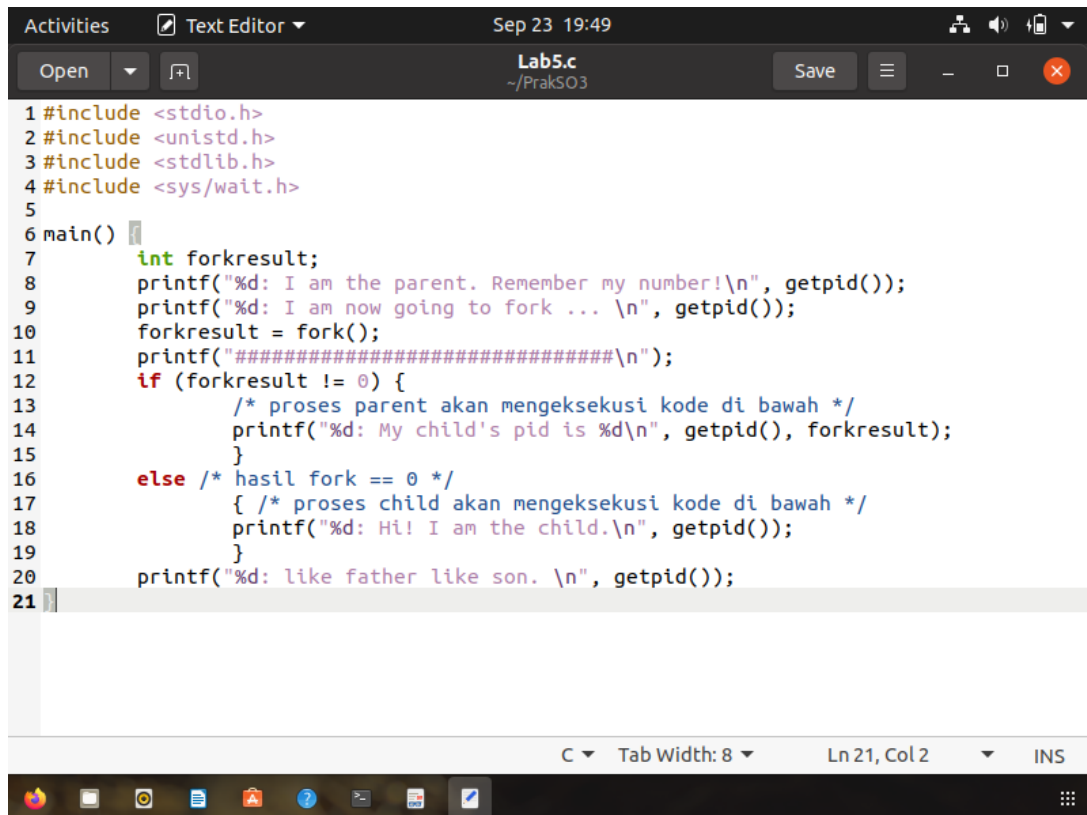


Nama :Rafli Azra Virendra Azhari
NIM : 24060119140080
Kelas : A1

Tugas 2 Praktikum Sistem Operasi

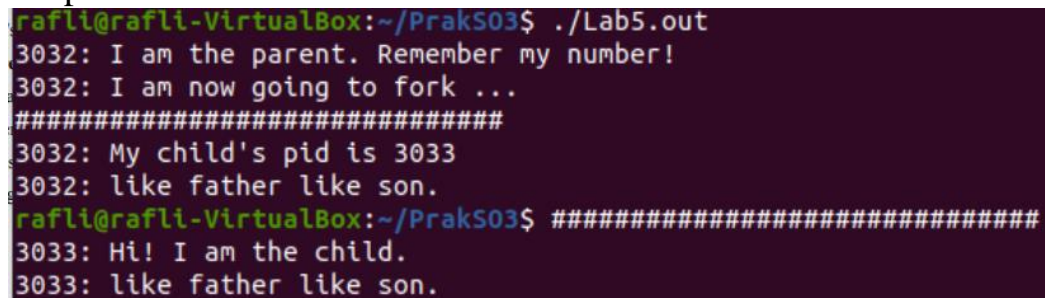
1. Lab5.c

- Isi Lab5.C



```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5
6 main()
7 {
8     int forkresult;
9     printf("%d: I am the parent. Remember my number!\n", getpid());
10    printf("%d: I am now going to fork ... \n", getpid());
11    forkresult = fork();
12    printf("#####\n");
13    if (forkresult != 0) {
14        /* proses parent akan mengeksekusi kode di bawah */
15        printf("%d: My child's pid is %d\n", getpid(), forkresult);
16    }
17    else /* hasil fork == 0 */
18    { /* proses child akan mengeksekusi kode di bawah */
19        printf("%d: Hi! I am the child.\n", getpid());
20    }
21    printf("%d: like father like son. \n", getpid());
22 }
```

- Output Lab5.c



```
rafli@rafli-VirtualBox:~/PrakS03$ ./Lab5.out
3032: I am the parent. Remember my number!
3032: I am now going to fork ...
#####
3032: My child's pid is 3033
3032: like father like son.
rafli@rafli-VirtualBox:~/PrakS03$ #####
3033: Hi! I am the child.
3033: like father like son.
```

- Kesimpulan Lab5.c

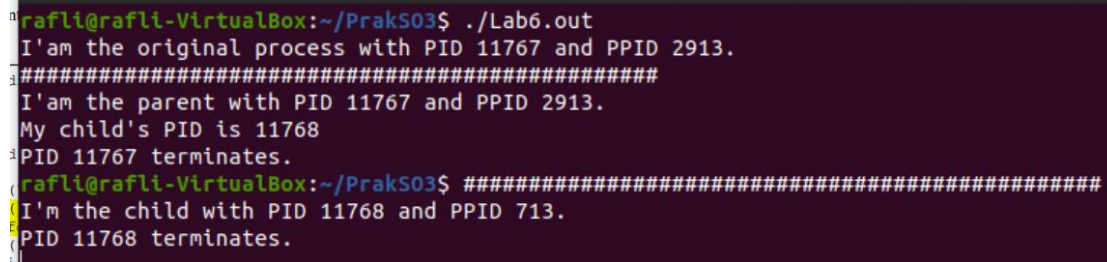
Jadi, dengan meng-include header `<unistd.h>` dan `<sys/wait>`, kita dapat menggunakan `fork()` karena 2 header tersebut digunakan untuk menunggu proses child. Integer `“forkresult”` digunakan untuk menyimpan hasil dari memanggil function `fork()`. Setelah itu print kalimat `“I am the parent. Remember my number!”` dan ambil process ID dari proses tersebut dengan `getpid()`. Dapat dilihat pada screenshot pid proses parent adalah 3032. Setelah itu gunakan integer `forkresult` untuk menginisiasi return value dari fungsi `fork()`. Selanjutnya, cek apakah hasil forking 0 atau tidak. Dapat dilihat kembali bahwa tidak 0, ini dikarenakan proses forking belum dijalankan, karena itu muncul `“My child’s pid is 3033”` dan `“like father like son”` dengan pid yang masih sama seperti parentnya. Setelah ini, forking baru jalan, dan akan dilakukan cek dimana hasil akan 0 yang berarti child sudah ada, dan keluarlah `“Hi! I am the child”` dan `“like father like son”` dengan pid yang sudah bertambah 1 karena ini adalah pid childnya.

2. Lab6.c

- Isi Lab6.c

```
1 #include <stdio.h>
2 main(){
3     int pid ;
4     printf("I'am the original process with PID %d and PPID %d.\n",
5         getpid(), getppid()) ;
6     pid = fork () ; /* Duplikasi proses, child dan parent */
7     printf("#####\n");
8     if ( pid != 0 ) /* jika pid tidak nol, artinya saya parent*/
9     {
10         printf("I'am the parent with PID %d and PPID %d.\n",
11             getpid(), getppid()) ;
12         printf("My child's PID is %d\n", pid ) ;
13     }
14     else /* jika pid adalah nol, artinya saya child */
15     {
16         sleep(4); /* memastikan supaya parent lebih dulu di terminasi*/
17         printf("I'm the child with PID %d and PPID %d.\n",
18             getpid(), getppid()) ;
19     }
20     printf ("PID %d terminates.\n", getpid()) ;
21 }
```

- Output Lab6.c



```
rafli@rafli-VirtualBox:~/PrakS03$ ./Lab6.out
I'am the original process with PID 11767 and PPID 2913.
#####
I'am the parent with PID 11767 and PPID 2913.
My child's PID is 11768
PID 11767 terminates.
rafli@rafli-VirtualBox:~/PrakS03$ #####
I'm the child with PID 11768 and PPID 713.
PID 11768 terminates.
```

- Kesimpulan Lab6.c

Pada Lab6.c akan dilakukan proses orphan, dimana proses child kehilangan parent prosesnya, karena sudah diterminasi terlebih dahulu daripada proses child-nya. Pertama program akan cek PID dan PPID proses original-nya, dapat dilihat pada screenshot bahwa PID adalah 11767 dan PPID adalah 2913. Setelah itu lakukan forking yang diinisialisasikan pada integer pid. Setelah itu dapat dilihat bahwa parent process terbuat dengan PID dan PPID yang sama dengan original process dan diprint “I’am the parent with PID 11767 and PPID 2913.” Juga, proses child telah terbuat dengan PID 11768. Seharusnya setelah “PID 11767 terminates” proses child langsung terbentuk dibawahnya, tetapi karena kita lakukan sleep(4) sebelum proses child, maka child tidak langsung terbentuk dan harus menunggu 4 detik. Pada saat ini proses parent diterminasi, sehingga proses child akan berjalan tanpa parent. Sebuah proses child tidak dapat berjalan tanpa parentnya, maka proses child akan diadopsi oleh init yang memiliki PPID 713. PPID yang seharusnya terjadi jika child tidak yatim adalah PID parentnya yaitu 11767 tetapi karena diadopsi oleh init, maka PPIDnya menjadi 713.

3. Lab7.c

- Isi Lab7.c

```
1#include <stdio.h>
2#include <unistd.h>
3#include <stdlib.h>
4int main ()
5{
6    int pid;
7    pid = fork(); /* Duplikasi proses, Child dan parent */
8    if (pid != 0) /* jika pid tidak nol, artinya saya parent */
9    {
10        while (1) /*Tidak Terminate dan tidak mengeksekusi wait()*/
11            sleep(100); /* berhenti selama 100 detik */
12    }
13    else /* pid adalah nol, artinya saya child */
14    {
15        exit(42); /* exit dengan angka berapapun */
16    }
17 }
```

- Output Lab7.c

```
rafli@rafli-VirtualBox:~/PrakS03$ ./Lab7.out > a.out &
[1] 2987
rafli@rafli-VirtualBox:~/PrakS03$ ps
  PID TTY          TIME CMD
 2957 pts/0        00:00:00 bash
 2987 pts/0        00:00:00 Lab7.out
 2988 pts/0        00:00:00 Lab7.out <defunct>
 2990 pts/0        00:00:00 ps
rafli@rafli-VirtualBox:~/PrakS03$ kill 2987
rafli@rafli-VirtualBox:~/PrakS03$ ps
  PID TTY          TIME CMD
 2957 pts/0        00:00:00 bash
 3010 pts/0        00:00:00 ps
[1]+  Terminated                  ./Lab7.out > a.out
rafli@rafli-VirtualBox:~/PrakS03$
```

- **Kesimpulan Lab7.c**

Bagian Lab7.c ini merupakan contoh dari proses zombie. Proses ini terjadi ketika parent mulai melakukan proses child dan proses child tersebut dihentikan, tetapi parent process tidak menerima exit code dari child. Karena ini parent harus menunggu hingga menerima exit code dari child dan selama menunggu, proses parent tidak akan menggunakan resource dan akan mati tetapi masih ada, karena ini proses ini dinamakan proses zombie. Tulisan <defunct> pada PID 2988 berarti proses ini adalah zombie. Proses dengan PID 2987 merupakan parent dari proses asli dan proses dengan PID 2988 adalah zombie.

Proses zombie akan terus berjalan di background hingga dibunuh(kill) oleh user atau sistem. Dengan membunuh/kill 2987, proses zombie dengan PID 2988 akan ikut mati karena telah kita terminate parentnya dan akan tidak exist lagi.