CS-362L Operating System Lab 02

Assignment 2

Linux system calls related to process and process control.

Objectives: To get familiar with different system calls used for process control.

Processing steps:

Step 1: Study fork() system call.
Fork system call is used for creating a new process, which is called **child process**, which runs concurrently with the process that makes the fork() call (parent process). After a new child process is created, both processes will execute the next instruction following the fork() system call. A child process uses the same pc(program counter), same CPU registers, same open files which use in the parent process.

It takes no parameters and returns an integer value. Below are different values returned by fork().

Negative Value: creation of a child process was unsuccessful. **Zero:** Returned to the newly created child process. **Positive value:** Returned to parent or caller. The value contains process ID of newly created child process.
Total number of process created = **2n**, where n is number of fork system calls.
Example:

```c
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main()
{

    // make two process which run same
    // program after this instruction
    fork();

    printf("Hello world!\n");
    return 0;
}
```

Output:
Hello world!
Hello world!

Step 2: Study exec() system call.
The exec() family of functions replaces the current process image with a new process image. It is used to create new process. The process have the same id as the previous process.

Step 3: Study wait() system call.
The wait() system call is used by a parent process to wait for the status of the

child to change. A status change can occur for a number of reasons, the program stopped or continued, but we'll only concern ourselves with the most common status change: the program terminated or exited. Once the parent calls wait(), it will block until a child changes state.
If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.
If only one child process is terminated, then return a wait() returns process ID of the terminated child process.
If more than one child processes are terminated than wait() reap any arbitrarily child and return a process ID of that child process.
When wait() returns they also define exit status (which tells our, a process why terminated) via pointer, If status are not NULL.
If any process has no child process then wait() returns immediately "-1".

Step 4: Study getpid() system call.
When any process is created, it has a unique id which is called its process id. This function returns the process id of the calling function.

Step 4: Study getppid() system call.
Getppid() returns the process ID of the parent of the current process.

Step 5: Running C/C++ program in Linux

- Open terminal. Type following command to install gcc or g++ complier: To check gcc version type this command:
  gcc -version
  sudo apt install gcc

- Now go to that folder where you will create C/C++ programs. For example, in Documents directory. Type these commands:
  cd Documents/
  Now create the file

- Open a file using any editor. Type the following commands in terminal.
  touch program1.c
  Where "program1" is the name of file. You can create a file with any name you want.

- Add code in the file (C code in C file and C++ code in C++ file.) Save the file and exit the editor.
  cat > program1.c

- Now write your program. After completion of program, press **ctrl+shift+D**

gcc program1.c -o test1
Where test is the executable or objectfile of program1.c program.

Note: Make sure you are in the same directory where you have created your program before compiling it.

- To run this program type this command:
  For running C program: **./test1**

Conclusion: At the end of this lab, student will be able to create process, control process and terminate process using system calls.

Exercise

1. Print Process ID, Parent process ID, for current process.
2. Write a program to create a child process using fork system call. Your program executes child process before parent process. Hint: use wait system call.
3. Write a C program using the fork () system call that generates the Fibonacci sequence in the child process.

References:

- ”Fork System Call”
  https://www.geeksforgeeks.org/fork-system-call/
- ”C compiler on ubuntu”
  https://linuxconfig.org/how-to-install-g-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux