The Department of Electrical and Computer Engineering

The University of Alabama in Huntsville

CPE 435 Lab-02

Processes

Introduction

Processes

A process is an instance of a program. The new process is created by issuing a fork system call. Each process has its own area of memory, protected against modification by other processes. The **fork** system call creates a copy of a process that was executing. The process that executed the fork is called the **parent** process and the new process is called the **child** process. The only difference in the returns from the fork system call is that the return value in the parent process is the process id of the newly created child process, while the return value in the child process is zero. If the fork is not successful, -1 is returned.

A process terminates by calling the **exit** system call. This system call never returns to the caller. When exit is called, an integer exit status is passed by the process to the kernel.

Exercises

1. 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. <u>What can you conclude about the values of the variable *val*?</u>

2. Write a C/C++ programming that does the following:
   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

3. 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.

4. 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.

Topics for Theory

Define and discuss in your own words the following: process, the five states of processes, orphan process, and zombie process. Additionally, write a paragraph on each of the functions mentioned in the hint (fork(), exit(), and wait()).

Hint

int fork() : function either returns 0 for child process or the process id if it is parent. **exit(status)**: when exit is called, an integer exit status is passed by the process to the kernel. (used usually by the child process to terminate and return it's status to the parent process). **int wait(int *status)**: a process can wait for one of its child processes to finish by executing the wait system call. The value returned by wait is the process id of the child process that terminated . If there are more than one child processes, then the wait function returns after any one child is terminated and the value returned is the pid of the child.