Assignment 3 : Operation on Process

August 31, 2018

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 befor you start the lab. Some sample codes for process creation using *fork* system call have been provided for your reference.

Instructions

- You are expected to run all the sample codes provided in the *Helpful Resources* section for Assignment 3. It will help you understand how to work with *fork,exit,wait and exec* system calls. At the same time it will help you in completing the homework assignments.

Class Assignments:

1. Use the *ps, ps lx, ps tree and ps -aux* command to display the process attributes.
2. Learn the *top* command to display the resource utilization statistics of processes
   - Open a terminal and type the *top* command
   - Start a browser and see the effect on the *top* display
   - Compile a C program and observe the same effect (Use a long loop - say while(1) to observe the effect)
   - From the *top* display, answer the following:
     - How much memory is free in the system?
     - Which process is taking more CPU?
     - Which process has got maximum memory share?
   - Write a CPU bound C program and a I/O bound C program (e.g. using more printf statements within while(1) loop), compile and execute both of them.
     Observe the effect of their CPU share using the *top* display and comment.

3. Write a program in C that creates a child process, waits for the termination of the child and lists its PID, together with the state in which the process was terminated (in decimal and hexadecimal)

4. Test the codes for creation of orphan process and zombie process given in the reading resource section of Assignment 3 in the course website.

5. In a C program, print the address of the variable and enter into a long loop (say using while(1)).

   - Start three to four processes of the same program and observe the printed address values.
   - 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.

6. Test the source code below:

   $for(i = 1; i \leq 10; i++)\{$
   fork();
   printf("The process with the PID=%d",getpid());
   }

   In the next phase, modify the code, such as after all created processes have finished execution, in a file *process_management.txt* the total number of created processes should be stored.

Homework Assignments:

1. 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.

2. Write two programs : one called *client.c*, the other called *server.c*. The client program lists a prompter and reads from the keyboard two integers and one of the characters '+' or '-'. The read information is transmitted with the help of the system call *execl* to a child process, which executes the server code. After the child (server) process fiishes the operation, it transmits the result to parent process (client) with the help of the system call *exit*. The client process prints the result on the screen and also reprints the prompter, ready for a new reading.