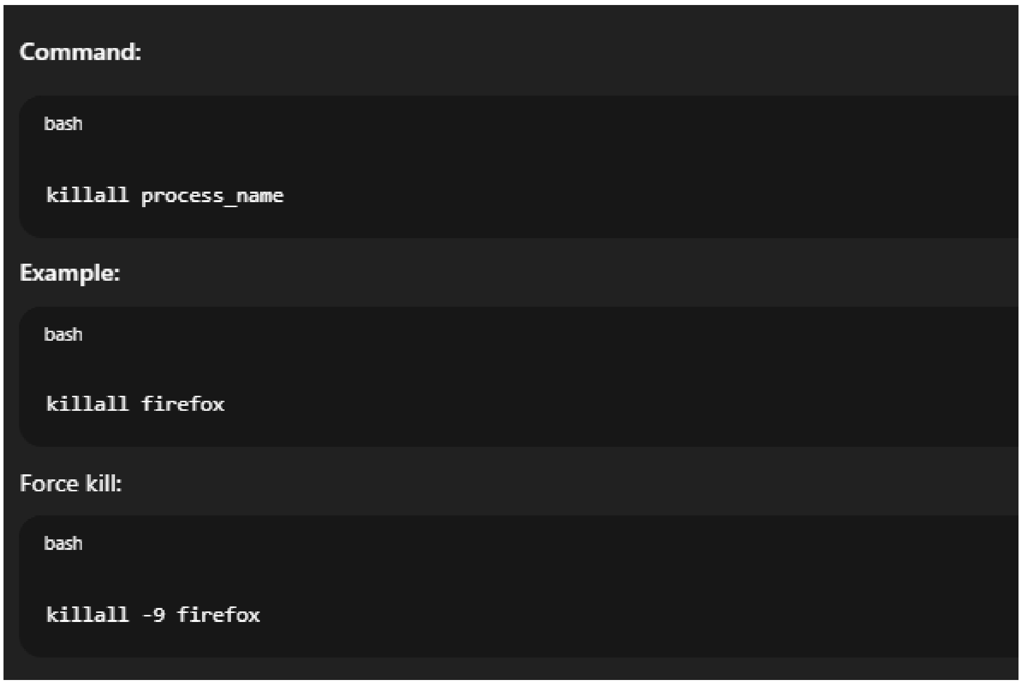
1. Linux Commands to Kill Processes

- ***Kill Process by Name:***
  Terminates all processes with the given name.



Command:

bash
```bash
killall process_name
```

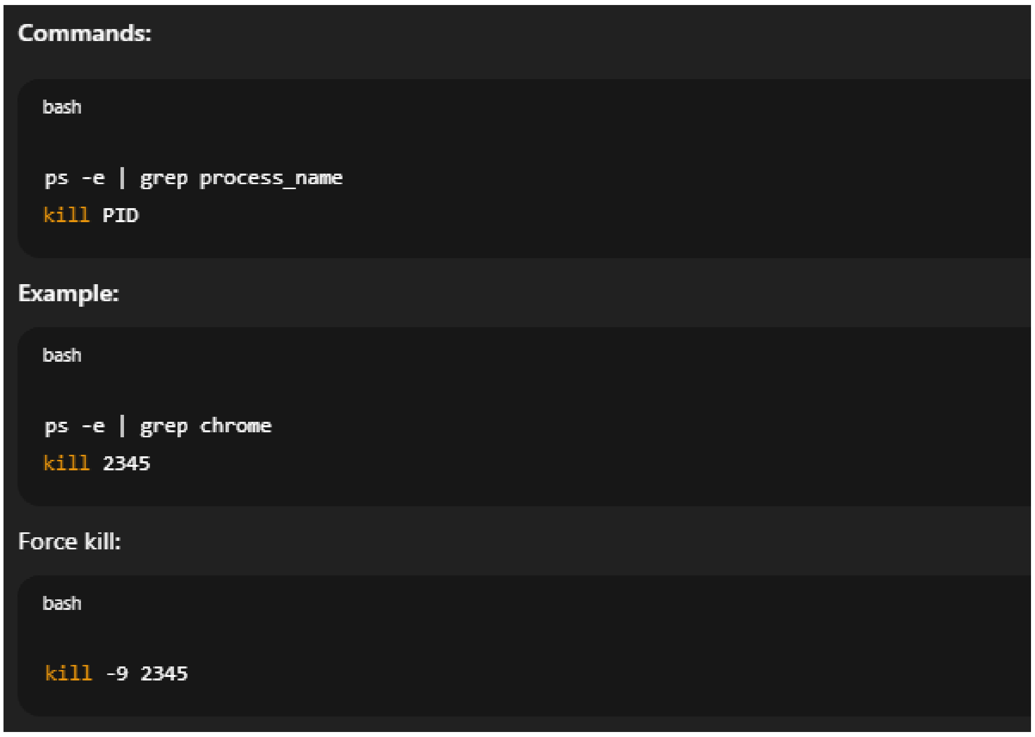
Example:

bash
```bash
killall firefox
```

Force kill:

bash
```bash
killall -9 firefox
```

- ***Kill Process Using Process Name (via PID):***
  First identifies the process ID using the process name and then terminates it.



Commands:

bash
```bash
ps -e | grep process_name
kill PID
```

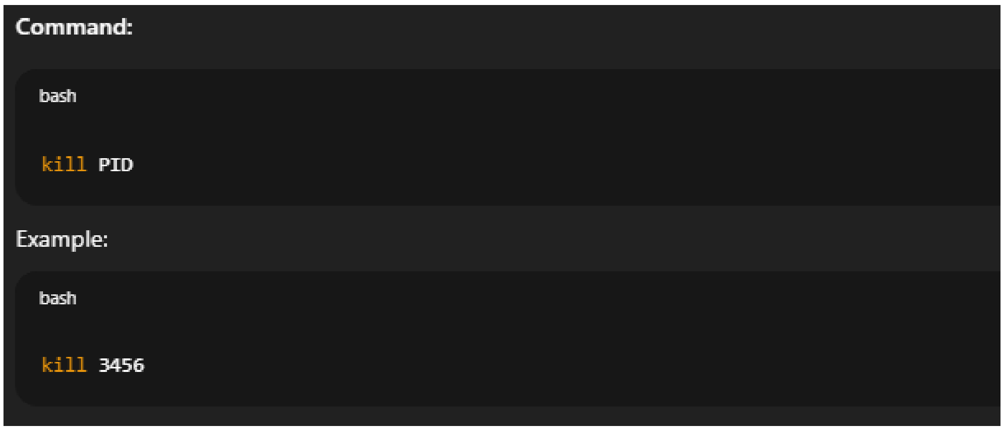
Example:

bash
```bash
ps -e | grep chrome
kill 2345
```

Force kill:

bash
```bash
kill -9 2345
```

- ***Kill a Single Process Using PID:***
  Terminates one specific process using its PID.



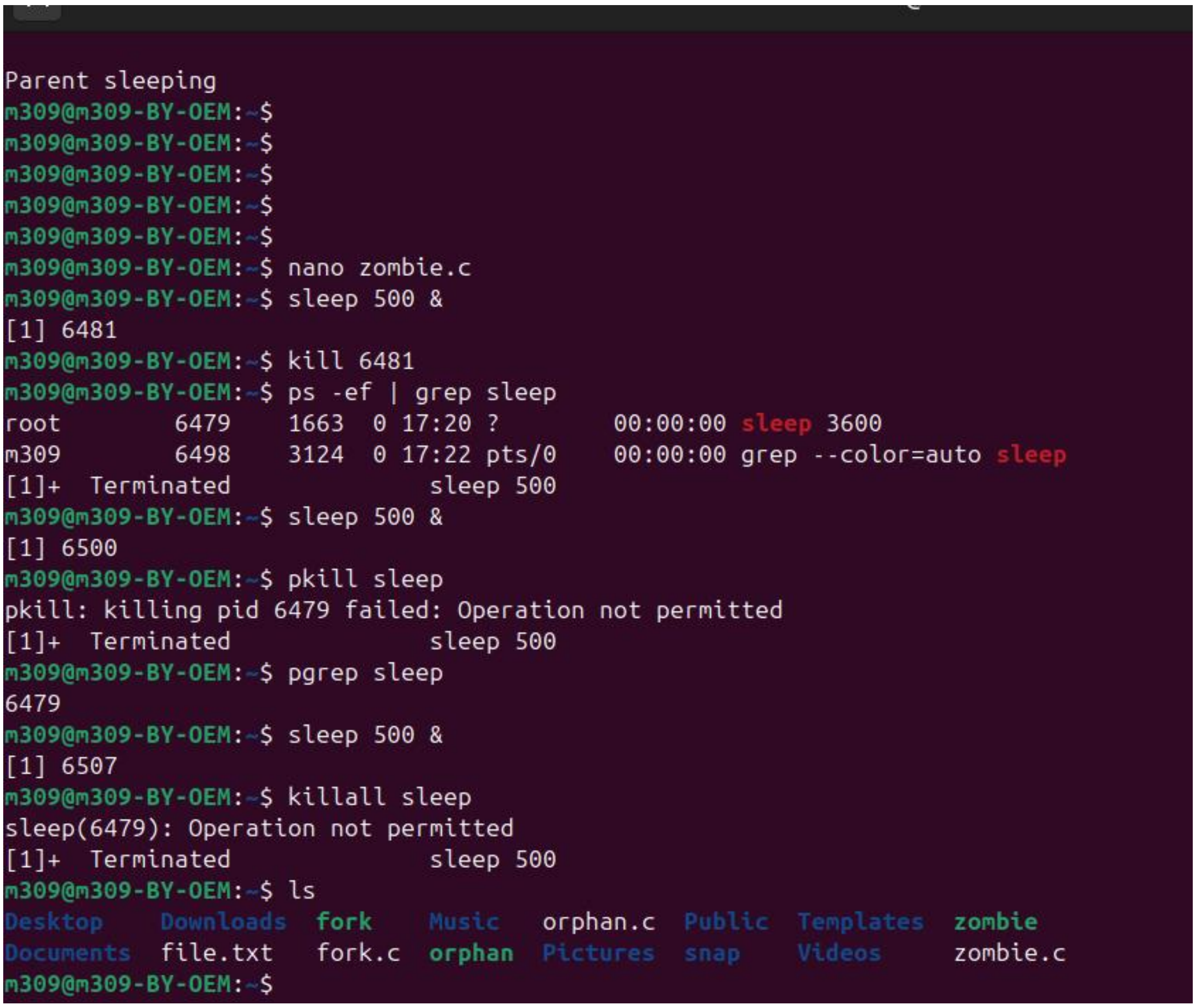
Command:

```bash
kill PID
```

Example:

```bash
kill 3456
```

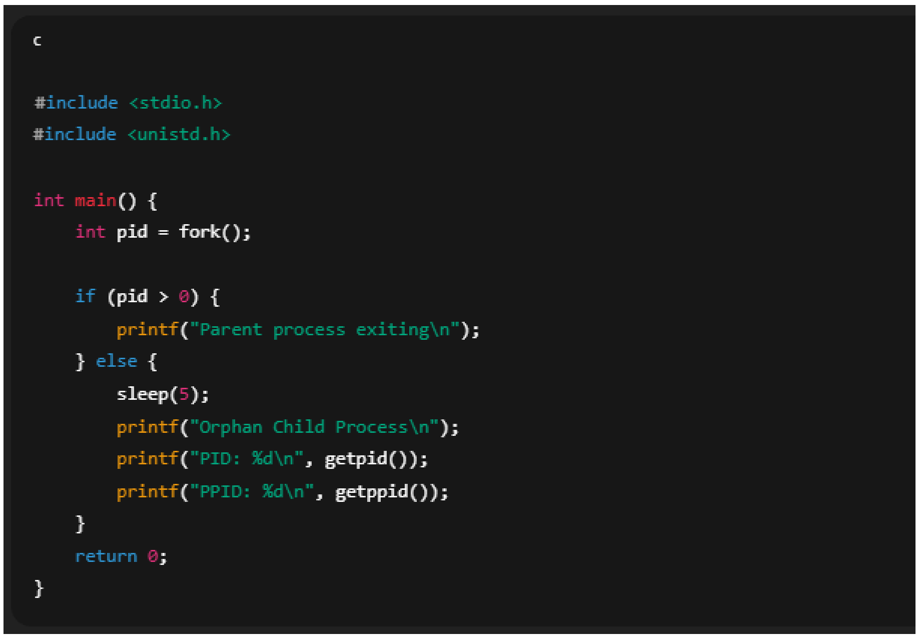
Output:



```
Parent sleeping
m309@m309-BY-OEM:~$
m309@m309-BY-OEM:~$
m309@m309-BY-OEM:~$
m309@m309-BY-OEM:~$
m309@m309-BY-OEM:~$
m309@m309-BY-OEM:~$ nano zombie.c
m309@m309-BY-OEM:~$ sleep 500 &
[1] 6481
m309@m309-BY-OEM:~$ kill 6481
m309@m309-BY-OEM:~$ ps -ef | grep sleep
root        6479    1663  0 17:20 ?        00:00:00 sleep 3600
m309        6498    3124  0 17:22 pts/0    00:00:00 grep --color=auto sleep
[1]+  Terminated              sleep 500
m309@m309-BY-OEM:~$ sleep 500 &
[1] 6500
m309@m309-BY-OEM:~$ pkill sleep
pkill: killing pid 6479 failed: Operation not permitted
[1]+  Terminated              sleep 500
m309@m309-BY-OEM:~$ pgrep sleep
6479
m309@m309-BY-OEM:~$ sleep 500 &
[1] 6507
m309@m309-BY-OEM:~$ killall sleep
sleep(6479): Operation not permitted
[1]+  Terminated              sleep 500
m309@m309-BY-OEM:~$ ls
Desktop    Downloads  fork    Music    orphan.c  Public  Templates  zombie
Documents  file.txt   fork.c  orphan   Pictures  snap    Videos     zombie.c
m309@m309-BY-OEM:~$
```

2. A) 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 process.

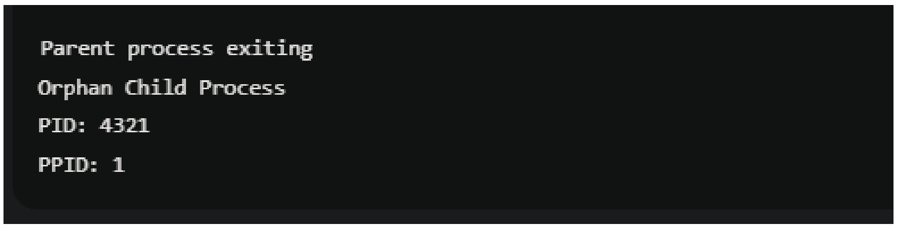


```c
#include <stdio.h>
#include <unistd.h>

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

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

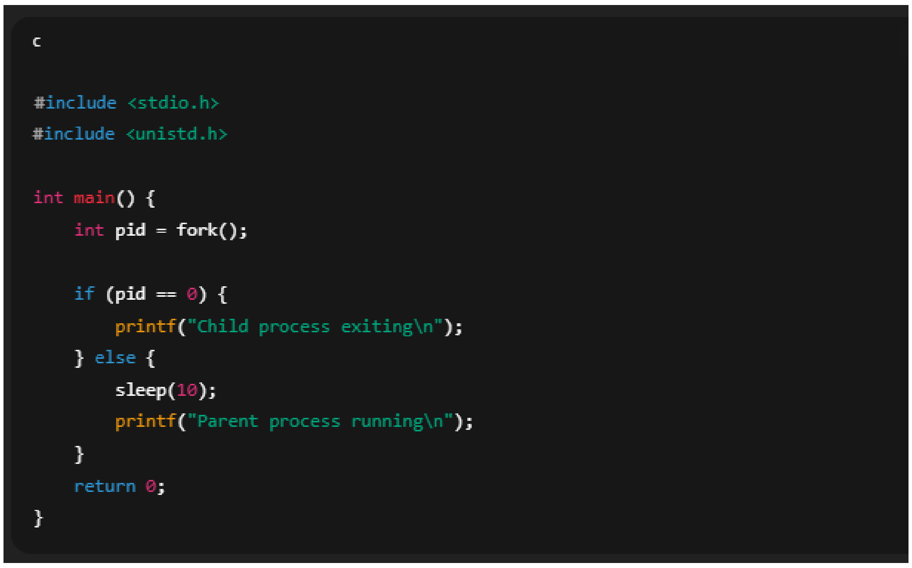
Output:



```
Parent process exiting
Orphan Child Process
PID: 4321
PPID: 1
```

(B) Zombie Process:

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

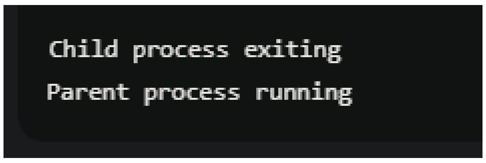


```c
#include <stdio.h>
#include <unistd.h>

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

    if (pid == 0) {
        printf("Child process exiting\n");
    } else {
        sleep(10);
        printf("Parent process running\n");
    }
    return 0;
}
```

Output:



```
Child process exiting
Parent process running
```

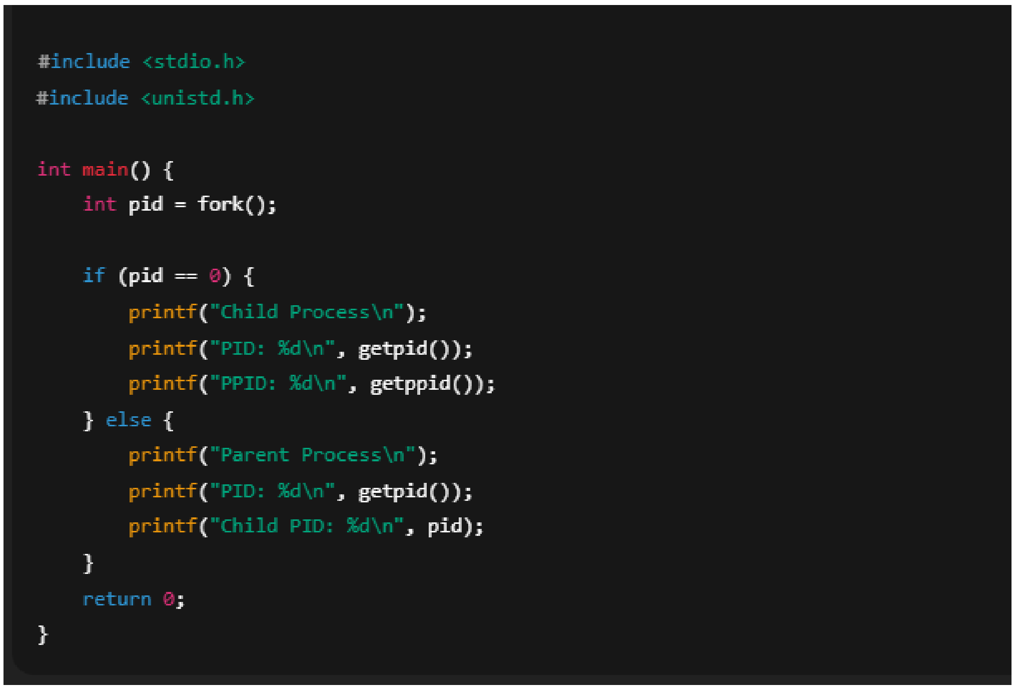
3.(A) Child Process:

A child process is a process that is created by another process using the fork () system call. The child process gets a new PID and executes independently of the parent process.

(B) Parent Process:

A parent process is the process that creates another process using the fork () system call. It receives the PID of the child process as the return value of fork ().

- ***IMP:***
  - fork () returns **0** → Child process
  - fork () returns **PID > 0** → Parent process

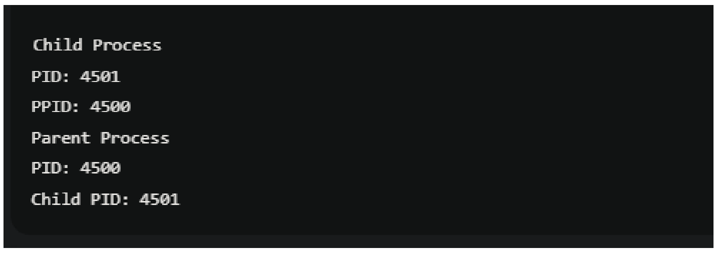


```c
#include <stdio.h>
#include <unistd.h>

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

    if (pid == 0) {
        printf("Child Process\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());
    } else {
        printf("Parent Process\n");
        printf("PID: %d\n", getpid());
        printf("Child PID: %d\n", pid);
    }
    return 0;
}
```

Output:



```
Child Process
PID: 4501
PPID: 4500
Parent Process
PID: 4500
Child PID: 4501
```