**ST. XAVIER’S COLLEGE**

**(Affiliated to Tribhuvan University)**

**Maitighar, Kathmandu**

****

**OPERATING SYSTEM REPORT #05**

**SUBMITTED BY:**

Pradeep Dahal

017BSCIT029

2nd year/ 4th sem

|  |  |
| --- | --- |
|  | Signature |
| Mr. Rabin Maharjan  (Lecturer) |  |
| Department of Computer Science | |

**SUBMITTED TO:**

# Implementing Process Creation in Linux.

## Looking at Processes

**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 process are created. Every process also has a parent process (except init). Thus, we can think the processes in the 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 pid, ppid;

pid = getpid();

ppid = getppid();

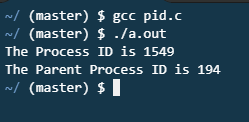
printf(“The Process ID is %d \n”, pid);

printf(“The Parent Process ID is %d \n”, ppid);

return 0;

}

**OUTPUT:**

****

**1.2 Viewing Processes**

ps – display the process running on that system.

pstree – display the all process in process hierarchy.

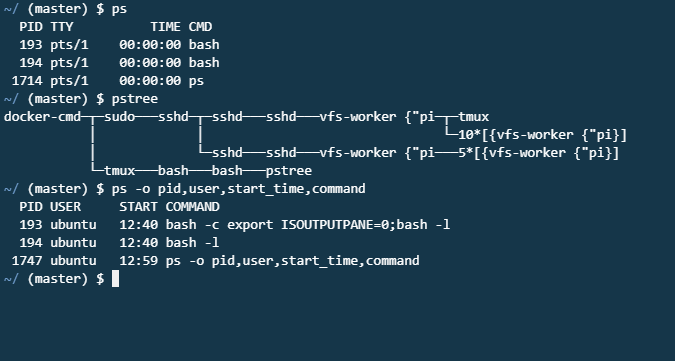
ps [options] – ps has different options.

e.g.: ps -e -o pid,user,start\_time,command

ps -o pid,ppid,user.

ps 12 -Creating a processes

**OUTPUT:**



## 2. Process Creation Using Fork:

Linux provides one function, fork that makes the child process that is 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 calls 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 return some value for its parent and the child process always has 0 pid.

**Ex. 2.1: Using fork**

#include<stdio.h>

int main()

{

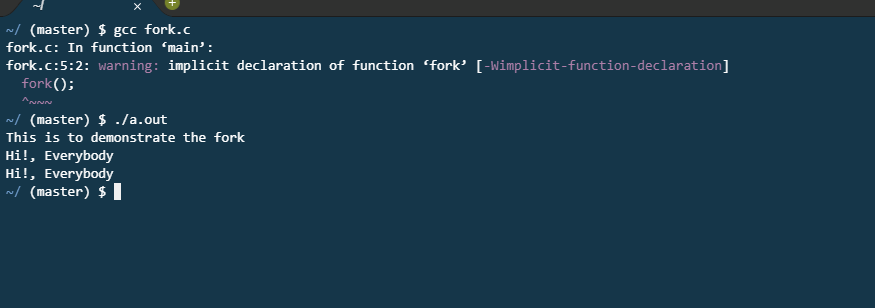
printf(“ This is to demonstrate the fork\n”);

fork();

printf(“Hi!, Everybody\n”);

}

**OUTPUT:**



**Ex. 2.2: Using fork**

#include < stdio.h>

#include <unistd.h>

int main()

{

int pid;

printf(“The main program process ID is %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 is %d \n”, pid);

}

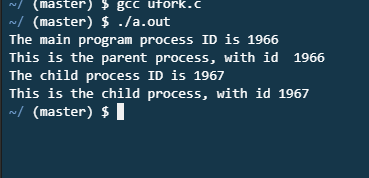
else

printf(“ This is the child process, with id %d\n”, (int) getpid());

return 0;

}

**OUTPUT:**



**Ex. 2.3: A program that creates multiple process using fork () system call.**

#include <unistd.h>

#include <stdio.h>

int main()

{

    // Creating first child

    int n1 = fork();

    // Creating second child. First child

    // also executes this line and creates

    // grandchild.

    int n2 = fork();

    if (n1 > 0 && n2 > 0)

    {

        printf("parent\n");

        printf("%d %d \n", n1, n2);

        printf(" my id is %d \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else if (n1 == 0 && n2 > 0)

    {

        printf("First child\n");

        printf("%d %d \n", n1, n2);

        printf("my id is %d  \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else if (n1 > 0 && n2 == 0)

    {

        printf("second child\n");

        printf("%d %d \n", n1, n2);

        printf("my id is %d  \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    else {

        printf("third child\n");

        printf("%d %d \n", n1, n2);

        printf(" my id is %d \n", getpid());

        printf(" my parentid is %d \n", getppid());

    }

    return 0;

}

**OUTPUT:**

