Assignment 4 : Operation on Process

August 26, 2020

Objective :

- This assignment is intended to learn how to create, work with and manipulate processes in Linux. You are expected to refer to the text book and references mentioned in the course website before you start the lab.

Instructions

- Refer to the sample codes provided in the *Tutorial/Assignment* segment of the course website (codes related to *fork*, *wait*, *exec* and *exit* system calls). You are expected to run all the sample codes provided before you begin working on the lab assignments.

Assignments:

1. Write a *CPU bound* C program and a *I/O bound* C program (e.g. use a number of *printf* statements within a *while(1)* loop). Compile and execute both of them.
   Observe the effect of their CPU share using the *top* display and comment.
2. Write a program in C that creates a child process, waits for the termination of the child and lists its *PID*
3. Compile and run the program *code_for_asgn_4.c* and record your observations. Perform the modification mentioned and answer the questions that follow.
   (a) Comment the inner loop in both the *if* and the *else* blocks, compile and run program *code_for_asgn_4.c* again. Record your observations.
   (b) Do you find any difference in the output. If not, then what do you think is the role of the inner loop in both *if* and the *else* blocks ?
   (c) Modify *code_for_asgn_4.c* in order to make the child process finish before the parent process starts
4. Write a C program to print the address of a variable and enter a long loop (*say using while(1)*).
   (a) Start three to four processes of the same program and observe the printed address values.

(b) Show how two processes which are members of the relationship parent-child are concurrent from execution point of view, initially the child is copy of the parent, but every process has its own data.

5. Create a file named *my_file.txt* that contains the following four lines :
Child 1 reads this line
Child 2 reads this line
Child 3 reads this line
Child 4 reads this line

   Write a C program that forks four other processes. After forking the parent process goes into wait state and waits for the children to finish their execution. Each child process reads a line from the file *my_file.txt* (Child 1 reads line 1, child 2 reads line 2, child 3 reads line 3 and child 4 reads line 4) and each prints the respective line. The lines can be printed in any order.

6. Write two programs file1.c and file2.c
Program file1.c uses these :

   (a) **fork()** to launch another process
   (b) **exec()** to replace the program driving this process, while supplying arguments to file2.c to complete its execution
   (c) **wait()** to complete the execution of the child process
   (d) file1.c takes two arguments x(a number less than 1) and n (number of terms to be added, 1 or more). For example: file1 0.5 5
   (e) When the child proces finishes, the parent prints:
   Parent(PID=yyy) : Done

   Program file2.c requires two arguments to obtain the approximate value of e^x by adding the first n terms in the relation : e^x = 1+x+x^2/2!+x^3/3!+....... and prints the result in the format:

   Child(PID=yyy) : For x = 0.5 the first 5 terms yields 1.6484375

   Hint : Child-specific processing immediately following the fork() command should load file2.c into the newly created process using the exec() command. This exec() command should also pass 2 arguments to the child. Refer to the man page of exec() command to know how to pass on arguments to the child process. Parent-specific processing should ensure that the parent will wait() for the child- specific processing to complete.