Proses Orphan

***Source Code* Lab6.c :**

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

int main()
{
   int pid;

   printf("I'am the original process with PID %d and PPID %d.\n",
   getpid(), getppid());
   pid = fork ();
   printf("################################################\n");
   if (pid != 0) /*Jika pid tidak nol, artinya saya parent*/
   {
      printf("I'am the parent with PID %d and PPID %d.\n",
      getpid(), getppid());
      printf("My child's PID is %d.\n", pid);
   }
   else /*Jika pid adalah 0, artinya saya child*/
   {
      sleep(4); /*Memastikan supaya parent lebih dulu di terminasi*/
      printf("I'm the child with PID %d and PPID %d.\n",
      getpid(), getppid());
   }
   printf("PID %d terminates.\n", getpid());
}
```

Hasil :

```
sailor_moon@tata-VirtualBox:~$ gcc Lab6.c -o Lab6.out
sailor_moon@tata-VirtualBox:~$ ./Lab6.out
I'am the original process with PID 2443 and PPID 2290.
################################################
I'am the parent with PID 2443 and PPID 2290.
My child's PID is 2444.
PID 2443 terminates.
################################################
sailor_moon@tata-VirtualBox:~$ I'm the child with PID 2444 and PPID 1395.
PID 2444 terminates.
```

Penjelasan :

Dalam *source code* diatas terdapat *original process* dengan PID 2443 dan PPID 2290. Kemudian fungsi fork diinisialisasi dalam varibael pid, apabila pid bernilai **BUKAN** 0 (nol) maka proses parent akan tetap berjalan dengan nilai PID dan PPID yang sama seperti *original process*. Saat pid dipanggil maka proses child akan dibuat. Setelah proses child dibuat maka pid akan bernilai 0 (nol) sehingga parent di-terminate menggunakan fungsi sleep (menangguhkan proses). Karena parent terminate sebelum child maka child akan menjadi *orphan* dan secara otomatis diadopsi oleh proses *'init'* asli yang PID-nya adalah 1.

Proses Zombie

***Source Code* Lab7.c :**

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

int main()
{
   int pid;

   pid = fork(); /*Duplikasi proses, child dan parent*/
   if (pid != 0) /*Jika pid tidak nol, artinya saya parent*/
   {
        while (1) /*Tidak terminate dan tidak mengeksekusi wait()*/
           sleep(5); /*Berhenti selama 100 detik*/
   }
   else /*pid adalah nol, artinya saya child*/
   {
        exit(3); /*exit dengan angka berapapun*/
   }
}
```

Hasil :

```
sailor_moon@tata-VirtualBox:~$ gcc Lab7.c -o Lab7.out
sailor_moon@tata-VirtualBox:~$ ./Lab7.out &
[1] 2464
sailor_moon@tata-VirtualBox:~$ ps
    PID TTY          TIME CMD
   2290 pts/0    00:00:00 bash
   2464 pts/0    00:00:00 Lab7.out
   2465 pts/0    00:00:00 Lab7.out <defunct>
   2470 pts/0    00:00:00 ps
sailor_moon@tata-VirtualBox:~$ kill 2464
[1]+  Terminated              ./Lab7.out
sailor_moon@tata-VirtualBox:~$ ps
    PID TTY          TIME CMD
   2290 pts/0    00:00:00 bash
   2473 pts/0    00:00:00 ps
sailor_moon@tata-VirtualBox:~$
```

Penjelasan :

Dalam *source code* diatas, fungsi `fork` diinisialisasi dalam varibael `pid` untuk menduplikasi proses, `child` dan `parent`. Apabila `PID` bernilai **BUKAN** `0(nol)` maka `parent` dikenai fungsi `sleep`. `Parent` tidak `terminate` (hidup) dan tidak mengeksekusi `wait()` maka kode pengembalian proses tidak akan pernah diterima dan proses anak menjadi zombie. Sedangkan apabila `PID` bernilai `0(nol)` maka akan di-`exit`. Hasil `run` pada terminal menunjukkan bahwa setelah *source code* dieksekusi muncul proses `parent` dan proses `child` yang merupakan zombie. Apabila `parent` di `terminate` atau di `kill` maka proses `parent` dan proses `child` yang menjadi zombie akan hilang.