Systems Software/Programming – Lab Manual

Lab 6 – Processes' Systems Calls fork(), wait(), exec(), exit()

Write C program for each of the problem below. After executing each of the C program, you will need to capture the output in text file format.
C program file will be named as StudentID_Lab6_x.c
Text file with output captured will be names as StudentID_Lab6_x.txt

Problem 1: Using a child process calculate e^x which is calculated with formula

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

- Create parent process as StudentID_Lab6_1_expo_parent.c
    - Use fork() to create a child process
    - exec() command to replace the program driving this process, while also supplying the arguments that this new program (as StudentID_Lab6_1_expo_child.c) needs to complete its execution
    - wait() command to wait for the completion of the execution of the child process. Parent process captures which child has exited and the exit value returned.
    - The program StudentID_Lab6_1_expo_parent.c will take two command line arguments and pass them on to the StudentID_Lab6_1_expo_child.c. The command line arguments for StudentID_Lab6_1_expo_parent.c are x (a number less than 1) and n (number of terms to be added, 1 or more). For example

      $ StudentID_Lab6_1_expo_parent 0.5 5
    - When the child process is finished and returns back using exit() systems call, the parent process:
        - Capture the reason why child has exited using macros such as WIFEXITED, WIFSIGNALED and also captures the exit value using macro WEXITSTATUS. It then prints the appropriate message for normal or abnormal termination of child process. If child process was terminated normally then it will also print "Child

        with PID xxx has exited with value of e=xxx" (Note: exit() function sends only integer value back to the parent)
    - Finally parent process prints "Parent (PID = xxx): done" → Note that this PID is for parent process

- Create parent process as StudentID_Lab6_1_expo_child.c
  - This program requires two arguments to obtain an approximation of x by adding the first n terms, and prints the result using the format
  - When child process is complete before the exit(), it prints "Child (PID =xxx): For x =0.5 the first 5 terms yield 1.6484375" → Note that this PID is for child process
  - Pass the value of e as calculated in child process as exit status by calling exit(e_var).

Problem 2: Using Problem 1 we have learned that parent process can get work done by child process e.g. we calculated a value of e^x for a given values of x and n. Now consider that you have been given multiple set of values for x and n so by creating multiple child processes and passing different values of x and n we can get work done fasters. That's what we need to implement in problem 2.

- Take the code of parent process as StudentID_Lab6_1_expo_parent.c and create a new file StudentID_Lab6_2_expo_parent.c
  - Accept the multiple values of x and n

    ```
    $ StudentID_Lab6_2_expo_parent 0.5 5 0.2 3 0.7 6
    ```
  - For each set of x and n values you need create a new child process. In this example you need to create 3 child processes. You can use argc value to find out how many child must be create if argc=7 then you need 3 child process
    - Child 1 for x=0.5 and n=5
    - Child 2 for x=0.2 and n=3
    - Child 2 for x=0.7 and n=6
- There will not be any change required in the child process

Submission:
StudentID_Lab6.zip with total 4 files (2 C program files + 2 captured output text files)

Optional Assignment:

Write a program print the process chain for a given process ID i.e. start with the current process, get its parent process, get the parent's parent and continue until you reach the init process and print the list of all pid and process names. You can use recursive function too.

```
faculty@faculty-OptiPlex-3040:/proc/29593$ ps axjf
PPID   PID   PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
  0      1     1      1  ?        -1    Ss      0    0:04  /sbin/init splash
……
  1    919   919    919  ?       -1     SLsl    0    0:00   /usr/sbin/lightdm
 919   936   936    936  tty7     936   Ssl+    0    47:22  \_ /usr/lib/xorg/Xorg -core :0 -seat seat0 -auth /
 919   1129  919    919  ?       -1     Sl      0    0:00   \_ lightdm --session-child 12 19
1129   1429  1429  1429  ?       -1     Ss     1000  0:00       \_ /sbin/upstart --user
1429   1518  1517  1517  ?       -1     S      1000  0:00            \_ upstart-udev-bridge --daemon --user
```

In the above example we see that process id 1518 (upstart-udev-bridge --daemon --user) is the leaf, its parent is 1429. 1429's (/sbin/upstart --user) parent is 1129, 1129's (lightdm --session-child 12 19) parent is 919, 919's (/usr/sbin/lightdm) parent is 1 (/sbin/init splash) and final parent for pid 1 is 0.