << endl << endl;
        }
        else
        {
                val += PARENTADD;
                cout << "**** Parent Process ****" << endl;
                cout << "Val = " << val << endl;
                cout << "Parent PID = " << getpid() << endl << endl;
        }
}
```

Appendix 2: lab02_exercise2.cpp

```
// ***************************************
// Program Title: Lab 02 - Exercise 2
// Project File: lab02_exercise2.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 01/25/2021
// ***************************************

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

void child1(int num1, int num2); // Function prototype

int main()
{
	srand (time(NULL)); // initialize random seed for numbers
	int num1 = rand() % 100;
	int num2 = rand() % 100;

	int val = 0;
	val = fork(); // Start child 1

	if (val == 0)
	{
		child1(num1, num2);
	}
	else
	{
		cout << "**** Parent Process ****" << endl;
		cout << "Parent PID = " << getpid() << endl << endl;
		wait(0);
		cout << "\n**** Parent Process ****" << endl;
		cout << "Ending program..." << endl;
		cout << "************************" << endl;
		exit(0);
		// return 0;
	}
}
```

```
void child1(int num1, int num2)
{
	int subResult = num1 - num2;
	int addResult = num1 + num2;
	int mulResult = num1 * num2;

	cout << "**** Child 1 Process ****" << endl;
	cout << "Child1 PID = " << getpid() << endl;
	cout << "Result of " << num1 << " - " << num2 << " = " << subResult << endl;

	int child1val = 0;
	child1val = fork(); // start child 2

	if (child1val == 0)
	{
		cout << "\n**** Child 2 Process ****" << endl;
		cout << "Child2 PID = " << getpid() << endl;
		cout << "Result of " << num1 << " + " << num2 << " = " << addResult << endl;
		exit(0);
	}
	else
	{
		wait(0); // wait for child2 to finish
		cout << "\n**** Child 1 Process ****" << endl;
		cout << "Result of " << num1 << " * " << num2 << " = " << mulResult << endl;
		exit(0);
	}
}
```

Appendix 3: lab02_exercise3.cpp

```
// ****************************************
// Program Title: Lab 02 - Exercise 3
// Project File: lab02_exercise3.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 01/25/2021
// ****************************************

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <cstdlib>
#include <cmath>
```

```
using namespace std;

int main (int argc, char* argv[])
{
	if(argv[1] == NULL)
	{
		cout << "*************** ERROR ***************" << endl;
		cout << "Please enter a number as a paramter." << endl;
		cout << "Exiting program..." << endl;
		cout << "************ END OF ERROR ***********" << endl;
		exit(0);
	}

	int num = atoi(argv[1]); // a to i function is a string parser from stdlib.h

	if (num%2 == 0)
	{
		int count = log2(num); // need log2 because there are 2^n processes
		cout << "Executing for loop " << count << " times" << endl;

		for (int i = 0; i < count; i++)
		{
			int val = 0;
			val = fork();
			if (val == 0)
			{
				cout << "Child ID = " << getpid() << endl;
			}

		}
		exit(0);
	}
	else if (num%2 != 0)
	{
		cout << "*************** ERROR ***************" << endl;
		cout << "Input was an odd number" << endl;
		cout << "Please try an even number" << endl;
		cout << "************ END OF ERROR ***********" << endl;
		cout << "Ending Program..." << endl;
		exit(0);
	}
	exit(0);
}
```

Appendix 4: lab02_exercise4.cpp

```
// ***************************************
// Program Title: Lab 02 - Exercise 4
// Project File: lab02_exercise4.cpp
// Name: David Thornton
// Course Section: CPE-435, SP 2021
// Due Date: 01/25/2021
// ***************************************

#include <stdio.h>
#include <iostream>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>

using namespace std;

int main ()
{
        cout << "**** Beginning Orphan Process ****" << endl;
        int val = 0;
        val = fork();

        if (val == 0)
        {
                cout << "Child PID = " << getpid() << endl;
                cout << "Child PPID = " << getppid() << endl;
                wait(0);
        }
        else
        {
                cout << "Parent PID = " << getpid() << endl;
                exit(0);
        }
        wait(0); // wait for orphan process to end

        cout << endl << "**** Beginning Zombie Process ****" << endl;
        int val2 = fork();
        if (val2 == 0)
        {
                cout << "Child PID = " << getpid() << endl;
                exit(0);
        }
        else
        {
                cout << "Parent PID = " << getpid() << endl;
                wait(0);
        }
        wait(0); // wait for zombie process to end

        cout << endl << "**** Beginning Sleeping Beauty Process ****" << endl;
        cout << "Sleeping Beauty PID = " << getpid() << endl;
        wait(0); // wait for sleeping beauty process to end

        cout << "Ending program..." << endl;
}
```