

Nama : Nashirudin Baqiy

NIM : 24060119130045

Kelas : A1

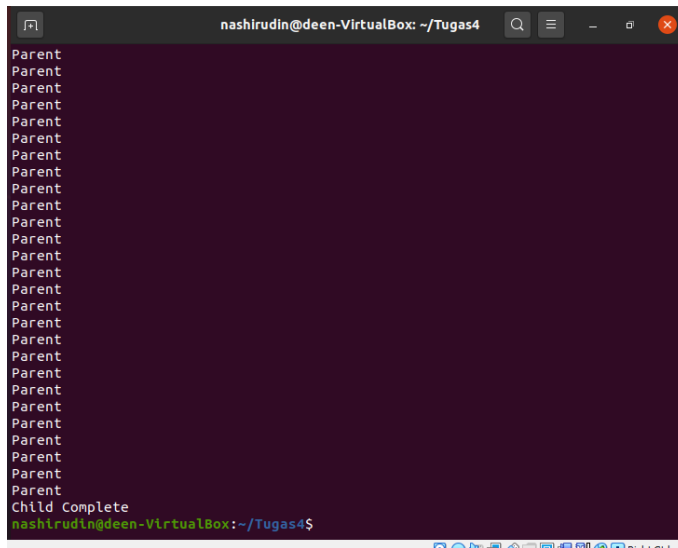
Tugas 4

Praktikum Sistem Operasi

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 int main(void)
7 {
8     // process id
9     int pid,i,endvalue;
10    // use fork to create a new process
11    endvalue=1000;
12    printf("calling fork()\n");
13    =fork();
14    // check to see if fork worked
15    if (pid<0)
16    {
17        printf("Fork failed\n");
18        exit(0);
19    }
20    else if (pid ==0)
21    {
22        // in child process
23        for (i=0; i<endvalue; i++)
24        {
25            printf("Child\n");
26            fflush(stdout);
27        }
28    }
29    else
30    {
31        // parent process
32        wait(NULL);
33        for(i=0; i<endvalue; i++)
34        {
35            printf("Parent\n");
36            fflush(stdout);
37        }
38        printf("Child Complete\n");
39        exit(0);
40    }
41 }
```

Output:



```
nashirudin@deen-VirtualBox: ~/Tugas4
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Parent
Child Complete
nashirudin@deen-VirtualBox: ~/Tugas4$
```

Mengetahui apakah dirinya sendiri sebagai proses parent atau child dari PID hasil forking. Child mengembalikan 0, parent mengembalikan PID childnya. Diketauilah parent dan childnya, karena parent melakukan wait() yang menunggu proses child selesai.

2. exec di parent.c

```
1 // Parent Program
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <error.h>
6 #include <signal.h>
7 #include <errno.h>
8 int main(void)
9 {
10     pid_t pid;
11     int status;
12     if((pid = fork()) < 0)
13     {
14         // probably out of processes
15         status = -1;
16     }
17     else if (pid == 0)
18     {
19         // in child so we execute process
20         // use the execl function to to run a shell an execute the child program
21         execl("/bin/sh", "sh", "-c", "./child", (char *)0);
22     }
23     while(1)
24     {
25         sleep(1);
26         printf("Parent");
27     }
28     printf("end of program");
29 }
```

Output:

[illegible]

Memastikan proses yang sedang berjalan lalu diganti proses exec. Kita memberi perintah sh yang memiliki fungsi command interpreter dengan flag -c membaca string sebagai command yang kita beri parameter ./child dan ./child dijalankan.

3. Keluarga exec

a. `excl`

Menjalankan dan mengeluarkan proses dari argument/command yang diberikan dan kita perlu untuk memberikan full path dimana command itu berada. Full path -> command -> flag (bila perlu) -> NULL

b. `execpl`

Hampir mirip `execl` sebelumnya, berbeda dibagian path yang diberi. Tidak perlu memberikan full path dari executable binary file tapi menggunakan path environment variable. Jika suatu command sudah terdaftar di path environment variable maka dapat digunakan tanpa menulis full path. Urutan penulisan sama dengan `execl`.

c. `execv`

Menerima array sebagai parameter, memerlukan full path agar dapat berjalan. Urutan penulisan sama seperti sebelumnya. Tidak ada NULL di Uji3.c karena NULL dijadikan bentuk terminated array argv[0].

d. `execvp`

Mirip `execv` tapi tidak perlu full path, hanya path environment variable. Seperti perbandingan `execl` dan `execlp` tapi parameternya array.

4. NULL

NULL dalam keluarga exec digunakan untuk menghentikan proses yang dijalankan di command exec.

5. Uji5.c PID

```
1#include <stdio.h>
2#include <unistd.h>
3#include <stdlib.h>
4#include <sys/wait.h>
5int main ( )
6{
7    int forkresult ;
8    printf ("%d: I am the parent. Remember my number!\n", getpid ( ) ) ;
9    printf ("%d: I am now going to fork ... \n", getpid ( ) ) ;
10   forkresult = fork ( ) ;
11   if (forkresult != 0)
12   { /* parent akan mengeksekusi kode ini */
13       printf ("%d: My child's pid is %d\n", getpid ( ), forkresult ) ;
14   }
15   else /* forkresult == 0 */
16   { /* child akan mengeksekusi kode ini */
17       printf ("%d: Hi ! I am the child.\n", getpid ( ) ) ;
18       printf ("%d: I'm now going to exec ls!\n\n", getpid ( ) ) ;
19       execlp ("ls", "ls", NULL) ;
20       printf ("%d: AAAAH ! ! My EXEC failed ! ! !\n", getpid ( ) ) ;
21       exit (1) ;
22   }
23   printf ("%d: like father like son. \n", getpid ( ) ) ;
24}
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas4$ ./uji5
3785: I am the parent. Remember my number!
3785: I am now going to fork ...
3785: My child's pid is 3786
3785: like father like son.
nashirudin@deen-VirtualBox:~/Tugas4$ 3786: Hi ! I am the child.
3786: I'm now going to exec ls!

child child.c parent parent.c simpleFork simpleFork.c uji5 uji5.c uji6.c uji7.c
```

Menggunakan exec untuk mengganti proses yang sedang berjalan yaitu forking. Command exec mengandung perintah ls maka ls dijalankan.

6. Uji6.c exit() tertangkap wait()

```
1#include <stdio.h>
2#include <unistd.h>
3#include <stdlib.h>
4#include <sys/wait.h>
5
6int main ( )
7{
8    int number=0, statval; /* sinyal yang dikirim child ditangkap oleh statval */
9    printf ("%d: I'm the parent !\n", getpid ( ) ) ;
10   printf ("%d: number = %d\n", getpid ( ), number ) ;
11   printf ("%d: forking ! \n", getpid ( ) ) ;
12   if ( fork ( ) == 0 )
13   {
14       printf ("%d: I'm the child !\n", getpid ( ) ) ;
15       printf ("%d: number = %d\n", getpid ( ), number ) ;
16       printf ("%d: Enter a number : ", getpid ( ) ) ;
17       scanf ("%d", &number) ;
18       printf ("%d: number = %d\n", getpid ( ), number ) ;
19       printf ("%d: exiting with value %d\n", getpid ( ), number ) ;
20       exit (number) ;
21   }
22   printf ("%d: number = %d\n", getpid ( ), number ) ;
23   printf ("%d: waiting for my kid !\n", getpid ( ) ) ;
24   wait (&statval) ;
25   printf("statval = %d\n", statval);
26   if ( WIFEXITED (statval) )
27   {
28       printf ("%d: my kid exited with status %d\n",
29               getpid ( ), WEXITSTATUS (statval) ) ;
30   }
31   else
32   {
33       printf ("%d: My kid was killed off ! ! \n", getpid ( ) ) ;
34   }
35}
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas4$ ./uji6
3797: I'm the parent !
3797: number = 0
3797: forking !
3797: number = 0
3797: waiting for my kid !
3798: I'm the child !
3798: number = 0
3798: Enter a number : 567
3798: number = 567
3798: exiting with value 567
statval = 14080
3797: my kid exited with status 55
```

Exit() ditangkap oleh wait(). Kita diminta sebuah angka, angka tersebut menjadi parameter exit() kemudian diberikan kepada wait() sebagai pointer. Statval merupakan angka yang diinput dikali 256. WEXITSTATUS mereturn low order 8bits exit status value dari child. "my kid exited with status 55" merupakan bukti nilai exit() diberikan kepada wait().

7. Uji7.c

```
1 #include <sys/types.h>
2 #include <stdio.h>
3 #include <unistd.h>
4 #include <stdlib.h>
5 #include <sys/wait.h>
6 int main ( )
7 {
8     int howmany, status, whichone, child1, child2 ;
9     if ( (child1 = (int) fork()) == 0 ) /* Parent melahirkan child ke-1 */
10     {
11         printf("Hi, I am the first child, my ID is %d, and my parent ID is %d\n", getpid(), getppid() ) ;
12         sleep(10) ;
13         printf("\nextiting first child\n");
14         exit(0) ;
15     }
16     else if (child1 == -1)
17     {
18         perror("1st fork: something went wrong\n") ;
19         exit(1) ;
20     }
21     if ( (child2 = (int) fork()) == 0 ) /* Parent melahirkan child ke-2 */
22     {
23         printf("Hi, I am the second child, my ID is %d, and my parent ID is %d\n", getpid(), getppid() ) ;
24         sleep(5) ;
25         printf("\nextiting second child\n");
26         exit(0) ;
27     }
28     else if (child2 == -1)
29     {
30         perror ("2nd fork: something went wrong\n") ;
31         exit(1) ;
32     }
33     printf ("This is parent, my ID is %d\n", getpid()) ;
34     howmany = 0 ;
35     while (howmany < 2) /* Wait Twice */
36     {
37         whichone = (int) wait(&status) ;
38         howmany++ ;
39         printf("whichone id = %d\n", whichone);
40         printf("child1 id = %d\n", child1);
41         printf("child2 id = %d\n", child2);
42         if (whichone == child1)
43             printf ("First child exited\n") ;
44         else if (whichone == child2)
45             printf ("Second child exited\n") ;
46         else
47         {
48             printf ("not child exited\n");
49             printf ("whichone = %d\n", whichone);
50         }
51         if ( (status & 0xffff) == 0 )
52             printf ("correctly\n") ;
53         else
54             printf ("incorrectly\n") ;
55     }
56     printf ("\nParent terminated\n");
57     return 0;
58 }
```

Output:

```
nashirudin@deen-VirtualBox:~/Tugas4$ ./uji7
This is parent, my ID is 3806
Hi, I am the second child, my ID is 3808, and my parent ID is 3806
Hi, I am the first child, my ID is 3807, and my parent ID is 3806

exiting second child
whichone id = 3808
child1 id = 3807
child2 id = 3808
Second child exited
correctly

exiting first child
whichone id = 3807
child1 id = 3807
child2 id = 3808
First child exited
correctly

Parent terminated
```

Parent membuat first child dan second child. Second child teroutput pertama karena first child memiliki sleep lebih lama dibanding first child. Child menunggu parent muncul sebelum child muncul karena kedua child sleep. Kemudian parent menunggu child exiting dari child terakhir ke child pertama. Parent mengenali child melalui nilai return yang dihasilkan child. Nilai ini masuk ke exit() kemudian diberikan kepada wait() maka wait memberikan nilai return PID dari child yang akan diterminasi.