

Tugas 4 Pembuatan Proses

Muhammad Naufal Pratama

24060119130056

Lab A1

1. simpleFork.c

```
1 #include <unistd.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/wait.h>
5 #include <sys/types.h>
6
7 int main(void)
8 {
9     // process id
10    int pid,i,endvalue;
11    // use fork to create a new process
12    endvalue=1000;
13    printf("calling fork()\n");
14    pid=fork();
15    // check to see if fork worked
16    if (pid<0)
17    {
18        printf("Fork failed\n");
19        exit(0);
20    }
21    else if (pid ==0)
22    {
23        // in child process
```

```
22    else if (pid ==0)
23    {
24        // in child process
25        for (i=0; i<endvalue; i++)
26        {
27            printf("Child\n");
28            fflush(stdout);
29        }
30    }
31    else
32    {
33        // parent process
34        wait(NULL);
35        for(i=0; i<endvalue; i++)
36        {
37            printf("Parent\n");
38            fflush(stdout);
39        }
40        printf("Child Complete\n");
41        exit(0);
42    }
43 }
44
```

```
Child
Child
Child
Child
Child
Child
Child
Child
Child
Child
Child
Child
Child
Parent
Parent
Parent
Parent
Parent
Parent
Parent
```

2. exec in parent.c

[illegible][illegible]

Tujuan dari dilakukannya pemanggilan fungsi `exec` (dalam kasus `parent.c` ini, `execl`) adalah untuk mematikan proses yang sedang berjalan (*current proses*) dan **menggantinya dengan proses `exec`** sesuai dengan *command* yang kita berikan. Dalam contoh program `parent.c` kita memberikan perintah `sh` yang memiliki fungsi layaknya *command* interpreter (shell) dengan *flag* `-c` yang berguna untuk membaca string sebagai *command* yang kita beri parameter `./child` (maksudnya, kita jalankan program `./child` melalui *command* line string dengan bantuan *flag* `-c` dan `sh` sebagai penerjemah ke bahasa *command*/kalau tidak ada `sh`, “`./child`” dianggap sebagai string biasa bukan *command*).

3. **exec family**

a. **execl**

Inti dari *command* `execl` ini adalah untuk menjalankan dan mengeluarkan proses dari argumen/*command* yang diberikan dan kita perlu untuk memberikan **full path dimana *command*** itu berada (di executable binary file yang terdapat di `/bin`). Pertama kita memberikan full pathnya, kemudian *command* yang akan kita jalankan, diikuti beberapa flag jika diperlukan, dan diakhiri dengan `NULL`.

b. **execlp**

Hampir mirip dengan `execl`, tetapi *command* ini agak berbeda di bagian path yang diberikan. `execlp` tidak perlu memberikan full path dari executable binary file melainkan dengan menggunakan **path environment variable**. Jika suatu *command* sudah terdaftar di path environment variable kita dapat menggunakan *command* ini tanpa harus menuliskan full path. Untuk aturan penulisan berikutnya hampir sama, yaitu *command* yang ingin dijalankan, flag, dan tak lupa `NULL`.

c. **execv**

`execv` dapat menerima **array sebagai parameter** didalamnya dan memerlukan full path agar dapat menjalankan perintah di dalamnya. Urutannya sama seperti `exec` sebelumnya, yaitu full path, flag, dan `NULL`. Jika dilihat tidak ada `NULL` dalam `Uji3.c`, hal ini dikarenakan `NULL` dijadikan dalam bentuk terminated array `argv[0]`. Intinya yang membedakan dari `execl` ada di bagian bentuk parameter yang diberikan, kalau `execl` dalam bentuk fungsi yang dijadikan argumen, sedangkan `execv` dalam bentuk array.

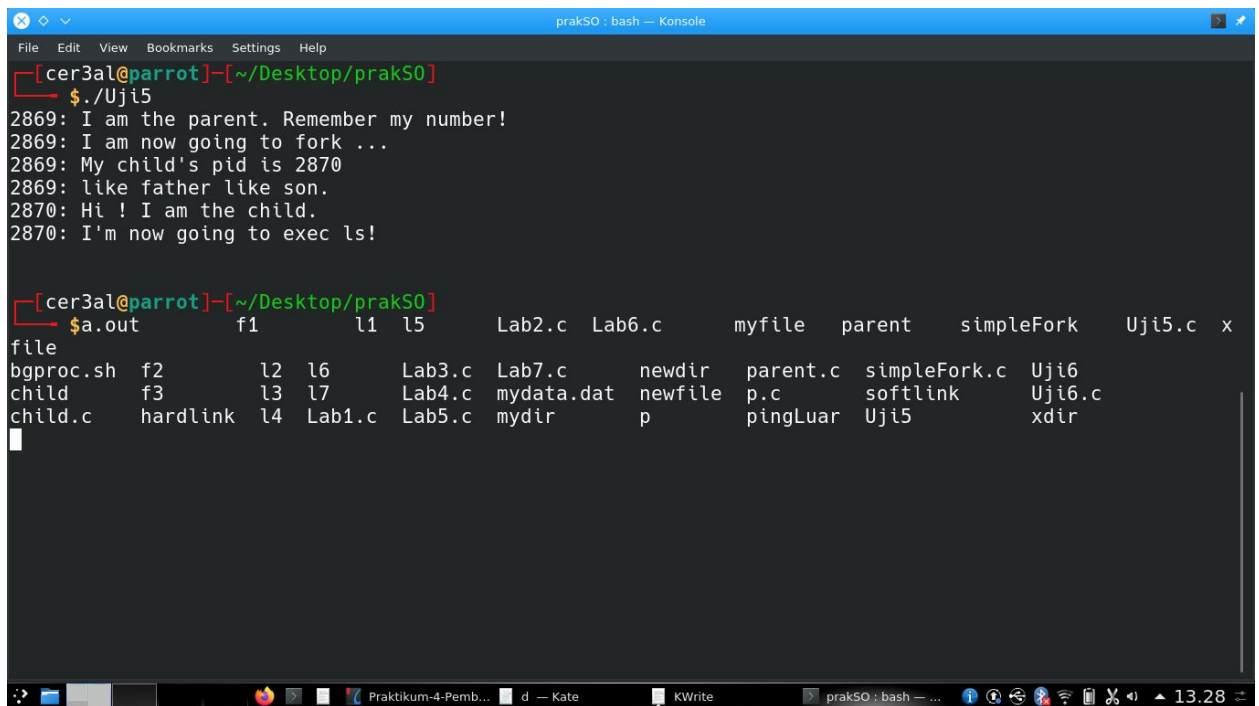
d. **execvp**

Sama seperti `execv` hanya saja kita tidak perlu full path hanya path environment variable (seperti `execlp`) dan juga menggunakan array sebagai parameter (seperti `execv`).

4. **NULL**

`Null` dalam *exec family* digunakan untuk **menghentikan proses** yang dijalankan di dalam *command* `exec`.

5. Uji5.c PID

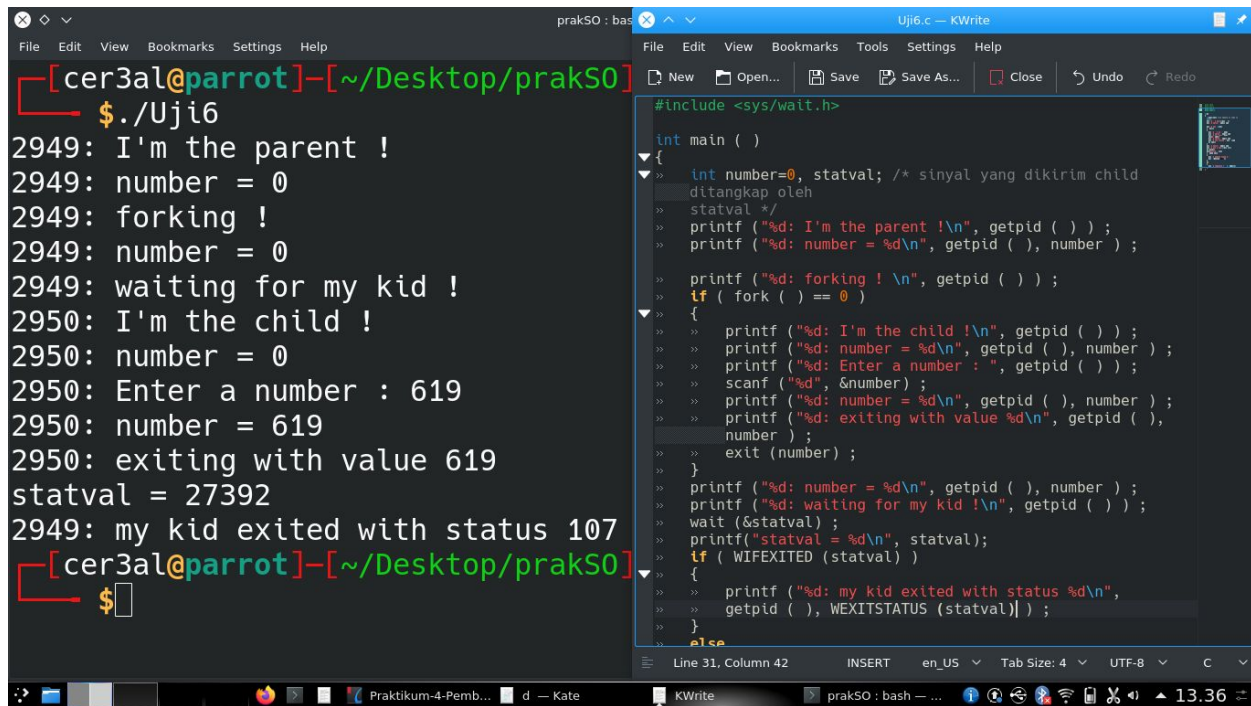


```
praksO : bash — Konsole
[cer3al@parrot]-[~/Desktop/praksO]
$ ./Uji5
2869: I am the parent. Remember my number!
2869: I am now going to fork ...
2869: My child's pid is 2870
2869: like father like son.
2870: Hi ! I am the child.
2870: I'm now going to exec ls!

[cer3al@parrot]-[~/Desktop/praksO]
$a.out      f1      l1  l5      Lab2.c  Lab6.c      myfile  parent  simpleFork  Uji5.c  x
file
bgproc.sh  f2      l2  l6      Lab3.c  Lab7.c      newdir  parent.c  simpleFork.c  Uji6
child      f3      l3  l7      Lab4.c  mydata.dat  newfile  p.c      softlink     Uji6.c
child.c    hardlink l4  Lab1.c  Lab5.c  mydir       p        pingLuar  Uji5        xdir
```

Exec **menggantikan** proses yang sedang berjalan dan menggantinya dengan proses yang dijalankan di dalam *command* exec. Sebelum exec dijalankan, proses forking berjalan seperti biasa, sedangkan ketika exec telah dilakukan maka proses forking digantikan dengan proses dari *command* yang diberikan di dalam exec atau perintah *ls* dan satu hal yang pasti, pengekseskuan exec **tidak mengubah PID** dari proses yang berjalan melainkan exec hanya menjalankan proses baru dengan **PID sama seperti sebelumnya**.

6. Uji6.c Exit() Wait()



```
[cer3al@parrot]--[~/Desktop/prakSO]
$ ./Uji6
2949: I'm the parent !
2949: number = 0
2949: forking !
2949: number = 0
2949: waiting for my kid !
2950: I'm the child !
2950: number = 0
2950: Enter a number : 619
2950: number = 619
2950: exiting with value 619
statval = 27392
2949: my kid exited with status 107
[cer3al@parrot]--[~/Desktop/prakSO]
$
```

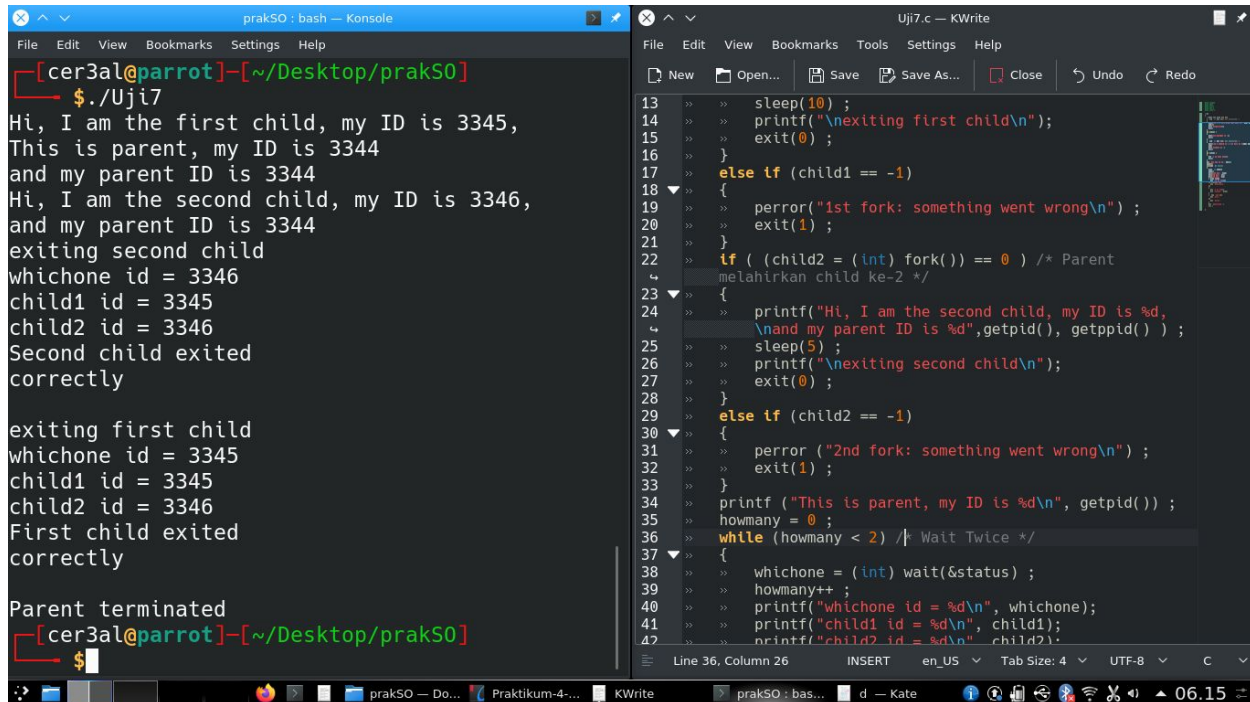
```
#include <sys/wait.h>

int main ( )
{
    int number=0, statval; /* sinyal yang dikirim child
    ditangkap oleh
    statval */
    printf ("%d: I'm the parent !\n", getpid ( ) );
    printf ("%d: number = %d\n", getpid ( ), number );

    printf ("%d: forking ! \n", getpid ( ) );
    if ( fork ( ) == 0 )
    {
        printf ("%d: I'm the child !\n", getpid ( ) );
        printf ("%d: number = %d\n", getpid ( ), number );
        printf ("%d: Enter a number : ", getpid ( ) );
        scanf ("%d", &number);
        printf ("%d: number = %d\n", getpid ( ), number );
        printf ("%d: exiting with value %d\n", getpid ( ),
        number );
        exit (number);
    }
    printf ("%d: number = %d\n", getpid ( ), number );
    printf ("%d: waiting for my kid !\n", getpid ( ) );
    wait (&statval);
    printf("statval = %d\n", statval);
    if ( WIFEXITED (statval) )
    {
        printf ("%d: my kid exited with status %d\n",
        getpid ( ), WEXITSTATUS (statval));
    }
    else
    {
        //
    }
}
```

Sinyal yang dikirim oleh *exit()* terbukti ditangkap oleh *wait()*. Dalam program tersebut, kita akan diminta untuk memasukkan sebuah angka. Setelah itu angka tersebut akan menjadi parameter dari *exit()* yang kemudian akan diberikan kepada *wait()* sebagai pointer. Nilai variabel *statval* merupakan angka yang diinput dikali 256. *WEXITSTATUS* akan mereturnkan low-order 8-bits exit status value dari si child. Sebagai bukti kontradiktif apakah *wait()* menerima lemparan dari *exit()* adalah melalui fakta bahwa baris “my kid exited with status” dijalankan dan untuk menjalankan baris ini diperlukan nilai dari *exit()* yang diberikan kepada *wait()*.

7. Uji7.c Multiple Child



```
praksO : bash — Konsole
[cer3al@parrot]~[/Desktop/praksO]
$ ./Uji7
Hi, I am the first child, my ID is 3345,
This is parent, my ID is 3344
and my parent ID is 3344
Hi, I am the second child, my ID is 3346,
and my parent ID is 3344
exiting second child
whichone id = 3346
child1 id = 3345
child2 id = 3346
Second child exited
correctly

exiting first child
whichone id = 3345
child1 id = 3345
child2 id = 3346
First child exited
correctly

Parent terminated
[cer3al@parrot]~[/Desktop/praksO]
$
```

```
Uji7.c — KWrite
File Edit View Bookmarks Tools Settings Help
New Open... Save Save As... Close Undo Redo

13 // sleep(10);
14 // printf("\nexiting first child\n");
15 // exit(0);
16 // }
17 // else if (child1 == -1)
18 // {
19 //     perror("1st fork: something went wrong\n");
20 //     exit(1);
21 // }
22 // if ( (child2 = (int) fork()) == 0 ) /* Parent
23 //     melahirkan child ke-2 */
24 // {
25 //     printf("Hi, I am the second child, my ID is %d,
26 //         \nand my parent ID is %d",getpid(), getppid() );
27 //     sleep(5);
28 //     printf("\nexiting second child\n");
29 //     exit(0);
30 // }
31 // else if (child2 == -1)
32 // {
33 //     perror("2nd fork: something went wrong\n");
34 //     exit(1);
35 // }
36 // printf("This is parent, my ID is %d\n", getpid());
37 // howmany = 0;
38 // while (howmany < 2) /* Wait Twice */
39 // {
40 //     whichone = (int) wait(&status);
41 //     howmany++;
42 //     printf("whichone id = %d\n", whichone);
43 //     printf("child1 id = %d\n", child1);
44 //     printf("child2 id = %d\n", child2);
45 // }
```

Berdasarkan Uji7.c parent dari kedua child (first child/3345 dan second child/3346) adalah parent dengan PID 3344. Hal ini dapat kita ketahui dengan mudah dengan cara **memanggil fungsi PPID pada child** yang telah dibuat. Ketika kita berikan PPID pada child pertama hasilnya adalah PID dari sang parent, yaitu 3344 dan untuk child kedua juga memberikan hasil PID sang parent, yaitu **3344** (Karena PID hasilnya sama maka artinya **kedua child memiliki parent yang sama**).

Parent dapat mengenali childnya melalui **nilai return yang dihasilkan dari si child** (nilai return child adalah PID parent). Dari nilai return ini akan masuk ke *exit()* dan dilemparkan ke *wait()* yang dimana wait akan memberikan nilai return PID dari child yang terminated. Dalam contoh ini, yang pertama terminated adalah child kedua baru kemudian diikuti child pertama. Hal ini disebabkan karena child pertama sleep lebih lama, yaitu 10 detik sedangkan child kedua hanya 5 detik.