CPE 435: OPERATING SYSTEMS LABORATORY.

Lab02

Processes.

Submitted by: Dan Otieno.

Date of Experiment: 01/20/23.

Report Deadline: 01/27/23.

Demonstration Deadline: 01/27/23.

Introduction

This lab involved exploring the different types of processes in an operating system and to better understand the fork () system call.

Theory

When the event of program execution is happening in an OS, we define that event as a process, in other words, a process defines the event of program execution in memory. Processes cannot be modified by other processes and while in execution, the fork() system call makes a copy of an ongoing process' address space. When fork() is called, a new process is achieved, called the child process. The process that initiates the fork() is called the parent process. The return value of a child process is 0, whereas that of the parent process is the process Id (PID) of the child process.

A process can be explained in five states within an operating system; New, Waiting, Ready, Running, and Terminated. A New Process is the beginning state, this is where a process is created. Following that is the Waiting state, where a process might be created while a separate event is in progress. In this case, the process must wait for that event to be completed before proceeding to the Ready state where it is attached to a processor. In the Running state, the set of instructions defined in the process are executed by the processor, and when that execution is completed, the process enters the Terminated state.

Some common processes that can be identified in a process table are Orphan, Zombie and Sleeping Beauty.

- ## Orphan Process:
    - An Orphan process is a child that stays running after the parent process has terminated. The orphaned child process is automatically adopted by a process dispatcher when this occurs. According to Wikipedia, "…orphaned processes waste server resources and can potentially leave a server starved for resources. However, there are several solutions to the orphan process problem:
        - Termination is the most commonly used technique; in this case the orphan is killed.
        - Reincarnation is a technique in which machines periodically try to locate the parents of any remote computations; at which point orphaned processes are killed.
        - Expiration is a technique where each process is allotted a certain amount of time to finish before being killed. If need be a process may "ask" for more time to finish before the allotted time expires…."
- ## Zombie Process:
    - A zombie is a process where the child has terminated but can still be found in the process table. The process has not been cleaned up, and therefore is retained as an

entry in the table, often denoted by a “Z” when the process table is launched during the execution of a program.

- Sleeping Beauty:
    - This process involves the operating system responding to events.

Other calls related to process execution are wait() and exit(). The wait() call just returns the exit status of a child process and the process ID of the specific child process that invoked wait(). Wait(0) call waits for the death of a child. The exit() call returns exit status of the child when the process is completed. When exit status is 0, that indicates a successful completion, and if not 0, completion has failed.

Results & Observation

Assignment 1:

Description:

Write a program in C/C++ that does the following in the order given below:

a. Declare and initialize a variable named val (and initialize it to 0)

b. Call the fork system call

c. 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 (val and the child’s pid should be printed in the same line)

d. 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 (val and the parent’s pid should be printed in the same line)

e. What can you conclude about the values of the variable val?

Program Output:

Assignment 2:

Description:

a. Create a new process child1

b. The parent (of child1) should print its id and wait for the termination of the child1 process.

c. The child1 process should call a function that subtracts two numbers passed as arguments and print the result on the console screen along with the pid of the process that is printing.

d. The child1 process should create a new child process (child2)

e. The child2 process should call a function that adds two numbers passed as arguments and print the result along with its pid while child1 waits.

f. Control should then come back to the first child process child1 that now should call a function that multiplies two numbers passed as arguments and print the result and pid of the printing process to the console and then terminate.

g. The waiting parent process should now resume and terminate the program.

h. Note: arguments to functions may be hard coded or taken in as user input.

Program Output:

Assignment 3:

Description:

Write a C/C++ program which creates n child processes and prints their process id. The n number of processes must be an even number and passed as input to the program by the user. If n is not even a message should indicate that the number is odd and terminates the program. Please make sure that your implementation has one parent process which creates n child processes.

Program Output:

```
-bash-4.2$ ./L2_ex3
Please enter an even number: 4

|There are 4 total number of processes|
-----------------------------------------
Parent: Pid = 594|      Child: Pid = 595|
Parent: Pid = 594|      Child: Pid = 596|
Parent: Pid = 594|      Child: Pid = 597|
Parent: Pid = 594|      Child: Pid = 598|
-bash-4.2$ ./L2_ex3
Please enter an even number: 6

|There are 6 total number of processes|
-----------------------------------------
Parent: Pid = 600|      Child: Pid = 603|
Parent: Pid = 600|      Child: Pid = 604|
Parent: Pid = 600|      Child: Pid = 605|
Parent: Pid = 600|      Child: Pid = 606|
Parent: Pid = 600|      Child: Pid = 607|
Parent: Pid = 600|      Child: Pid = 608|
-bash-4.2$ ./L2_ex3
Please enter an even number: 8

|There are 8 total number of processes|
-----------------------------------------
Parent: Pid = 609|      Child: Pid = 610|
Parent: Pid = 609|      Child: Pid = 611|
Parent: Pid = 609|      Child: Pid = 612|
Parent: Pid = 609|      Child: Pid = 614|
Parent: Pid = 609|      Child: Pid = 616|
Parent: Pid = 609|      Child: Pid = 617|
Parent: Pid = 609|      Child: Pid = 618|
Parent: Pid = 609|      Child: Pid = 619|
-bash-4.2$ ./L2_ex3
Please enter an even number: 7
Error!
Odd number entered!
Terminating Program....
-bash-4.2$ ./L2_ex3
Please enter an even number: 19
Error!
Odd number entered!
Terminating Program....
-bash-4.2$
```

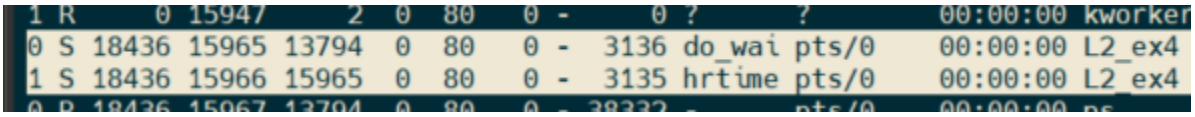
Assignment 4:

Description:

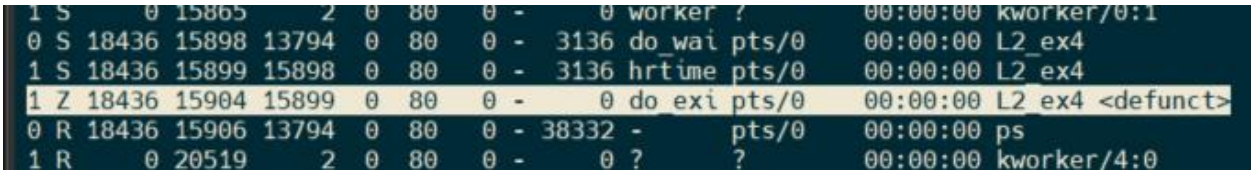
Write a C/C++ program (or programs) that demonstrates the following concepts:
a. Orphan process
b. Zombie process
c. Sleeping beauty process
d. Verify that you actually achieved all the three states mentioned above by using the terminal. Put a screenshot of the process table in your report. Each program should print out their pid, which should be visible in the process table.

Program Output:

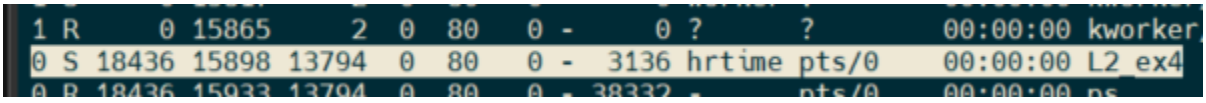
Orphan Process:



Zombie Process:



Sleeping Beauty Process:



Appendix

Assignment 1 code.

```
/*
DAN OTIENO
CPE 434-01
LAB 2
Exercise 1
*/
using namespace std;
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

/*******Function for fork********/
void forkprogram()
{
        int val = 0;
        pid_t pid = fork();

        if(pid == 0)
        {
                val += 2;
                cout << "**********I am the child***********" << endl <<
endl;
                cout << "My Pid is " << getpid() << " and Val = " << val <<
endl;
        cout << endl;
                cout << "************************************" << endl <<
endl;
        }
        else
        {
                val += 5;
                cout << "**********I am the parent***********" << endl <<
endl;
                cout << "My Pid is " << getpid() << " and Val = " << val <<
endl;
        cout << endl;
                cout << "************************************" << endl <<
endl;
        }

}
/***********************************************/
/*********************MAIN**********************/
int main()
{
        forkprogram();

        return 0;
}
```

Assignment 2 code.

```
/*
DAN OTIENO
CPE 434-01
LAB 2
Exercise 2
*/
using namespace std;
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>


/**********************Function for child processes**********************/
void childprocess(int num1, int num2)
{
        int add = num1 + num2;
        int sub = num1 - num2;
        int mul = num1 * num2;

        /**********************New Process for Child 1**************************/
        cout << "------------------I am child 1--------------------" << endl;
        cout << "My Pid is: " << getpid() << endl;
        cout << "Subtraction Result = " << sub << endl;
        cout << "**************************************************" << endl << endl;
        /*****************************************************************/

        /*************************** Child 2 *********************************/
        pid_t pid = fork();

        if(pid == 0)
        {
                cout << "------------------I am child 2--------------------" << endl;
                cout << "My Pid is: " << getpid() << endl;
                cout << "Addition Result = " << add << endl;
                cout << "**************************************************" << endl <<
endl;
                exit(0);
        }
        else
        {
                wait(0);
                cout << "----------------I am child 1 again----------------" << endl;
                cout << "My Pid is: " << getpid() << endl;
                cout << "Multiplication Result = " << mul << endl;
```

```
                cout << "************************************************" << endl <<
endl;
                exit(0);
        }
        /*************************************************************************/
}
/*************************************************************************/
/*****************************MAIN*************************************/

int main()
{

        int firstnum, secondnum;


        cout << "Enter the first number: ";
        cin >> firstnum;
        cout << endl;

        cout << "Enter the second number: ";
        cin >> secondnum;
        cout << endl;

        pid_t mainpid = fork();

        if(mainpid == 0)
        {
                childprocess(firstnum, secondnum);
        }

        else
        {
                cout << "----------------I am the parent----------------" << endl;
                cout << "My Pid is: " << getpid() << endl << endl;
                wait(0);
                cout << "----------------I am the parent----------------" << endl;
                cout << "Terminating Program....." << endl;
                cout << "************************************************" << endl <<
endl;
                exit(0);
        }
}
/*************************************************************************/
```

Assignment 3 code.

```
/*
DAN OTIENO
CPE 434-01
LAB 2
Exercise 3
*/
using namespace std;
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

void chTest(int);

/**********************Function to generate number of processes**********************/
void nForks()
{
        pid_t pid = fork();

        if(pid == 0)
        {
                cout << "Parent: Pid = " << getppid() << "|" << "\tChild: Pid = " << getpid() << "|" <<
endl;
   exit(0);
        }
}

/*****************************************************************************/
/*********************************MAIN*****************************************/

int main()
{

        int n;

        cout << "Please enter an even number: ";
        cin >> n;


 if(!(n%2 == 0))
 {
  cout << "Error!" << endl;
  cout << "Odd number entered!" << endl;
                cout << "Terminating Program...." << endl;
                exit(0);
```

```
 }

 else if(n%2 == 0)
 {
  cout << endl << "|There are " << n << " total number of processes|" << endl;
  cout << "----------------------------------------" << endl;

  for(int i = 0; i<n; i++)
          {
    nForks();
          }
 }

 exit(0);
}
```

Assignment 4 code.

```
/*
DAN OTIENO
CPE 434-01
LAB 2
Exercise 4
*/
using namespace std;
#include <iostream>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>


/**************ORPHAN FUNCTION*************/
void orphan()
{
        pid_t o_pid = fork();

        if(o_pid > 0)
        {
                cout << "Orphan parent PID: " << getpid() << endl; //Orphan's parent.
        }
        else if(o_pid == 0)
        {
                sleep(25);
                cout << "Orphan child PID: " << getpid() << endl; //Orphan child.
        }
}
```

```
/**************ZOMBIE FUNCTION*************/
void zombie()
{
	pid_t z_pid = fork();

	if(z_pid > 0)
	{
		cout << "Zombie PID: " << getpid() << endl; //Zombie's parent.
		sleep(30);
	}
	else
	{
		exit(0); //Zombie child.
	}
}

/**************BEAUTY FUNCTION*************/
void beaut()
{
		sleep(40);
		cout << "Sleeping Beauty PID: " << getpid() << endl;
}

/**************MAIN FUNCTION*************/
main( )
{
	pid_t or_pid = fork();

	if(or_pid == 0)
	{
		orphan();  //Call Orphan function in Main.
		exit(0);
	}

	wait(0);   //Wait for previous process to complete.

	zombie();  //Call Zombie function in Main.

	wait(0);   //Wait for previous process to complete.

	beaut();   //Call Sleeping Beauty function in Main.

	exit(0);
}
/*****************************************/
```

Cited Sources:

https://en.wikipedia.org/wiki/Orphan_process