

- Zombie Process Code

A screenshot of a terminal window titled "program2.c *". The window shows the following C code:

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

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

    if(pid == 0)
    {
        printf("Child Process\n");
    }
    else
    {
        sleep(10);
    }
    return 0;
}
```

The code defines a simple program that creates a child process. If the child process (pid == 0) is executed, it prints "Child Process\n". Otherwise, it sleeps for 10 seconds and then returns 0.

- Orphan Process Code

A screenshot of a terminal window titled "program1.c". The window shows the following C code:

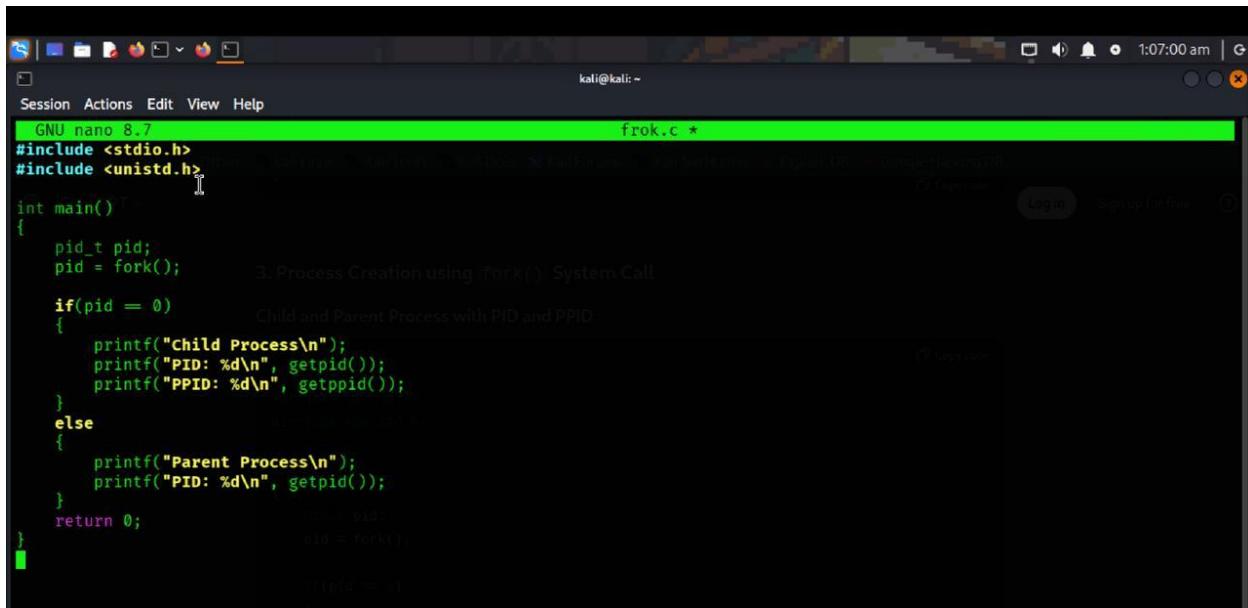
```
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>

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

    if(pid > 0)
    {
        sleep(2);
    }
    else
    {
        sleep(5);
        printf("Child Process\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());
    }
    return 0;
}
```

The code defines a program that creates a child process. If the parent process (pid > 0) is executed, it sleeps for 2 seconds. If the child process is executed, it sleeps for 5 seconds, then prints "Child Process\n", its PID, and its PPID.

- Fork



```
GNU nano 8.7
#include <stdio.h>
#include <unistd.h>
int main()
{
    pid_t pid;
    pid = fork();      // Process Creation using fork() System Call
    if(pid == 0)        // Child and Parent Process with PID and PPID
    {
        printf("Child Process\n");
        printf("PID: %d\n", getpid());
        printf("PPID: %d\n", getppid());
    }
    else
    {
        printf("Parent Process\n");
        printf("PID: %d\n", getpid());
    }
    return 0;           // pid = fork();
}                      // (pid == -1)
```