

1. 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

```
M ~

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ killall firefox
firefox: no process found

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ killall word
word: no process found

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ killall crome
crome: no process found

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ sleep 300
Terminated

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ sleep 300
.
.
.
MAYUR@LAPTOP-5DSA01RI MSYS ~
$ killall sleep
MAYUR@LAPTOP-5DSA01RI MSYS ~
$
```

```
M ~

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ pkill sleep
MAYUR@LAPTOP-5DSA01RI MSYS ~
$ sleep 300
Terminated

MAYUR@LAPTOP-5DSA01RI MSYS ~
$
```

The image shows two terminal windows side-by-side. The left window has a single command: \$ sleep 300. The right window shows a sequence of commands: \$ sleep 300 & [2] 2003, followed by \$ kill 2003, which results in -bash: kill: (2003) - No such process [2]+ Terminated sleep 300, and finally \$ |.

```
MAYUR@LAPTOP-5DSA01RI MSYS ~
$ sleep 300

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ sleep 300 &
[2] 2003

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ kill 2003

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ kill %2
-bash: kill: (2003) - No such process
[2]+ Terminated sleep 300

MAYUR@LAPTOP-5DSA01RI MSYS ~
$ |
```

2 .Write a program for process creation using C

- Orphan Process

- Zombie Process

orphan process

The image shows a terminal window with a C program named hello.c. The program uses fork() to create a child process. If the parent process exits, the child becomes an orphan. The code prints the parent's PID and the child's PID.

```
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>

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

    if (pid > 0) {
        // Parent process
        printf("Parent process exiting\n");
    } else {
        // Child process
        sleep(5);
        printf("Child process becomes orphan\n");
        printf("PID: %d, PPID: %d\n", getpid(), getppid());
    }
    return 0;
}
```

The image shows a terminal window with the following session:

```
MAYUR@LAPTOP-5DSA01RI MSYS :
nano nano hello.c

MAYUR@LAPTOP-5DSA01RI MSYS :
gcc hello.c -o hello

MAYUR@LAPTOP-5DSA01RI MSYS :
Parent process exiting
Child process becomes orphan
PID: 1924, PPID: 1
```

zombie Process

```
M ~
GNU nano 8.7                                     hello2.c
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid = fork();
    if (pid == 0) {
        // Child process
        printf("Child process exiting\n");
    } else {
        // Parent process
        sleep(10); // Parent does not call wait()
        printf("Parent process running\n");
    }
    return 0;
}
```

```
[MAYUR@LAPTOP MSYS ~
$ nano hello2.c

[MAYUR@LAPTOP MSYS ~
$ gcc hello2.c -o hello

[MAYUR@LAPTOP MSYS ~
$ ./hello2
-bash: ./hello2: No such file or directory

[MAYUR@LAPTOP MSYS ~
$ ./hello2.c
./hello2.c: line 4: syntax error near unexpected token `('
./hello2.c: line 4: `int main() {'

[MAYUR@LAPTOP MSYS ~
$ nano hello.c

[MAYUR@LAPTOP MSYS ~
$ gcc hello2.c --o hello

[MAYUR@LAPTOP MSYS ~
$ ./hello2
Child process exiting
Parent process running

[MAYUR@LAPTOP MSYS ~
$ ./hello2
Child process exiting
Parent process running

[MAYUR@LAPTOP MSYS ~
$
```

The screenshot shows a terminal window with a title bar 'M ~'. Inside, there are three separate terminal sessions. The first session shows the command 'ps -ef | grep Z' with output for the user 'mayur'. The second session shows the same command. The third session shows the command again. Below these sessions, there is a large black area with a vertical scrollbar on the right.

```
[MAYUR@LAPTOP MSYS ~
$ ps -ef | grep Z
mayur      1965      1946 pty1      19:45:20 grep Z

[MAYUR@LAPTOP MSYS ~
$ ps -ef | grep Z

[MAYUR@LAPTOP MSYS ~
$ ps -ef | grep Z
```

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

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

M ~

```
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>

int main() {
    pid_t pid;

    pid = fork();

    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;
}
```

AG Help ^O Write Out ^F Where Is ^K Cut ^T Execute ^C Location M-U Undo
AX Exit ^R Read File ^V Replace ^U Paste ^J Justify A/ Go To Line M-E Redo

19:54 31-01-2026

M ~

```
athar@THOR MSYS ~
$ nano hello3.c

athar@THOR MSYS ~
$ gcc hello3.c -o hello3

athar@THOR MSYS ~
$ ./hello3
Parent Process
Child Process
PID = 1980
Child PID = 1981
PID = 1981
PPID = 1980

athar@THOR MSYS ~
$ |
```

19:55 31-01-2026