PRACTICAL NO: 4

AIM:

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 lD

2. Write a program for process creation using C

- Orphan Process
- Zombie Process

3. Create the process using fork () system call

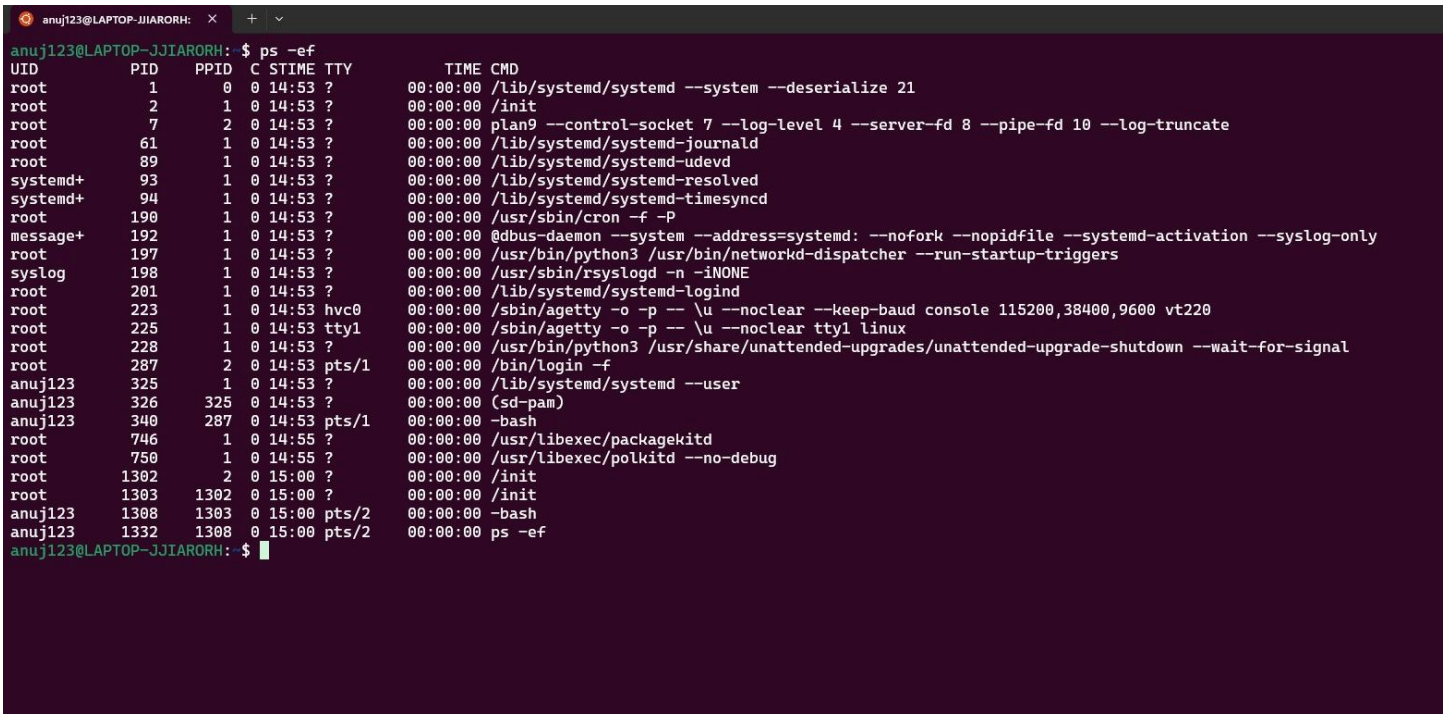
- Child Process creation
- Parent Process creation
- PPID and PID THEORY:

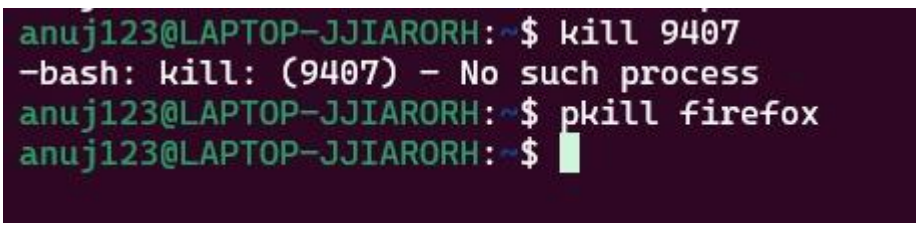
This practical focuses on understanding process management in the Linux operating system. A process is a program in execution, and Linux manages multiple processes through multitasking. Processes are created using the fork() system call, which forms parent and child processes. Each process is identified using PID and PPID to maintain hierarchy. The experiment also demonstrates process execution, termination, and the use of system calls like exec() and wait(). It further explains special cases such as orphan and zombie processes in Linux

PERFORMANCE:

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

COMMAND:





2. Write a program for process creation using C

Orphan Process:

An orphan process is a child process whose parent process terminates before the child finishes execution. The orphan process is adopted by the in it or system process.

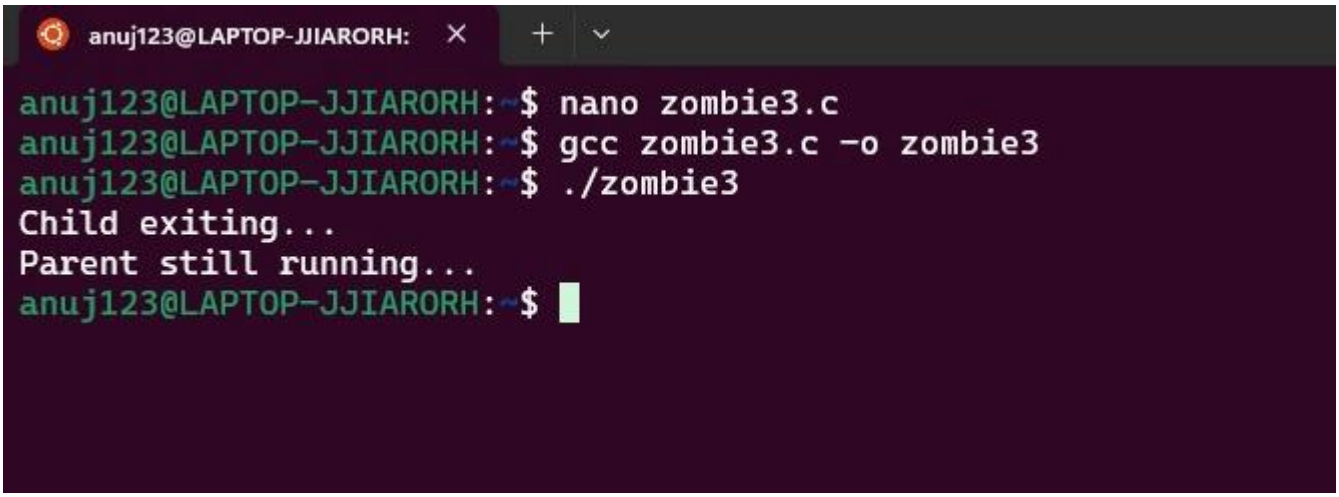
- orphan.c :
- output:


```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child
        sleep(5);
        printf("Child Process\n");
        printf("PID  = %d\n", getpid());
        printf("PPID = %d (Parent is init)\n", getppid());
    } else {
        // Parent
        printf("Parent exiting...\n");
    }

    return 0;
}
```

- **Zombie Process**

A zombie process is a child process that has completed execution but still remains in the process table because its parent has not read its exit status.

- **Zombie.c :**
- **Output:**




```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();

    if (pid == 0) {
        // Child
        printf("Child exiting...\n");
    } else {
        // Parent
        sleep(10); // Parent does not call wait()
        printf("Parent still running...\n");
    }

    return 0;
}
```

3. **Create the process using fork () system call**
    - Child Process creation
    - Parent Process creation
    - PPID and PID

Process:

A process is an instance of a program that is currently being executed in the operating system. It includes program code, data, stack and system resources.

Process ID (PID):

Process ID (PID) is a unique numerical identifier assigned by the operating system to each running process for identification and management.

Parent Process:

A parent process is a process that creates one or more child processes using system calls such as fork().

Child Process:

A child process is a newly created process that is generated by a parent process and executes independently.

Parent Process ID (PPID):

PPID shows the process ID of the parent of a running process.

- fork_demo:
- Output:


```c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;

    pid = fork();

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

    return 0;
}
```

4. Infinite loop process:

An infinite loop process is a process that runs continuously without termination until it is manually stopped by the user or system.

```
m309@m309-BY-OEM:~$ nano loop.c
m309@m309-BY-OEM:~$ gcc loop.c -o loop
m309@m309-BY-OEM:~$ ./loop
Running...
Running...
Running...
Running...
Running...
Running...
```

```
 GNU nano 7.2                                    loop.c *
#include <stdio.h>
#include <unistd.h>

int main() {
    while (1) {
        printf("Running...\n");
        sleep(1);
    }
    return 0;
}
```

Stopped the infinite loop: