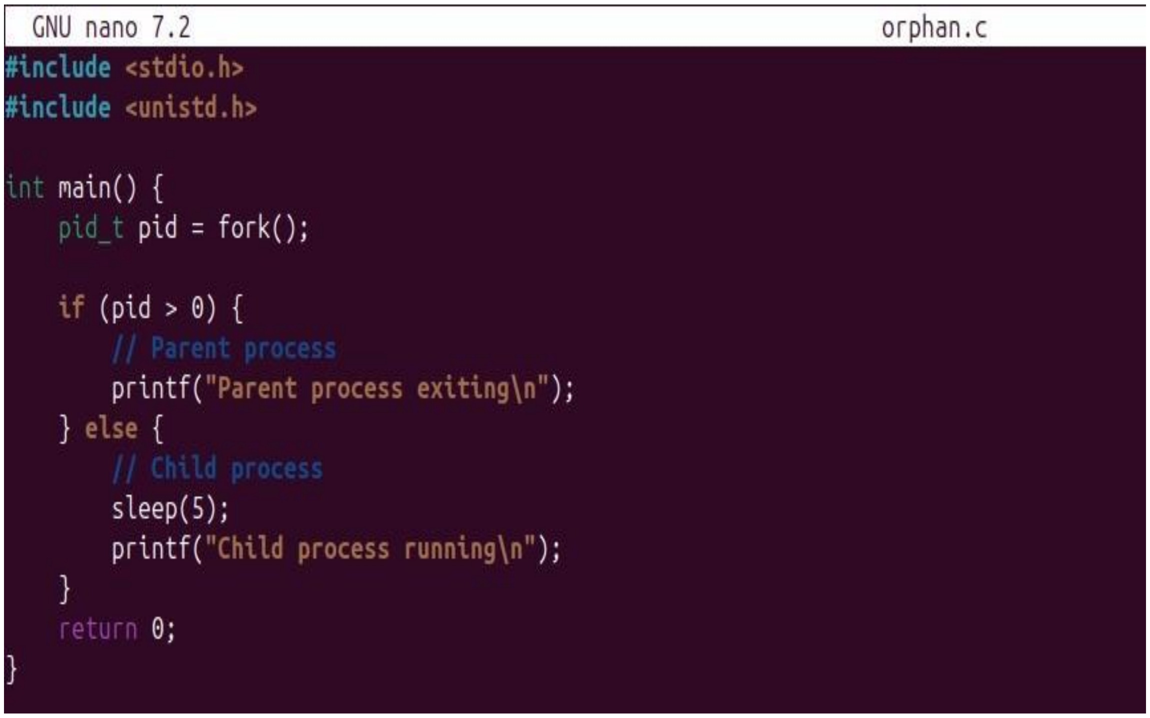
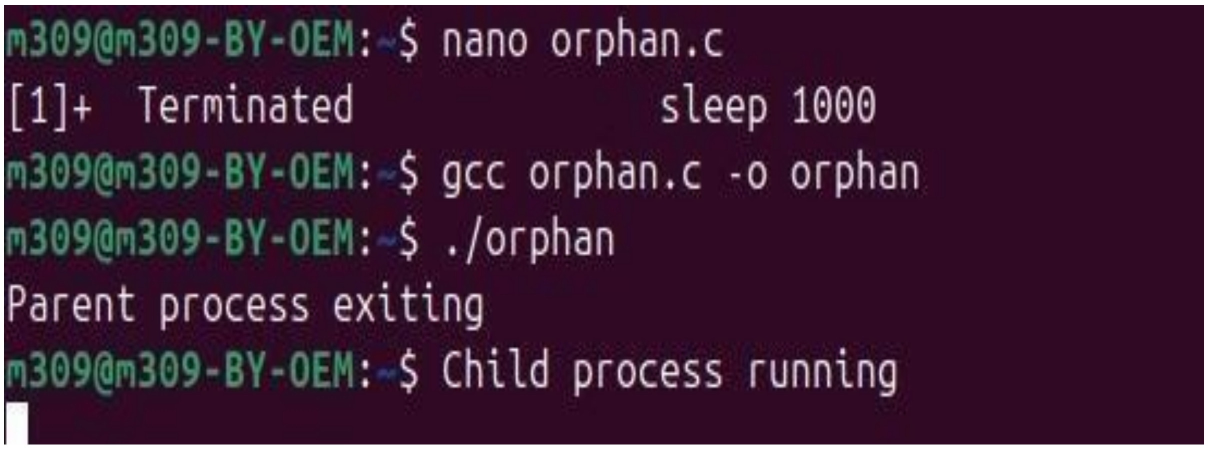
2 .Write a program for process creation using C

- **Orphan Process**

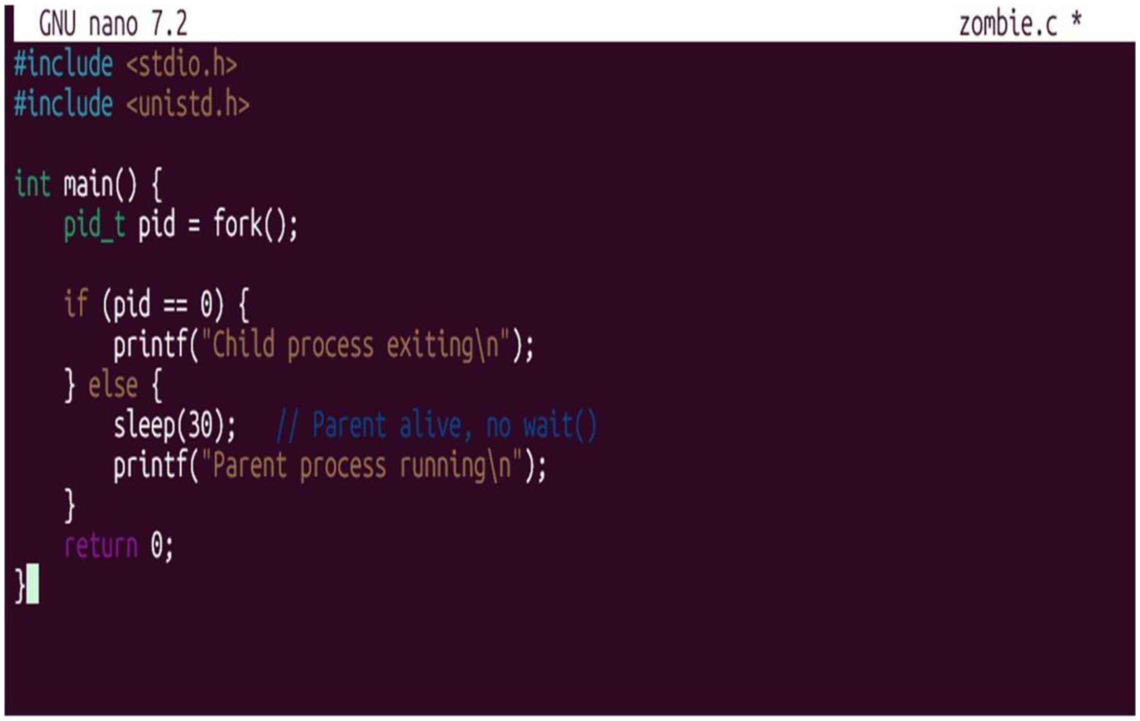


```
GNU nano 7.2                                                   orphan.c
#include <stdio.h>
#include <unistd.h>

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

    if (pid > 0) {
        // Parent process
        printf("Parent process exiting\n");
    } else {
        // Child process
        sleep(5);
        printf("Child process running\n");
    }
    return 0;
}
```

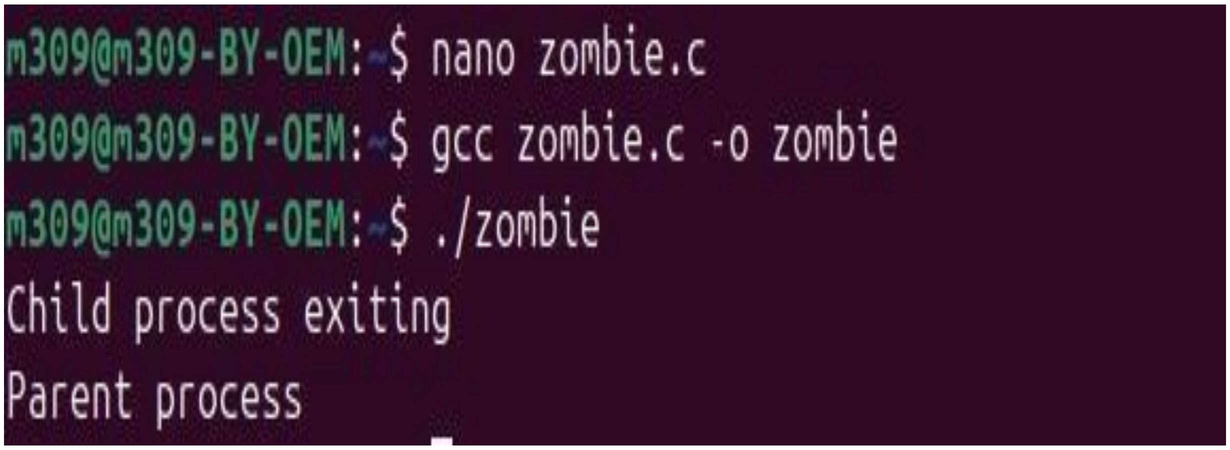


```
m309@m309-BY-OEM:~$ nano orphan.c
[1]+  Terminated              sleep 1000
m309@m309-BY-OEM:~$ gcc orphan.c -o orphan
m309@m309-BY-OEM:~$ ./orphan
Parent process exiting
m309@m309-BY-OEM:~$ Child process running
```

- **Zombie Process**



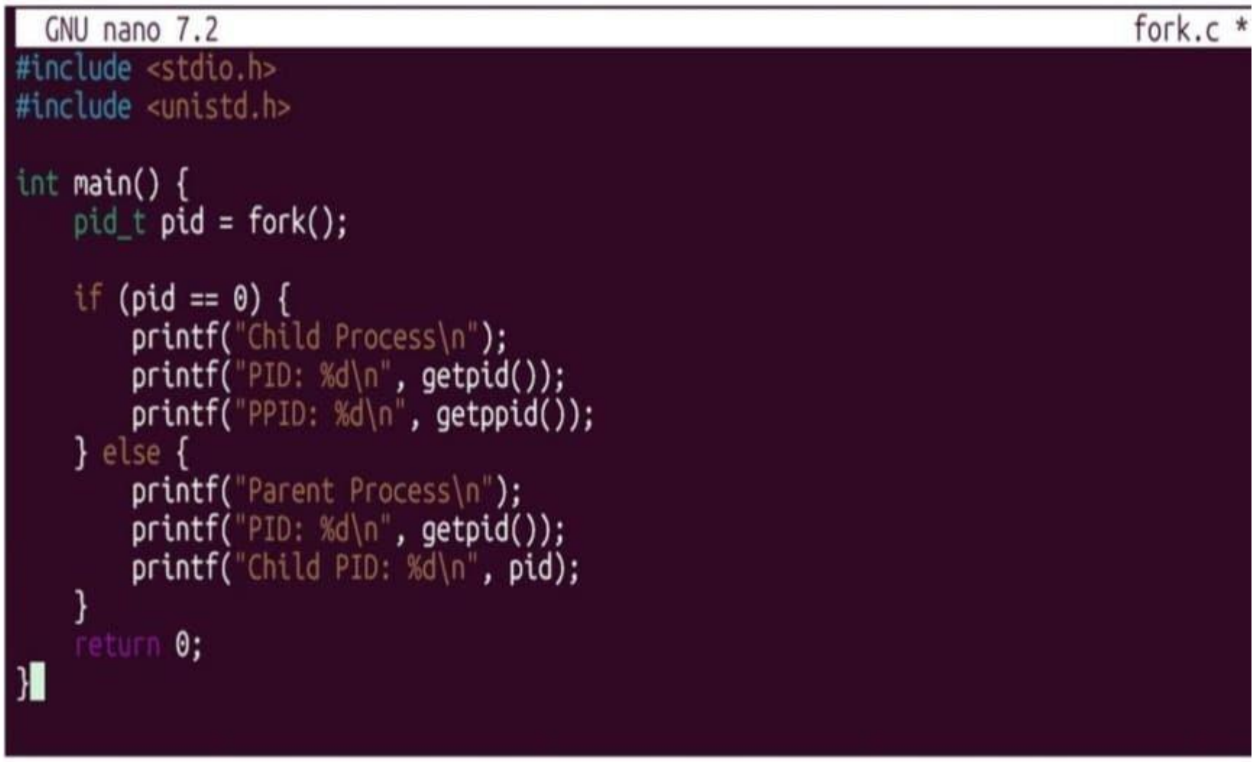
```
GNU nano 7.2                                                        zombie.c *
#include <stdio.h>
#include <unistd.h>

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

    if (pid == 0) {
        printf("Child process exiting\n");
    } else {
        sleep(30);    // Parent alive, no wait()
        printf("Parent process running\n");
    }
    return 0;
}
```



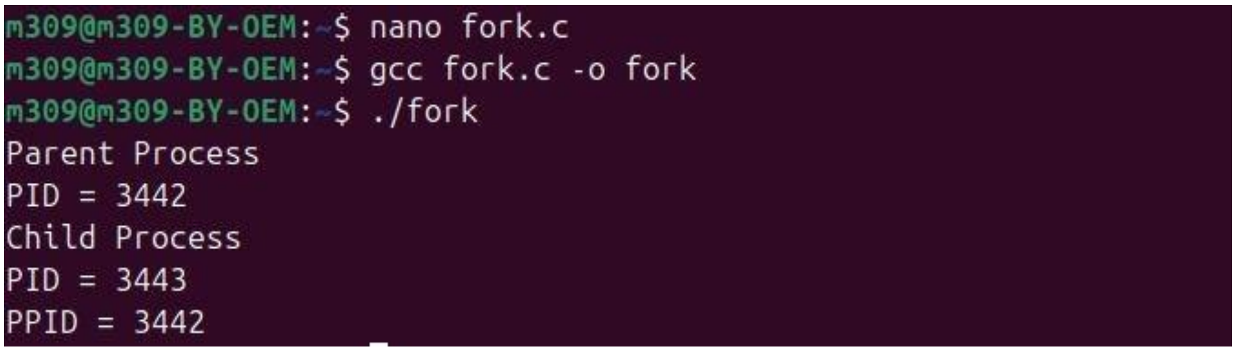
```
m309@m309-BY-OEM:~$ nano zombie.c
m309@m309-BY-OEM:~$ gcc zombie.c -o zombie
m309@m309-BY-OEM:~$ ./zombie
Child process exiting
Parent process
```

3. Create the process using fork () system call

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



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

int main() {
    pid_t 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;
}
```



```
m309@m309-BY-OEM:~$ nano fork.c
m309@m309-BY-OEM:~$ gcc fork.c -o fork
m309@m309-BY-OEM:~$ ./fork
Parent Process
PID = 3442
Child Process
PID = 3443
PPID = 3442
```