

Manajemen Proses



NIM : 11321014

NAMA : Yosafat Hazael Tambun

D3 TEKNOLOGI INFORMASI

A. Teori

1. Pertanyaan berikut terkait dengan konsep-konsep dasar Proses
 - a. Definisikan program.
 - b. Definisikan proses.
 - c. Definisikan Zombie Process
 - d. Definisikan Orphan Process
 - e. Jelaskanlah Process Control Block (PCB)

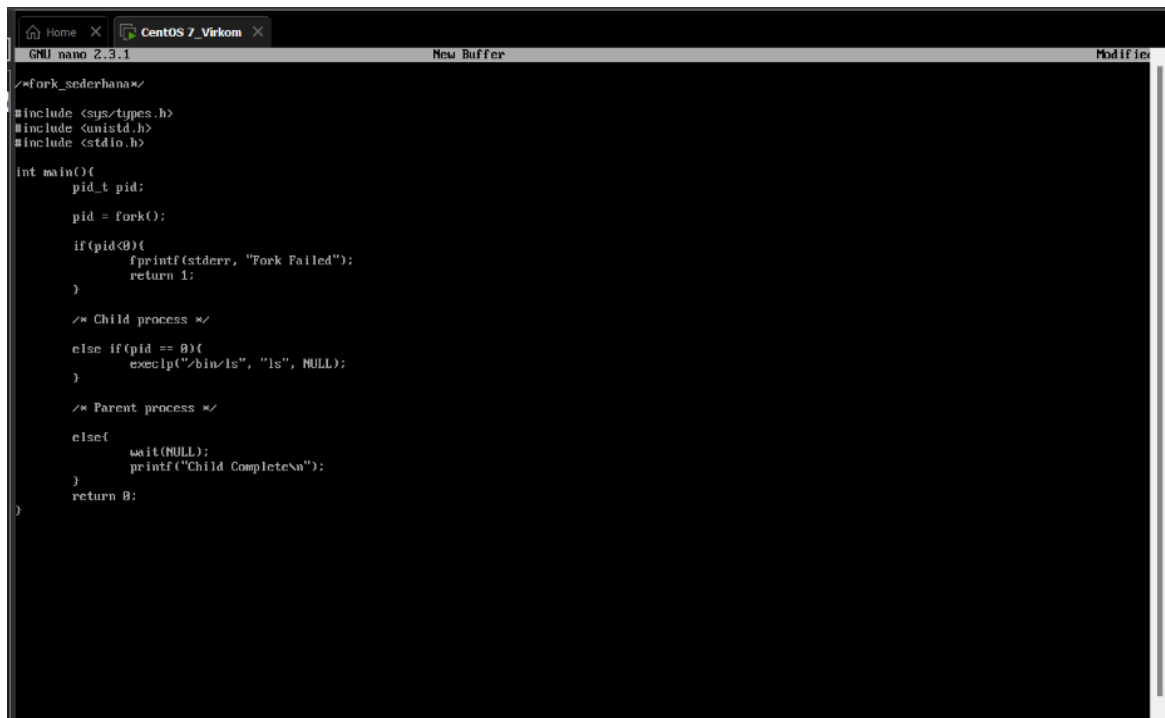
Jawaban :

- a. **Program** Adalah suatu rancangan struktur yang disusun sesuai alur Algoritma dengan tujuan mempermudah suatu permasalahan.
- b. **Proses** adalah rangkaian tindakan, perbuatan, atau pengolahan yang mengubah masukan menjadi keluaran.
- c. **Zombie Process** adalah proses yang eksekusinya selesai tetapi masih memiliki entri di tabel proses. Proses zombie biasanya terjadi untuk proses anak, karena proses induk masih perlu membaca status keluar anaknya.
- d. **Orphan Process** adalah sebuah proses yang ada dalam komputer dimana proses induk telah selesai atau berhenti bekerja namun proses anak sendiri tetap berjalan.
- e. **Process Control Block** merupakan berbagai informasi lain yang diperlukan sistem operasi untuk mengontrol dan berkoordinasi dengan berbagai proses yang aktif

2. Tulislah program berikut

```
1 /*fork_sederhana*/
2
3 #include <sys/types.h>
4 #include <unistd.h>
5 #include <stdio.h>
6
7 int main (){
8     pid_t pid;
9
10    pid = fork();
11
12    if(pid<0){
13        fprintf(stderr, "Fork Failed");
14        return 1;
15    }else if(pid == 0){
16        execlp("/bin/ls", "ls", NULL);
17    }else{
18        wait(NULL);
19        printf("Child Complete\n");
20    }
21
22    return 0;
```

Jawaban :

A screenshot of a terminal window titled 'CentOS 7_Virkom'. The terminal shows a C program being edited in nano. The program is a simple forking example. It includes headers for `<sys/types.h>`, `<unistd.h>`, and `<stdio.h>`. The `main` function declares a `pid_t` variable `pid`. It calls `fork()` to create a child process. If `fork()` returns 0, it's the child process, which runs `execlp("/bin/ls", "ls", NULL)`. If `fork()` returns a non-zero value, it's the parent process, which calls `wait(NULL)` to wait for the child to finish, then prints "Child Complete\n" and returns 0. If `fork()` returns -1, it prints an error message and returns 1. The program is saved and the terminal shows the execution output: "Child Complete\n".

```
GNU nano 2.3.1                                New Buffer                                Modified
~/fork_sederhana~/
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(){
    pid_t pid;

    pid = fork();

    if(pid<0){
        fprintf(stderr, "Fork Failed");
        return 1;
    }

    /* Child process */
    else if(pid == 0){
        execlp("/bin/ls", "ls", NULL);
    }

    /* Parent process */
    else{
        wait(NULL);
        printf("Child Complete\n");
    }
    return 0;
}
```

3. Dari kode program di atas jelaskan apa yang dimaksud dengan :

- a. `pid_t`
- b. `fork()`
- c. `execlp()`
- d. `wait()`

Jawaban :

- a. **pid_t** adalah tipe data yang digunakan pada Linux yang digunakan untuk mempresentasikan id dari process, dan dapat digunakan dengan menginclude header.
- b. **fork()** merupakan suatu system call untuk membuat sebuah proses baru yang berjalan bersamaan dengan proses yang memanggil parent process.
- c. **execlp** digunakan untuk menjalankan command pada terminal dengan command yang akan dijalankan terdapat pada parameter yang dikirimkan melalui function.
- d. **wait()** digunakan untuk menunda proses pemanggilan parent process hingga satu dari children processnya keluar sebuah pertanda diterima, kemudian setelah children process berhenti, parent process melanjutkan pengeksesusiannya.

4. Beri tanda bagian program yang merupakan proses induk (*parent process*) dan proses anak (*child process*).

Jawaban :

```

/*fork_sederhana*/

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main(){
    pid_t pid;

    pid = fork();

    if(pid<0){
        fprintf(stderr, "Fork Failed");
        return 1;
    }

    /* Child process */

    else if(pid == 0){
        execlp("/bin/ls", "ls", NULL);
    }

    /* Parent process */

    else{
        wait(NULL);
        printf("Child Complete\n");
    }
    return 0;
}

```

B. Pemrograman

1. Tuliskan kode program berikut

```

1 /*cetakpid.c*/
2
3 #include <unistd.h>
4 #include <stdio.h>
5
6 int main (){
7
8     printf("The process ID  %d\n", (int) getpid());
9     printf("ID parent process ID is %d\n", (int) getppid());
10    return 0;
11
12 }

```

Eksekusi program di atas dengan cara sebagai berikut:

```
eka@eka-VirtualBox:~/Documents/Prak_OS_Process$ gcc -o cetakpid cetakpid.c
```

- a. Jelaskan perbedaan antara getpid() dengan getppid().
- b. Jelaskan mengapa setiap kali program di atas dieksekusi, maka akan menampilkan process ID yang berbeda. Jelaskan mengapa?

Jawaban :

```
int main (){
    printf("The Process ID");
    printf("ID parent process ID is %d\n", (int) getppid());
    return 0;
}
```

[Wrote 8 lines]

```
[root@localhost ~]# gcc cetakpid.c -o out2
[root@localhost ~]# out2
-bash: out2: perintah tidak ditemukan
[root@localhost ~]# ./out2
The Process IDID parent process ID is 1292
[root@localhost ~]#
```

- a. getpid() adalah function yang mengembalikan ID proses yang sedang berjalan, sedangkan getppid() merupakan function yang mengembalikan ID proses dari parent process yang melakukan pemanggilan proses atau yang sedang berjalan prosesnya.
 - b. Setiap kali program diatas dieksekusi akan menampilkan process ID yang berbeda karena ketika dijalankan secara terus menerus dan berturut-turut process ID akan bertambah satu, yang dapat diartikan bahwa setiap proses baru (masing masing instans baru dengan nama program yang sama) memiliki sebuah process ID yang baru. Ketika process ID tersebut mencapai jumlah maksimumnya maka process ID akan dimulai dari angka satu.
2. Pada kode program berikut, proses baru akan dibentuk dengan menggunakan fungsi **system()**.

```
1 /*system.c*/
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 int main (){
6
7     printf("Running ps with system \n");
8     system("ps -ax | more");
9     printf("Done\n");
10
11     return 0;
12
13 }
```

Eksekusilah program di atas kemudian *capture* hasilnya. Tunjukkanlah proses mana yang menjalankan proses **ps -ax | more** dengan menandai ID proses induk-nya.

Jawaban :

[root@localhost ~]# gcc system.c -o system

[root@localhost ~]# ./system

PID	TTY	STAT	TIME	COMMAND
1	?	Ss	0:01	/usr/lib/systemd/systemd --switched-root --system --deserialize 21
2	?	S	0:00	[kthreadd]
3	?	S	0:00	[ksoftirqd/0]
5	?	S<	0:00	[kworker/0:0H]
7	?	S	0:00	[migration/0]
8	?	S	0:00	[rcu_bh]
9	?	R	0:00	[rcu_sched]
10	?	S	0:00	[watchdog/0]
12	?	S	0:00	[kdevtmpfs]
13	?	S<	0:00	[netns]
14	?	S	0:00	[khungtaskd]
15	?	S<	0:00	[writeback]
16	?	S<	0:00	[kintegrityd]
17	?	S<	0:00	[bioset]
18	?	S<	0:00	[kblockd]
19	?	S<	0:00	[md]
25	?	S	0:00	[kswapd0]
26	?	SN	0:00	[ksmd]
27	?	SN	0:00	[khugepaged]
28	?	S<	0:00	[crypto]
36	?	S<	0:00	[kthrotld]
38	?	S<	0:00	[kmpath_rdacd]
39	?	S<	0:00	[kpsmoused]
40	?	S<	0:00	[ip6_addrconf]
60	?	S<	0:00	[deferwq]
93	?	S	0:00	[kauditd]
277	?	S<	0:00	[ata_sff]
280	?	S	0:00	[scsi_eh_0]
282	?	S<	0:00	[scsi_tmf_0]
283	?	S	0:00	[scsi_eh_1]
284	?	S<	0:00	[scsi_tmf_1]
285	?	S	0:00	[scsi_eh_2]
287	?	S<	0:00	[scsi_tmf_2]
288	?	S	0:00	[kworker/u2:3]
289	?	S	0:00	[scsi_eh_3]

Dapat dilihat bahwa kolom bagian paling kiri berisi informasi PID (Process ID). Untuk proses induk yang menjalankan proses **ps -ax | more** sebagai berikut. Bisa kita lihat dimana parent process adalah process dengan PID 2376, yang ditandai dengan **./system** yang merupakan perintah yang sebelumnya digunakan untuk mencompile program.

565	?	S	0:00	[md0_raid1]
584	?	S<	0:00	[xfs-buf/sdail]
585	?	S<	0:00	[xfs-data/sdail]
587	?	S<	0:00	[xfs-cow/sdail]
590	?	S<	0:00	[xfs-cil/sdail]
591	?	S<	0:00	[xfs-reclaim/sdail]
594	?	S<	0:00	[xfs-log/sdail]
596	?	S<	0:00	[xfs-eofblocks/s]
599	?	S	0:00	[xfsaild/sdail]
645	?	S<sl	0:00	/sbin/auditd
669	?	Ssl	0:00	/usr/lib/polkit-1/polkitd --no-debug
670	?	Ssl	0:00	/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation
675	?	Ss	0:00	/usr/lib/systemd/systemd-logind
678	?	Ssl	0:00	/usr/sbin/rsyslogd -n
695	?	Ssl	0:00	/usr/bin/python -Es /usr/sbin/firewalld --nofork --nopid
698	?	Ssl	0:00	/usr/sbin/NetworkManager --no-daemon
819	?	S	0:00	/sbin/dhclient -d -q -sf /usr/libexec/nm-dhcp-helper -pf /var/run/dhclient-enp0s3.pid -lf /var/lib/NetworkManager/dhclient-enp0s3.conf enp0s3
1001	?	Ssl	0:00	/usr/bin/python -Es /usr/sbin/tuned -l -P
1002	?	Ss	0:00	/usr/sbin/sshd -D
1222	?	Ss	0:00	/usr/libexec/postfix/master -u
1223	?	S	0:00	pickup -l -t unix -u
1224	?	S	0:00	qmgr -l -t unix -u
1283	?	Ss	0:00	login -- root
1292	ttty1	Ss	0:00	-bash
1367	?	Ss	0:00	/usr/sbin/crond -n
1422	?	Ss	0:00	/usr/sbin/anacron -s
1445	?	R	0:00	[kworker/0:2]
8493	?	S	0:00	[kworker/0:0]
8521	?	S	0:00	[kworker/0:1]
8524	ttty1	S+	0:00	./out3
8525	ttty1	S+	0:00	sh -c ps -ax more
8526	ttty1	R+	0:00	ps -ax
8527	ttty1	R+	0:00	sh -c ps -ax more

Done[root@localhost ~]#

3. Pada kode program berikut, proses baru akan dibentuk dengan menggunakan fungsi `exec()`

```
1 /*execlp.c*/
2 #include <unistd.h>
3 #include <stdlib.h>
4 #include <stdio.h>
5
6 int main (){
7
8     printf("Running ps with execlp \n");
9     execlp("ps","ps","-axl",NULL);
10    printf("Done\n");
11
12    exit (0);
```

Jalankan kode program pada nomor 3. Amati hasilnya dan bandingkan hasilnya dengan program pada nomor 2. Temukan perbedaannya dan jelaskan mengapa?

Jawaban :

```
[root@localhost ~]# gcc execlp.c -o execlp
```

```
[root@localhost ~]# ./execlp
```

Dari hasil compile process ID yang menampilkan seluruh daftar ditandai dengan "ps-ax" adalah PID 2427.

```
4 0 487 1 20 0 36828 2492 ep_pol Ss ? 0:00 /usr/lib/systemd/systemd-journald
4 0 500 1 20 0 46904 4876 ep_pol Ss ? 0:00 /usr/lib/systemd/systemd-udevd
4 0 508 1 20 0 129552 4132 poll_s Ss ? 0:00 /usr/sbin/lvmetad -f
1 0 562 2 0 -20 0 0 rescue S< ? 0:00 [bioset]
1 0 563 2 0 -20 0 0 rescue S< ? 0:00 [bioset]
1 0 564 2 20 0 0 0 md_thr S ? 0:00 [md1_raid1]
1 0 565 2 20 0 0 0 md_thr S ? 0:00 [md0_raid1]
1 0 584 2 0 -20 0 0 rescue S< ? 0:00 [xfs-buf/sdall]
1 0 585 2 0 -20 0 0 rescue S< ? 0:00 [xfs-data/sdall]
1 0 587 2 0 -20 0 0 rescue S< ? 0:00 [xfs-conv/sdall]
1 0 590 2 0 -20 0 0 rescue S< ? 0:00 [xfs-cil/sdall]
1 0 591 2 0 -20 0 0 rescue S< ? 0:00 [xfs-reclaim/sdall]
1 0 594 2 0 -20 0 0 rescue S< ? 0:00 [xfs-log/sdall]
1 0 596 2 0 -20 0 0 rescue S< ? 0:00 [xfs-eofblocks/sl]
1 0 599 2 20 0 0 0 xfsail S ? 0:00 [xfsaild/sdall]
5 0 645 1 16 -4 55452 892 ep_pol S<sl ? 0:00 /sbin/auditd
4 999 669 1 20 0 534888 12824 poll_s Ssl ? 0:00 /usr/lib/polkit-1/polkitd --no-
4 81 670 1 20 0 32776 1864 ep_pol Ssl ? 0:00 /bin/dbus-daemon --system --add
4 0 675 1 20 0 24252 1724 ep_pol Ss ? 0:00 /usr/lib/systemd/systemd-logind
4 0 678 1 20 0 212128 4152 poll_s Ssl ? 0:00 /usr/sbin/rsyslogd -n
4 0 695 1 20 0 336332 28792 poll_s Ssl ? 0:00 /usr/bin/python -Es /usr/sbin/f
4 0 698 1 20 0 763092 11340 poll_s Ssl ? 0:00 /usr/sbin/NetworkManager --no-d
4 0 819 698 20 0 113372 15908 poll_s S ? 0:00 /sbin/dhclient -d -q -sf /usr/l
4 0 1001 1 20 0 562388 18600 poll_s Ssl ? 0:00 /usr/bin/python -Es /usr/sbin/t
4 0 1002 1 20 0 105996 4076 poll_s Ss ? 0:00 /usr/sbin/sshd -D
5 0 1222 1 20 0 89544 2064 ep_pol Ss ? 0:00 /usr/libexec/postfix/master -w
4 89 1223 1222 20 0 89648 3988 ep_pol S ? 0:00 pickup -l -t unix -u
4 89 1224 1222 20 0 89716 4016 ep_pol S ? 0:00 qmgr -l -t unix -u
4 0 1283 1 20 0 94396 2460 do_wai Ss ? 0:00 login -- root
4 0 1292 1283 20 0 115524 2172 do_wai Ss tty1 0:00 -bash
4 0 1367 1 20 0 126280 1676 hrtimr Ss ? 0:00 /usr/sbin/crond -n
1 0 1422 1 20 0 125484 1132 sigsus Ss ? 0:00 /usr/sbin/anacron -s
1 0 1445 2 20 0 0 0 - R ? 0:01 [kworker/0:2]
1 0 8521 2 20 0 0 0 worker S ? 0:00 [kworker/0:1]
1 0 8530 2 20 0 0 0 worker S ? 0:00 [kworker/0:0]
0 0 8539 1292 20 0 149092 1476 - R+ tty1 0:00 ps -axl
[root@localhost ~]#
```

Perbedaannya yaitu pada file `system.c` kita dapat melihat command `./system` yang dijalankan beserta dengan argument pada `system` yang kita berikan children process dari `./system` tersebut dimana kita dapat melihat proses dengan command `sh -c ps -ax | more`, kemudian `ps -ax`, dan `more` memiliki process IDnya tersendiri. Sedangkan pada file `execlp.c` kita hanya

melihat satu proses saja yang menandakan munculnya daftar proses ini. Proses tersebut adalah PID 2427 ditandai dengan command ps -ax yang menampilkan keseluruhan daftar secara langsung.

4. Tuliskan kode program berikut.

```
1 /*fork_3.c*/
2
3 #include <stdio.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6
7 int main (int arg, char *argv[]){
8     int pid;
9
10    printf("Main Process ID (PID) = %d Parent Process ID (PPID) = %d\n", getpid(), getppid());
11
12    pid = fork();
13
14    if(pid == 0){
15        printf("This is the child process\n");
16        printf("PID = %d\n", pid);
17        printf("Child's PID %d parent PID %d\n", getpid(), getppid());
18    }else{
19        printf("This is the parent process\n");
20        printf("PID = %d\n", pid);
21        printf("Parent's PID %d parent PID %d\n", getpid(), getppid());
22    }
23
24    return 0;
25 }
```

Eksekusi kode program pada nomor 4, amati hasilnya, kemudian jelaskan hasil dari program tersebut. Proses manakah yang dijalankan pertama kali, apakah proses induk atau proses anak? Mengapa?

Jawaban :

```
Main Process ID (PID) = 16046 Parent Process ID (PPID) = 16043
This is the child process
PID = 16047
Child's PID 16046 parent PID 16043
Main Process ID (PID) = 16046 Parent Process ID (PPID) = 16043
This is the child process
PID = 0
Child's PID 16047 parent PID 16046
```

Proses yang dijalankan pertama kali dari hasil compiler yang dieksekusi oleh kode program adalah parent process, karena child process akan menduplikasi ruang memori dari parent process.

5. Tuliskan kode program berikut


```

1 #include <sys/types.h>
2 #include <unistd.h>
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main (){
7     pid_t pid;
8     char *message;
9     int n;
10
11     printf("Fork program starting\n");
12     pid = fork();
13
14     switch(pid){
15         case -1:
16             perror("Fork failed");
17             exit(0);
18
19         case 0:
20             message = "This is the child";
21             n = 5;
22             break;
23
24         default:
25             message = "This is the parent";
26             n = 3;
27             break;
28     }
29
30     for(; n>0; n--){
31         puts(message);
32         sleep(1);
33     }
34
35     exit (0);

```

Program di atas akan menjalankan dua proses yaitu proses induk dan proses anak. Proses anak akan dijalankan sebanyak 5 kali, dan proses induk akan dijalankan sebanyak 3 kali. Jalankan program di atas dan amati hasil yang terjadi?

Jawaban :

```

[root@localhost ~]# gcc switch.c -o switch
[root@localhost ~]# ./switch
Fork program starting
This is the parent
This is the child
This is the parent
This is the child
This is the parent
This is the child
This is the child
[root@localhost ~]# This is the child
[root@localhost ~]# _

```

Ketika dijalankan terlihat parent process ditampilkan ke layar sebanyak 3 kali sedangkan child process sebanyak 5 kali. Namun untuk child process ketika ditampilkan untuk ke 5 kali, tulisan "This is the child" muncul pada bagian untuk user mengetik command pada terminal. Parent process yang terlebih dahulu muncul dan diikuti dengan child process. Terdapat loop/perulangan yang dibuat untuk melakukan decremental untuk variable n yang di assign ke masing masing parent process dan child process.

6. Sesuai dengan siklus pembentukan proses anak dengan menggunakan fork() hingga proses anak diterminasi seperti yang ditunjukkan pada slide presentasi halaman 3.21, maka proses induk harus menunggu seluruh proses anak selesai dengan memanggil system call wait(). Dengan memodifikasi program nomor 5, tuliskan program di bawah (halaman selanjutnya). Pada program tersebut anda akan menerapkan system call wait() dengan menggunakan library sys/wait.h.

```
1 #include <sys/types.h>
2 #include <sys/wait.h>
3 #include <unistd.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 int main (){
8     pid_t pid;
9     char *message;
10    int n;
11    int exit_code;
12
13    printf("Fork program starting\n");
14    pid = fork();
15
16    switch(pid){
17        case -1:
18            perror("Fork failed");
19            exit(0);
20
21        case 0:
22            message = "This is the child";
23            n = 5;
24            exit_code = 37;
25            break;
26
27        default:
28            message = "This is the parent";
29            n = 3;
30            exit_code = 0;
31            break;
32    }
33
34    for(; n>0; n--){
35        puts(message);
36        sleep(1);
37    }
38
39    if(pid!=0){
40        int stat_val;
41        pid_t child_pid;
42
43        child_pid = wait(&stat_val);
44
45        printf("Child has finished with PID = %d\n", child_pid);
46        if(WIFEXITED(stat_val)){
47            printf("Child exited with code %d\n", WEXITSTATUS(stat_val))
48        }else{
49            printf("Child terminated abnormally\n");
50        }
51    }
52
53    exit(exit_code);
```

Jawaban :

```
[root@localhost ~]# gcc switch.c -o out
[root@localhost ~]# ./out
Fork program starting
This is the parent
This is the child
This is the child
This is the parent
This is the child
This is the parent
This is the child
This is the child
Child has finished with PID = 2328
Child exited with code 37
[root@localhost ~]#
```

7. Jelaskan apa efek dari menggunakan fungsi `wait()` dari program di atas?
Bandingkan hasil dari program pada nomor 5 dengan hasil program nomor 6, apa yang dapat anda simpulkan dari kedua program tersebut?

Jawaban :

Hasil decrement terakhir dari variable `n` untuk mengetikkan command pada terminal. Namun pada nomor 6 function `wait()` yang diberikan membuat parent process untuk menunggu child process selesai dieksekusi untuk kemudian parent process melakukan terminasi sehingga dapat dilihat exit code = 0.

8. Jelaskan mengapa parent process harus memanggil system call **`wait()`** dan apa yang terjadi apabila system call **`wait()`** tidak dipanggil?

Jawaban :

system call `wait()` dipanggil untuk membuat parent process menunggu child process menyelesaikan eksekusi processnya hingga selesai. Setelah child process berhenti atau diterminasi maka giliran parent process berikutnya untuk berhenti beroperasi, jika system call `wait()` tidak dipanggil maka dapat terjadi yang namanya orphan process, parent pocess berhenti tanpa menunggu child process untuk berhenti terlebih dahulu atau dengan kata lain dimana suatu child process masih sedang berjalan atau dieksekusi namun tidak lagi memiliki suatu parent process yang manaunginya.

9. Tuliskan kode program berikut.

```

1 /*zombie.c*/
2
3 #include <stdlib.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6
7 int main (int arg, char *argv[]){
8     pid_t pid;
9
10    pid = fork();
11
12    if (pid > 0){
13        sleep(60);
14    }else{
15        exit (0);
16    }
17
18    return 0;
19 }

```

Jawaban :

```

#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

int main (int arg, char *argv[]){

    pid_t pid;

    pid = fork();

    if(pid>0){
        sleep(60);
    }else{
        exit (0);
    }

    return 0;
}

```

10. Eksekusi program di atas. Observasi hasil dari eksekusi program, Anda dapat menggunakan perintah **ps -al** pada terminal yang lain untuk melihat proses yang sedang berjalan. Apakah hasil dari perintah **ps -al**, jelaskan mengapa terjadi hal demikian?

Jawaban :

```

[root@localhost ~]# gcc zombie.c -o zombie
[root@localhost ~]# ./zombie
[root@localhost ~]# ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R    0    2357   2271  0  80   0 - 34377 -          tty1        00:00:00 ps
[root@localhost ~]# ./zombie

ps -al
^C
[root@localhost ~]# ps -al
F S  UID    PID  PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 R    0    2360   2271  0  80   0 - 34377 -          tty1        00:00:00 ps
[root@localhost ~]#

```

Melalui hasil compile terlihat bahwa yang dieksekusi pada terminal lainnya yang menjalankan zombie mengalami hal yang disebut zombie process sebagaimana dimaksudkan pada kode di soal no 9 dengan suatu pernyataan `if > pid = 0` maka parent process akan melakukan sleep selama 60 s walaupun sebenarnya pengeksekusian dari parent process sudah selesai dilakukan dan process masih terlihat ketika dipanggil `ps -al`.

C. Tugas Pemrograman

1. Buatlah sebuah program untuk mengurutkan data menggunakan algoritma Selection Sort. Namakan program sebagai `child_process_ssort.c`. Kompilasi program untuk menghasilkan berkas tereksekusi (executable file): `child_process_ssort.exe`. Proses ini akan anda gunakan sebagai proses anak.

Jawaban :

```
#include <stdio.h>

void swap(int *min, int *great){
    int temp;
    temp = *min;
    *min = *great;
    *great = temp;
}

void selectionSort(int myArr[], int n){
    int i, j, min_index;
    for(i = 0; i < n-1; i++){
        min_index = i;
        for(j = i+1; j < n; j++){
            if(myArr[j] < myArr[min_index]){
                min_index = j;
            }
        }
        swap(&myArr[i], &myArr[min_index]);
    }
}

int main(){
    int i;
    int myArr[] = {61, 13, 51, 67, 37, 12};
    int n = sizeof(myArr)/sizeof(myArr[0]);

    selectionSort(myArr,n);

    for(i = 0; i<n; i++){
        printf("%d\n", myArr[i]);
    }
    return 0;
}
```

```
[root@localhost ~]# gcc child_process_ssort.c -o out
[root@localhost ~]# ./out
12
13
37
51
51
67
[root@localhost ~]#
```

2. Buatlah proses induk yang bertujuan untuk membuat dan mengeksekusi proses anak pada no. 1 di atas. Beri nama program anda sebagai `parent_process.c`. Kompilasi dan jalankan program.

Jawaban :

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

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

    if(pid<0){
        fprintf(stderr, "Fork failed\n");
        return 1;
    }else if(pid == 0){
        printf("Child is being executed\n");
        execlp("./child_process_ssort", "child_process_ssort", NULL);
    }else{
        wait(NULL);
        printf("Child execution complete\n");
    }
    printf("End of parent program\n");
    return 0;
}

```

```

[root@localhost ~]# nano parent_process.c
[root@localhost ~]# gcc parent_process.c -o out
[root@localhost ~]# out
-bash: out: command not found
[root@localhost ~]# ./out
Child is being executed
End of parent program
Child execution complete
End of parent program
[root@localhost ~]#

```