

Practical : 03

Aim: Implement process creation using the fork() system call in Linux to generate parent and child processes, retrieve their PID and PPID, and analyse system behaviour by creating orphan and zombie processes.

1. Adam is working in an IT company. He has been given a task to reduce the load of a system by killing some of the processes running in the LINUX operating system. Which commands will he use to complete the given task with the help of the following operation?

- Kill processes by name
- Kill a process based on the process name
- Kill a single process at a time with the given process ID

```
M ~

HP@DESKTOP-IUUK17D MSYS ~
$ taskkill //IM notepad.exe
SUCCESS: Sent termination signal to the process "Notepad.exe" with PID 20448.

HP@DESKTOP-IUUK17D MSYS ~
$ |
```

```
M ~

HP@DESKTOP-IUUK17D MSYS ~
$ taskkill //IM chrome.exe //F
SUCCESS: The process "chrome.exe" with PID 12764 has been terminated.
SUCCESS: The process "chrome.exe" with PID 13796 has been terminated.
SUCCESS: The process "chrome.exe" with PID 4664 has been terminated.
SUCCESS: The process "chrome.exe" with PID 3244 has been terminated.
SUCCESS: The process "chrome.exe" with PID 15004 has been terminated.
SUCCESS: The process "chrome.exe" with PID 18048 has been terminated.
SUCCESS: The process "chrome.exe" with PID 7760 has been terminated.
SUCCESS: The process "chrome.exe" with PID 14172 has been terminated.
SUCCESS: The process "chrome.exe" with PID 12592 has been terminated.
SUCCESS: The process "chrome.exe" with PID 6376 has been terminated.
SUCCESS: The process "chrome.exe" with PID 13072 has been terminated.
SUCCESS: The process "chrome.exe" with PID 16388 has been terminated.
SUCCESS: The process "chrome.exe" with PID 15084 has been terminated.
SUCCESS: The process "chrome.exe" with PID 16336 has been terminated.
SUCCESS: The process "chrome.exe" with PID 10696 has been terminated.
SUCCESS: The process "chrome.exe" with PID 12580 has been terminated.
SUCCESS: The process "chrome.exe" with PID 12372 has been terminated.
SUCCESS: The process "chrome.exe" with PID 8824 has been terminated.
SUCCESS: The process "chrome.exe" with PID 18396 has been terminated.
SUCCESS: The process "chrome.exe" with PID 2788 has been terminated.
SUCCESS: The process "chrome.exe" with PID 15928 has been terminated.
```

2. Write a program for process creation using C - Orphan Process

```
M ~
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;

    pid = fork(); // Create child process

    if (pid > 0) {
        // Parent process
        printf("Parent process\n");
        printf("Parent PID = %d\n", getpid());
        sleep(2); // Parent exits early
    }
    else if (pid == 0) {
        // Child process
        sleep(5); // Child runs longer
        printf("Child process\n");
        printf("Child PID = %d\n", getpid());
        printf("Parent PID after orphan = %d\n", getppid());
    }
    else {
        printf("Fork failed\n");
    }

    return 0;
}
```

OUTPUT:

```
M ~
HP@DESKTOP-IUUK17D MSYS ~
# gcc hello.c -o hello

HP@DESKTOP-IUUK17D MSYS ~
# ./hello
Parent process
Parent PID = 627

HP@DESKTOP-IUUK17D MSYS ~
# Child process
Child PID = 628
Parent PID after orphan = 1
```

▪ Zombie Process

```
M ~
GNU nano 8.7
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main() {
    pid_t pid;

    pid = fork(); // Create child process

    if (pid > 0) {
        // Parent process
        printf("Parent process\n");
        printf("Parent PID = %d\n", getpid());
        printf("Child PID = %d\n", pid);

        // Parent sleeps, child exits first + becomes zombie
        sleep(10);
        printf("Parent exiting after child\n");
    }
    else if (pid == 0) {
        // Child process
        printf("Child process\n");
        printf("Child PID = %d\n", getpid());
        printf("Child exiting now...\n");
        exit(0); // Child exits immediately
    }
    else {
        printf("Fork failed\n");
    }

    return 0;
}
```

OUTPUT:

```
M ~
HP@DESKTOP-IUUK17D MSYS ~
# gcc hello.c -o hello
HP@DESKTOP-IUUK17D MSYS ~
# ./hello
Parent process
Child process
Parent PID = 659
Child PID = 660
Child PID = 660
Child exiting now...

Parent exiting after child
HP@DESKTOP-IUUK17D MSYS ~
#
HP@DESKTOP-IUUK17D MSYS ~
# |
```

3.Create the process using fork () system call.

- Child Process creation
- Parent process creation
- PPID and PID

M ~

```
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;

    // Create a child process
    pid = fork();

    if (pid < 0) {
        // Fork failed
        printf("Fork failed!\n");
    }
    else if (pid == 0) {
        // Child process
        printf("Child process\n");
        printf("Child PID = %d\n", getpid());
        printf("Child's Parent PID = %d\n", getppid());
    }
    else {
        // Parent process
        printf("Parent process\n");
        printf("Parent PID = %d\n", getpid());
        printf("Parent's Child PID = %d\n", pid);
    }

    return 0;
}
```

OUTPUT:

```
HP@DESKTOP-IUUK17D MSYS ~
$ nano hello.c

HP@DESKTOP-IUUK17D MSYS ~
$ gcc hello.c -o hello

HP@DESKTOP-IUUK17D MSYS ~
$ ./hello
Parent Process
Child Process
Parent PID = 1233
Child PID = 1234
Child PID = 1234
Parent PID = 1233

HP@DESKTOP-IUUK17D MSYS ~
$ |
```