

## **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 ID

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:

```
anuj123@LAPTOP-JJIARORH: ~$ ps -ef
UID          PID    PPID  C  STIME TTY          TIME CMD
root           1        0  0  14:53 ?        00:00:00 /lib/systemd/systemd --system --deserialize 21
root           2        1  0  14:53 ?        00:00:00 /init
root           7        2  0  14:53 ?        00:00:00 plan9 --control-socket 7 --log-level 4 --server-fd 8 --pipe-fd 10 --log-truncate
root          61        1  0  14:53 ?        00:00:00 /lib/systemd/systemd-journald
root          89        1  0  14:53 ?        00:00:00 /lib/systemd/systemd-udev
systemd+      93        1  0  14:53 ?        00:00:00 /lib/systemd/systemd-resolved
systemd+      94        1  0  14:53 ?        00:00:00 /lib/systemd/systemd-timesyncd
root         190        1  0  14:53 ?        00:00:00 /usr/sbin/cron -f -P
message+     192        1  0  14:53 ?        00:00:00 @dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
root         197        1  0  14:53 ?        00:00:00 /usr/bin/python3 /usr/bin/networkd-dispatcher --run-startup-triggers
syslog       198        1  0  14:53 ?        00:00:00 /usr/sbin/rsyslogd -n -iNONE
root        201        1  0  14:53 ?        00:00:00 /lib/systemd/systemd-logind
root        223        1  0  14:53 hvc0     00:00:00 /sbin/agetty -o -p -- \u --noclear --keep-baud console 115200,38400,9600 vt220
root        225        1  0  14:53 tty1     00:00:00 /sbin/agetty -o -p -- \u --noclear tty1 linux
root        228        1  0  14:53 ?        00:00:00 /usr/bin/python3 /usr/share/unattended-upgrades/unattended-upgrade-shutdown --wait-for-signal
root        287        2  0  14:53 pts/1    00:00:00 /bin/login -f
anuj123      325        1  0  14:53 ?        00:00:00 /lib/systemd/systemd --user
anuj123      326      325  0  14:53 ?        00:00:00 (sd-pam)
anuj123      340      287  0  14:53 pts/1    00:00:00 -bash
root        746        1  0  14:55 ?        00:00:00 /usr/libexec/packagekitd
root        750        1  0  14:55 ?        00:00:00 /usr/libexec/polkitd --no-debug
root       1302        2  0  15:00 ?        00:00:00 /init
root       1303      1302  0  15:00 ?        00:00:00 /init
anuj123     1308      1303  0  15:00 pts/2    00:00:00 -bash
anuj123     1332     1308  0  15:00 pts/2    00:00:00 ps -ef
anuj123@LAPTOP-JJIARORH: ~$
```

```
anuj123@LAPTOP-JJIARORH: ~$ ps -ef |grep firefox
anuj123      1342     1308  0  15:02 pts/2    00:00:00 grep --color=auto firefox
anuj123@LAPTOP-JJIARORH: ~$ ps -ef |grep firefox
anuj123      1350     1308  0  15:03 pts/2    00:00:00 grep --color=auto firefox
anuj123@LAPTOP-JJIARORH: ~$
```

```
anuj123@LAPTOP-JJIARORH: ~$ kill 9407
-bash: kill: (9407) - No such process
anuj123@LAPTOP-JJIARORH: ~$ pkill firefox
anuj123@LAPTOP-JJIARORH: ~$
```

## 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 init or system process.

➤ orphan.c :

➤ output:

```
anuj123@LAPTOP-JJIARORH:~$ nano orphan.c
anuj123@LAPTOP-JJIARORH:~$ gcc orphan.c -o orphan
anuj123@LAPTOP-JJIARORH:~$ ./ orphan
-bash: ./: Is a directory
anuj123@LAPTOP-JJIARORH:~$ ./orphan
Parent exiting...
anuj123@LAPTOP-JJIARORH:~$ Child Process
PID  = 1431
PPID = 1303 (Parent is init)
```

```
#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**

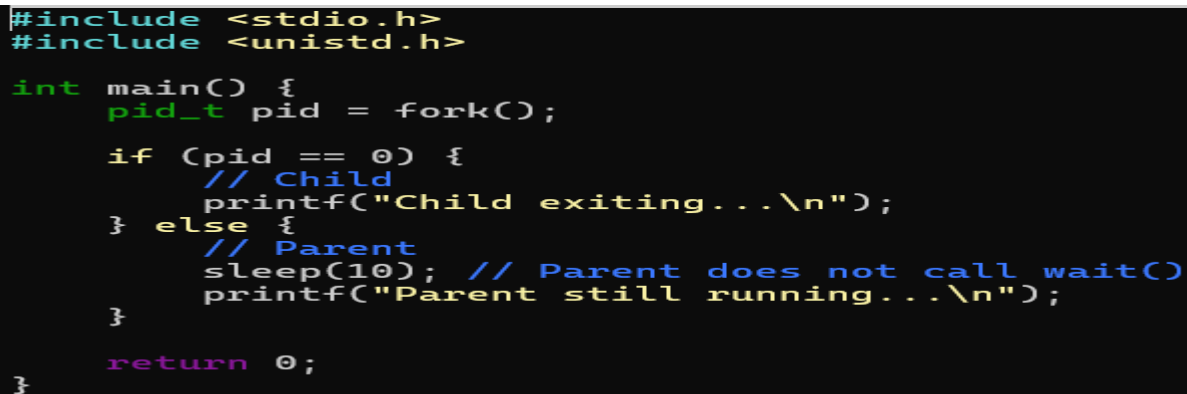
A zombie process is a child process that has completed execution but still remains in the process table because its parent has not read its exit status.

➤ **Zombie.c :**

➤ **Output:**



```
anuj123@LAPTOP-JJIARORH: ~$ nano zombie3.c
anuj123@LAPTOP-JJIARORH: ~$ gcc zombie3.c -o zombie3
anuj123@LAPTOP-JJIARORH: ~$ ./zombie3
Child exiting...
Parent still running...
anuj123@LAPTOP-JJIARORH: ~$
```



```
#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:

```
anuj123@LAPTOP-JJIARORH: ~$ nano fork.c
anuj123@LAPTOP-JJIARORH: ~$ gcc fork.c -o fork
gcc: error: unrecognized command-line option '-o'
anuj123@LAPTOP-JJIARORH: ~$ gcc fork.c -o fork
anuj123@LAPTOP-JJIARORH: ~$ ./fork
Parent Process
PID = 1517
Child PID = 1518
Child Process
PID = 1518
PPID = 1303
anuj123@LAPTOP-JJIARORH: ~$
```

```
GNU nano 0.2
#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:

[illegible]

