

# ST. XAVIER'S COLLEGE

(Affiliated to Tribhuvan University)  
Maitighar, Kathmandu



## **OS Lab Assignment #3**

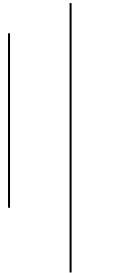
Process Creation

### **SUBMITTED BY:**

Milan Rawal

018BSCIT019

2<sup>nd</sup> Year/4<sup>th</sup> Sem



### **SUBMITTED TO:**

<b>Er. Rajan Karmacharya (Coordinator)</b>	
<b>Er Rabin Maharjan (Lecturer)</b>	

**Department of Computer Science**

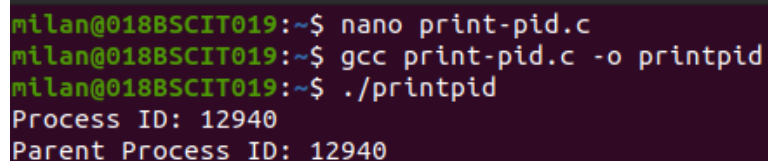
## TITLE: Implementing Process Creation in Linux

### 1. Process IDs

Each process in a Linux system is identified by its unique process ID, referred to as pid. Process IDs are 1- 32768 numbers that are assigned sequentially by Linux as new processes are created. Every process also has a parent process (except init). Thus, we can think the processes in Linux are arranged in a tree, with the init process at its root. The parent process ID, or ppid, is simply the process ID of the process's parent. Most of the process manipulation functions are declared in the header file <unistd.h>. A program can obtain the process ID of the process it's running with the getpid() system call, and it can obtain the process ID of its parent process with the getppid() system call.

#### Ex. 1.1: Printing the Process ID

```
#include <stdio.h>
#include <unistd.h>
int main() {
    int p_id, p_pid;
    p_id = getpid(); /*process id*/
    p_pid = getppid(); /*parent process id*/
    printf("Process ID: %d\n", p_id);
    printf("Parent Process ID: %d\n", p_pid);
    return 0;
}
```



```
milan@018BSCIT019:~$ nano print-pid.c
milan@018BSCIT019:~$ gcc print-pid.c -o printpid
milan@018BSCIT019:~$ ./printpid
Process ID: 12940
Parent Process ID: 12940
```

### 2. Process Creation using fork

Linux provides one function, fork, that makes the child process an exact copy of its parent process. Linux also provides another set of functions, the exec family, that replaces the current process image with a new process image.

#### Using fork

When programs call fork, the parent process continues executing the program from the point that fork was called. The child process, too, executes the same process from the same place. The fork returns some value for its parent and the child process always has 0 pid.

#### Ex. 2.1: Using fork

```
#include<stdio.h>
#include <sys/types.h>
#include<unistd.h>
int main(){
    printf("This is to demonstrate the fork\n");
    fork();
    printf("Hi!, Everybody\n");
    return 0;
}
```

```
milan@018BSCIT019:~$ nano fork1.c
milan@018BSCIT019:~$ gcc fork1.c -o fork1
milan@018BSCIT019:~$ ./fork1
This is to demonstrate the fork
Hi!, Everybody
Hi!, Everybody
```

### Ex. 2.2: Using fork

```
#include <stdio.h>
#include <unistd.h>
int main(void){
    int pid;
    printf("The main program process ID=%d \n", (int) getpid());
    pid = fork();
    if (pid != 0){
        printf("This is the parent process, with ID % d \n", (int) getpid());
        printf("The child process ID=%d \n", pid);
    }
    else printf(" This is the child process, with ID %d\n", (int) getpid());
}
```

```
milan@018BSCIT019:~$ nano fork2.c
milan@018BSCIT019:~$ gcc fork2.c -o fork2
milan@018BSCIT019:~$ ./fork2
The main program process ID=13356
This is the parent process, with ID 13356
The child process ID=13357
This is the child process, with ID 13357
```