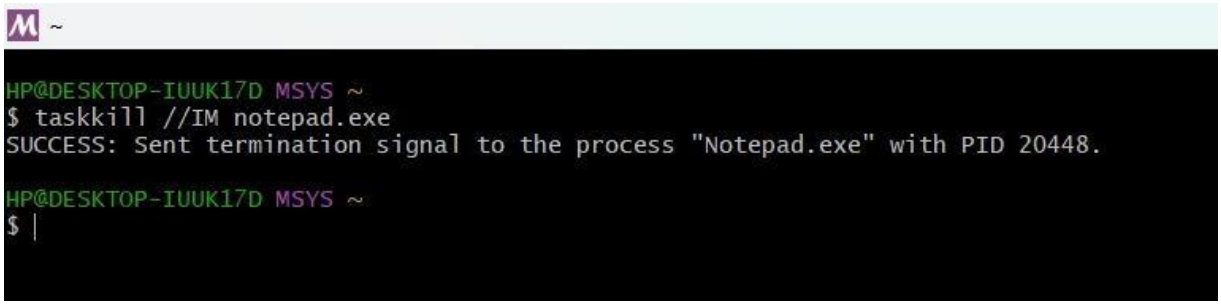
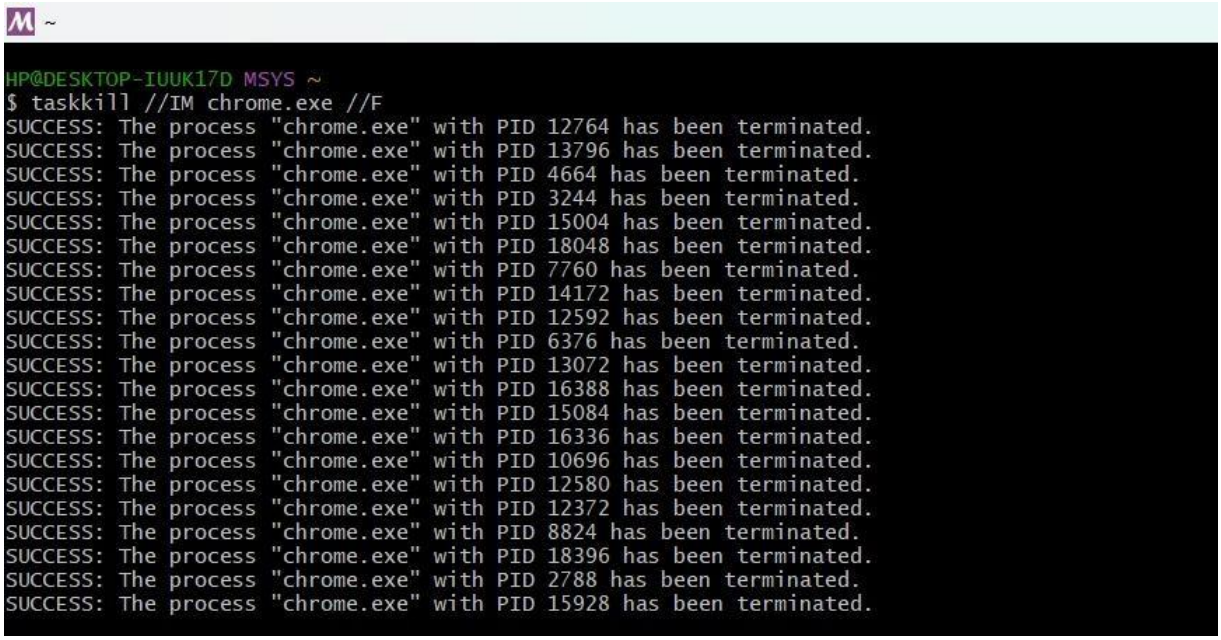
Practical : 03

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





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

```
~
  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:

```
~

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**

```
~
  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:

```
~
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

```
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 ~
$
```