

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

```
user@AnushkaGudadhe MINGW64 ~/Desktop (main)
$ taskkill //IM notepad.exe
SUCCESS: Sent termination signal to the process "Notepad.exe" with PID 23956.
```

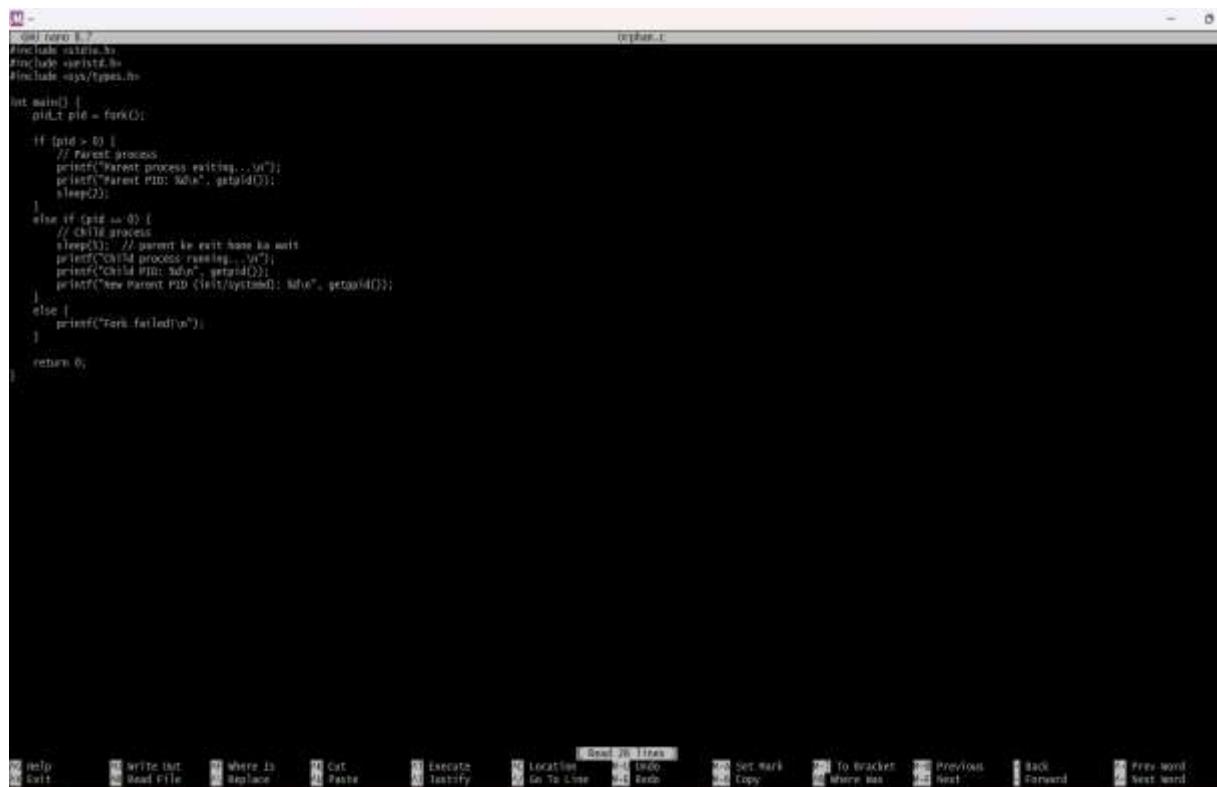
```
user@AnushkaGudadhe MINGW64 ~/Desktop (main)
$ |
```

```
user@AnushkaGudadhe MINGW64 ~/Desktop (main)
$ taskkill //IM notepad.exe
SUCCESS: Sent termination signal to the process "Notepad.exe" with PID 16872.

user@AnushkaGudadhe MINGW64 ~/Desktop (main)
$ taskkill //IM chrome.exe //F
SUCCESS: The process "chrome.exe" with PID 20864 has been terminated.
SUCCESS: The process "chrome.exe" with PID 14628 has been terminated.
SUCCESS: The process "chrome.exe" with PID 18832 has been terminated.
SUCCESS: The process "chrome.exe" with PID 12692 has been terminated.
SUCCESS: The process "chrome.exe" with PID 18976 has been terminated.
SUCCESS: The process "chrome.exe" with PID 2480 has been terminated.
SUCCESS: The process "chrome.exe" with PID 11384 has been terminated.
SUCCESS: The process "chrome.exe" with PID 3840 has been terminated.
SUCCESS: The process "chrome.exe" with PID 13432 has been terminated.
SUCCESS: The process "chrome.exe" with PID 17508 has been terminated.
SUCCESS: The process "chrome.exe" with PID 15504 has been terminated.
```

2. Write a program for process creation using C

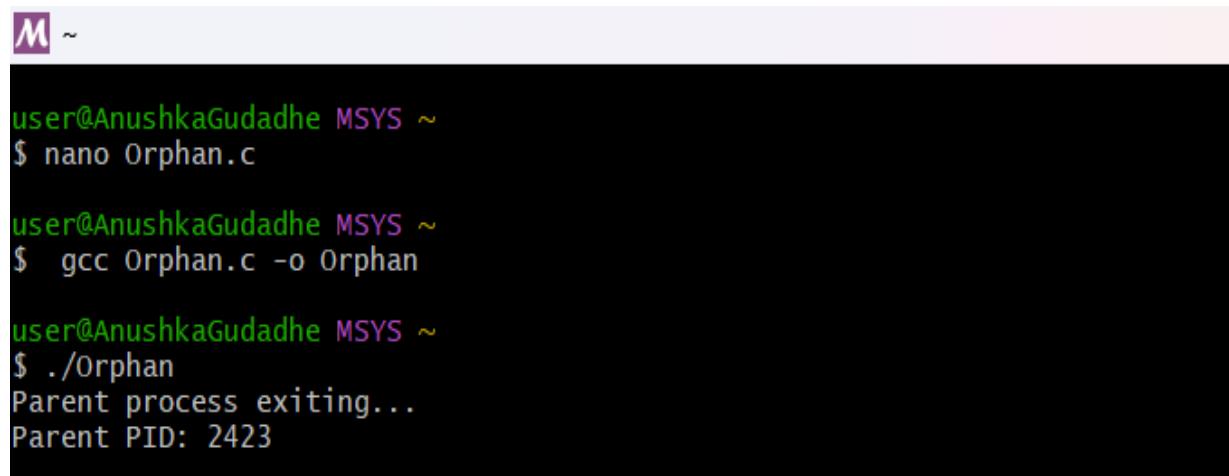
▪ Orphan Process



```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/types.h>
#include <sys/types.h>

int main() {
    pid_t pid = fork();
    if (pid > 0) {
        // Parent process
        printf("Parent process exiting...\n");
        printf("Parent PID: %d\n", getpid());
        sleep(2);
    } else if (pid == 0) {
        // Child process
        sleep(2); // parent ka exit hoga ka wait
        printf("Child process running...\n");
        printf("Child PID: %d\n", getpid());
        printf("New Parent PID (%d/systemid): %d\n", getpid());
    } else {
        printf("fork failed!\n");
    }
    return 0;
}
```

OUTPUT:



```
user@AnushkaGudadhe MSYS ~
$ nano Orphan.c

user@AnushkaGudadhe MSYS ~
$ gcc Orphan.c -o Orphan

user@AnushkaGudadhe MSYS ~
$ ./Orphan
Parent process exiting...
Parent PID: 2423
```

• Zombie Process

The screenshot shows a terminal window with the following content:

```
user@AnushkaGudadhe MSYS ~
$ nano Zombie.c
int main()
{
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child exiting\n");
    } else {
        sleep(10);
        printf("Parent running\n");
    }
    return 0;
}

user@AnushkaGudadhe MSYS ~
$ gcc Zombie.c -o Zombie
user@AnushkaGudadhe MSYS ~
$ ./Zombie
Child exiting
Parent running
```

OUTPUT:

The screenshot shows a terminal window with the following content:

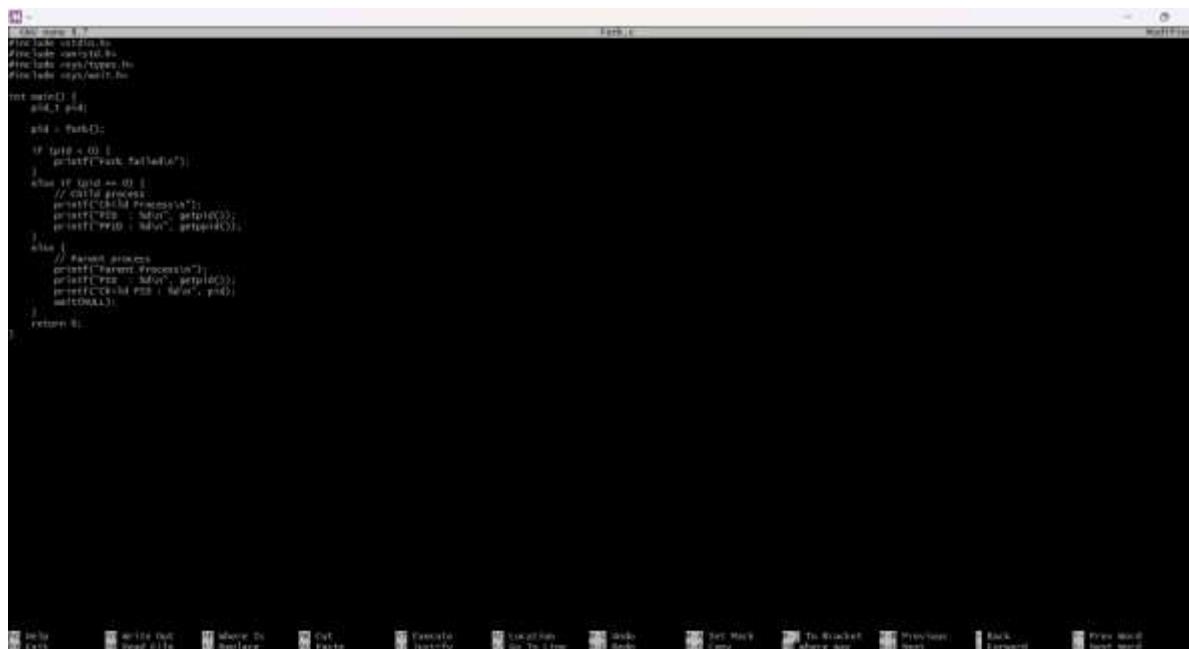
```
M ~

user@AnushkaGudadhe MSYS ~
$ nano Zombie.c
int main()
{
    pid_t pid = fork();
    if (pid == 0) {
        printf("Child exiting\n");
    } else {
        sleep(10);
        printf("Parent running\n");
    }
    return 0;
}

user@AnushkaGudadhe MSYS ~
$ gcc Zombie.c -o Zombie
user@AnushkaGudadhe MSYS ~
$ ./Zombie
Child exiting
Parent running
```

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

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



```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/malloc.h>
#include <sys/sysproto.h>
#include <sys/conf.h>

int main() {
    pid_t pid;
    if ((pid = fork()) < 0) {
        perror("fork failed");
    } else if (pid == 0) {
        // child process
        printf("PPID : %d\n", getppid());
        printf("PID : %d\n", getpid());
    } else {
        // parent process
        printf("Parent Process\n");
        printf("PID : %d\n", getpid());
        printf("Child PID : %d\n", pid);
        waitpid(pid, 0, 0);
    }
    return 0;
}
```

OUTPUT:

```
user@AnushkaGudadhe MSYS ~
$ nano Fork.c

user@AnushkaGudadhe MSYS ~
$ gcc Fork.c -o Fork

user@AnushkaGudadhe MSYS ~
$ ./Fork
Parent Process
PID : 2440
Child PID : 2441
Child Process
PID : 2441
PPID : 2440
```