

Refresher Lab for Process Management

Part- 1: Creating child processes and spawning new programs

In this section you will learn the following

1. Identifying process information using linux commands
2. Identifying process id using C library function getpid
3. Using fork() function call to spawn a new process
4. Using sleep() function
5. Using exec() function to load new programs to memory

Finding the process-id and status of current running programs on Linux:

Use the command *ps* for finding the processes running on the system. Running *ps* with the options *ax* will give the following output.

```
rick@ubuntu:~$ ps ax
  PID TTY          STAT       TIME COMMAND
    1 ?           Ss        0:02 /sbin/init auto noprompt
    2 ?           S          0:00 [kthreadd]
    4 ?           I<         0:00 [kworker/0:0H]
    6 ?           I<         0:00 [mm_percpu_wq]
    7 ?           S          0:00 [ksoftirqd/0]
    8 ?           I          0:00 [rcu_sched]
    9 ?           I          0:00 [rcu_bh]
   10 ?           S          0:00 [migration/0]
   11 ?           S          0:00 [watchdog/0]
   12 ?           S          0:00 [cpuhp/0]
```

Create a simple C program as follows and store it in test.c:

```
#include <stdio.h>

int main()
{
    int n;

    printf("Enter a number: ");
    scanf("%d",&n);
    printf("The entered number is %d\n",n);
}
```

Compile the program using gcc and run the binary file. While the program is running, open another terminal and find the process id of the program using *ps ax*

Finding the process-id from within the program using the function getpid():

Q1. The process id of a program can also be found out by using *getpid ()* function. Find out how to use the function from Linux man(manual) pages and use it in the previous program to find the process id of the program.

Creating a new process using fork:

A new process is created with the fork system call. When fork is called, the operating system creates a new process: it assigns a new process entry in the process table and clones the information from the current one. All file descriptors that are open in the parent will be open in the child. The executable memory image is copied as well. As soon as the fork call returns, both the parent and child are now running at the same point in the program. The only difference is that the child gets a different return value from fork. The parent gets the process ID of the child that was just created.

SIT Internal

The child gets a return of 0. If fork returns -1 then the operating system was unable to create the process.

The following program shows how a child program is created:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int n = 0;
int main()
{
    int pid;

    pid = fork();

    if(pid == 0){
        printf("This is a child process with pid= %d\n",getpid());
    }
    else{
        printf("This is a parent process with pid= %d\n",getpid());
        printf("This is a parent process with fork function returning a value %d\n",pid);
    }
}
```

Use *ps ax* to find the two process id's of both child and parent processes. To make the process wait for some time before finishing use *sleep()* function.

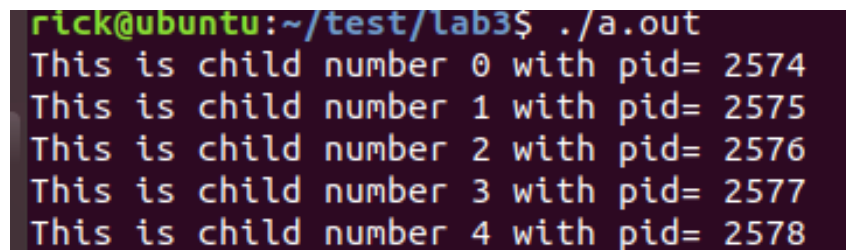
```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int n = 0;
int main()
{
    int pid;

    pid = fork();

    if(pid == 0){
        printf("This is a child process with pid= %d\n",getpid());
        sleep(5);
    }
    else{
        printf("This is a parent process with pid= %d\n",getpid());
        printf("This is a parent process with fork function returning a value %d\n",pid);
        sleep(5);
    }
}
```

Q2: Write a program, which create 5 child processes using fork. Each of the child process should print the child number and its process id. The output of the program should be as follows. Each line should be printed by each child program.



```
rick@ubuntu:~/test/lab3$ ./a.out
This is child number 0 with pid= 2574
This is child number 1 with pid= 2575
This is child number 2 with pid= 2576
This is child number 3 with pid= 2577
This is child number 4 with pid= 2578
```

Spawning different program from a parent program:

Fork function create a copy of the program as a child and both parent and child process runs at the same time. However, when the operating system boots up, different child process has to be spawned. To run different programs we can combine two function calls – fork and exec.

Fork function will create a copy of the current process as a child process. Exec function on the other hand replaces current process binary with a new process binary. The exec() call replaces the entire current contents of the process with a new program. It loads the program into the current process space and runs it from the entry point. There are a

SIT Internal

few variants of exec function. In this lab we will just concentrate on `execl()` where the arguments to the program are passed a list of strings.

Compile the first program *test.c* using `gcc` and store the binary in a file *test* using the following command

```
gcc test.c -o test
```

Write the following program and store it in a file named *spawn.c*

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>

int n = 0;
int main()
{
    int pid;

    execl("./test", "./test", NULL);

    printf("I am printing from the program spawn.c\n");
}
```

Compile the program and run it, to see how the program execute. Show the output to one of the lab coordinators.

Q3: Write the following three different programs:

1. To print *Hello*
2. To print *World*
3. To print *SIT is the best*

Write a program that create three child programs using `fork()`. Each child program use `execl()` to call one of the above programs and executes it.