CMR College of Engineering & Technology

(UGC Autonomous)

Kandlakoya, Medchal, Hyderabad - 501401

Department of Computer Science And Engineering

A Dissertation submitted to JNTU Hyderabad in partial fulfillment of the academic requirements for the award of the degree.

Bachelor of Technology

in

Computer Science and Engineering

Submitted by

Meer Sameer (22H51A05H1)

Midde Manu Priya (22H51A05H2)

Perugu Sai Kumar (22H51A05H6)

Under the esteemed guidance of

S. Sangeetha

Asst. Professor

Department of Computer Science and Engineering

CMR COLLEGE OF ENGINEERING & TECHNOLOGY

(UGC Autonomous)

*Approved by AICTE *Affiliated to JNTUH *NAAC Accredited with A+ Grade

KANDLAKOYA, MEDCHAL ROAD, HYDERABAD - 501401.

2022- 2026

CMR College of Engineering & Technology

(UGC Autonomous)

Kandlakoya, Medchal, Hyderabad - 501401

Department of Computer Science And Engineering

OPERATING SYSTEMS LAB PROJECT (A405509) - R22	
Project Batch No : 13	
Roll Number	**Name of the Student**
22H51A05H1	Meer Sameer
22H51A05H2	Midde Manu Priya
22H51A05H6	Perugu Sai Kumar

ABSTRACT

C program that demonstrates the creation of orphan and zombie processes on Unix-like operating systems. An orphan process is created when a parent process terminates, leaving its child process to be adopted by the `init` process (PID 1). A zombie process is created when a child process terminates but remains in the process table because its parent has not yet read its exit status. The program showcases these concepts through two scenarios, providing a practical understanding of process management and cleanup.

Guide Signature **Project Coordinator** **HOD**

AIM: Write C programs that illustrate how an orphan and zombie process are created.

EXECUTION:

Steps to follow:

1. Save the Program: Save the provided C code in a file named orphan_zombie.c.

2. Open a Terminal: Open your terminal application.

3. Navigate to the Directory: Navigate to the directory where you saved orphan_zombie.c

4. Compile the Program: Use gcc (GNU Compiler Collection) to compile the program:

5. Run the Executable:

Execute the compiled program:

6. Observe Orphan Process:

• The parent process will terminate, creating an orphan process.

• The orphan child process will be adopted by the init process (PID 1).

• Use the ps command in another terminal to observe the orphan process:

• Look for the orphan child process with PPID 1.

7. Observe Zombie Process:

• After creating the orphan process, the program will create a zombie process.

• The child process will terminate, but the parent process will delay collecting its exit status, leaving the child in a zombie state.

• Use the ps command in another terminal to observe the zombie process:

• Look for the child process in the Z (zombie) state.

8. Cleanup:

◦ After the parent process collects the child's exit status, the zombie process will be cleaned up.

◦ The program will print messages indicating the cleanup process.

CODE:

This program will create a named pipe and write a message to it.

```
#include <stdio.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

#include <unistd.h>

void create_orphan() {

    pid_t pid = fork();



    if (pid > 0) {

        printf("Parent process (PID: %d) is terminating\n", getpid());

        exit(0);

    } else if (pid == 0) {

        sleep(5);

        printf("Orphan child process (PID: %d) is running, adopted by init\n",
getpid());

    } else {

        perror("fork");

        exit(1);
```

```
    }
}
void create_zombie() {
    pid_t pid = fork();
    if (pid > 0) {
        sleep(5);
        printf("Parent process (PID: %d) is calling wait()\n", getpid());
        wait(NULL);
        printf("Parent process (PID: %d) finished collecting child status\n",
getpid());
    } else if (pid == 0) {
        printf("Child process (PID: %d) is terminating\n", getpid());
        exit(0);
    } else {
        perror("fork");
        exit(1);
    }
}
```

```
int main() {
    pid_t pid;
    create_orphan();

    sleep(10);
    create_zombie();
    sleep(10);
    return 0;
}
```

OUTPUT:

```
Parent process (PID: 12345) is terminating
Orphan child process (PID: 12346) is running, adopted by init
Child process (PID: 12347) is terminating
Parent process (PID: 12348) is calling wait()
Parent process (PID: 12348) finished collecting child status
```