

Tugas 12

Sistem Operasi



Nama : Rheza Dewangga Rendragraha

Kelas: 1 D4 Teknik Informatika B

NRP: 2110191044

Tugas Pendahuluan

1. Apa yang dimaksud dengan system call ?
System call secara sederhana adalah jembatan penghubung antara user mode dan kernel mode, system call digunakan oleh pengguna ketika pengguna akan melakukan suatu proses yang membutuhkan resources yang mana resources hanya bisa dipanggil dari mode kernel.
2. Apa yang dimaksud dengan system call fork(), execl(), dan wait(). Jawablah dengan menggunakan perintah man (contoh : man 2 fork, man 2 execl, dan man 2 wait)?
 - a. Fork : Digunakan untuk membuat proses baru dengan menduplikasi pemanggilan proses. Proses baru adalah child process. Dan proses pemanggilan adalah parent process
 - b. Execl : Digunakan untuk menggantikan proses yang sedang berjalan saat ini dengan proses baru.
 - c. Wait : Digunakan untuk pergantian keadaan dalam child proses pemanggilan, dan memperoleh informasi tentang child yang keadaannya telah berubah. Perubahan keadaan dianggap sebagai child yang diakhiri.
3. Apa yang dimaksud sistem virtual memory, proses swapping dan buffer cache pada manajemen memory ?
 - a. Sistem virtual memory : Virtual memory adalah skema alokasi penyimpanan yang mana memori sekunder dapat dialamatkan seperti itu adalah bagian dari main memory. Pengalamatan program yang digunakan untuk menyesuaikan memory dibedakan dari pengalamatan memory system yang digunakan untuk mengidentifikasi tempat penyimpanan fisik dan alamat program yang dibuat diterjemahkan secara otomatis ke alamat mesin yang sesuai.
 - b. Swap Process : Mekanisme di mana proses dapat ditukar sementara di luar memory utama ke penyimpanan sekunder (disk) dan membuat memory tersebut tersedia untuk proses lain.
 - c. Buffer Cache : Digunakan untuk menyimpan data ketika data tersebut sedang dipindahkan antara dua perangkat atau antara perangkat dan aplikasi.
4. Apa yang dimaksud perintah free dan cat /proc/meminfo ?
 - a. Free : Perintah free digunakan untuk menampilkan jumlah total dari memori fisik dan memori swap yang tersedia dan yang telah digunakan, begitu juga dengan buffer dan cache yang digunakan oleh kernel. Informasi tersebut dikumpulkan dengan menguraikan /proc/meminfo.
 - b. Cat /proc/meminfo : Pada dasarnya sama saja dengan menampilkan perintah sekumpulan free semenjak free dikumpulkan pada /proc/meminfo.\
5. Apa yang dimaksud perintah ps
Digunakan untuk menampilkan informasi dari proses yang aktif.

Percobaan 1

```
1 #include <iostream>
2 using namespace std;
3 #include <sys/types.h>
4 #include <unistd.h>
5 /* getpid() adalah system call yg dideklarasikan pada
   unistd.h.
6 Menghasilkan suatu nilai dengan type pid_t.
7 pid_t adalah type khusus untuk process id yg ekuivalen dg
   int
8 */
9 int main(void) {
10 pid_t mypid;
11 uid_t myuid;
12 for (int i = 0; i < 3; i++) {
13     mypid = getpid();
14     cout << "I am process " << mypid << endl;
15     cout << "My parent is process " << getppid() << endl;
16     cout << "The owner of this process has uid " << getuid()
17     << endl;
18     /* sleep adalah system call atau fungsi library
19     yang menghentikan proses ini dalam detik
20     */
21     sleep(1);
22 }
23 return 0;
24 }
```

Line 1 : #include adalah sebuah prosesor pengarah yang mengarah ke compiler untuk meletakkan kode dari header file iostream.h ke dalam program

Line 2 : Dalam programming, kita tidak bisa memiliki variable atau fungsi yang mempunyai dengan nama yang sama. Jadi untuk menghindari konflik tersebut, kita menggunakan namespace. Kemudian std singkatan dari standard, yang berarti kita menggunakan segala sesuatu dengan standard namespace.

Line 3 : Memanggil library yang menyediakan tipe data pid_t

Line 4 : Sebuah perintah untuk memanggil library unistd.h, yaitu library yang menyediakan akses ke API system operasi standar POSIX

Line 5 : getpid() memberi return PID dari proses calling. Kemudian unistd.h menentukan bermacam – macam konstanta dan tipe, dan mendeklarasikan bermacam fungsi.

Line 6 : Tipe pid_t adalah singkatan dari *process identification* dan digunakan untuk merepresentasikan process id. Kapanpun kita ingin mendeklarasikan variable akan ditangani oleh process id.

Line 7 : Sebuah comment yang berfungsi untuk memberikan deskripsi program pada dokumentasi script.

Line 8 : Menutup comment

Line 9 : Pendeklarasian sekaligus membuka statement fungsi main dimana fungsi ini tempat utama dieksekusinya program.

Line 10 : Mendeklarasi variabel mypid dengan tipe data pid_t yang valuenya ekuivalen dengan integer.

Line 11 : Mendeklarasi variabel myuid bertipedata uid_T yang valuenya akan ekuivalen dengan integer

Line 12 : sistem akan terus melaksanakan perintah yang terdapat dibawah baris 12 hingga looping memenuhi i kurang dari 3.

Line 13 : sesuai namanya, proses ini mengambil ID dari proses mypid=getpid().Get id ialah get user identity yang akan mengembalikan user id yang sebenarnya dari proses pemanggilan.

Line 14 : cout digunakan untuk menampilkan karakter yang ingin ditampilkan, cout memiliki arti character out. Karena dikombinasikan dengan sebuah variabel mypid, itulah alasan mengapa penulisannya menjadi cout<<"i am a process"<<mypid<<endl;.

Line 15 : sama seperti sebelumnya, cout pada baris ke 15 dikombinasikan dengan getpid(), dan kata-kata yang ingin ditampilkan ialah "my parent is process". Get id ialah get user identity yang akan mengembalikan user id yang sebenarnya dari proses pemanggilan

Line 16 : cout berarti melakukan pencetakan string "The owner of this process has uid " dan getuid() adalah *get user identity* yang akan mengembalikan user ID yang sebenarnya dari proses pemanggilan.

Line 17 : Menampilkan baris baru dan membuat stream

Line 18 : Untuk membuka statement komen pada teks string

Line 19 : berfungsi sebagai pembuka komen pada teks string. Pada baris 19 ialah komen itu sendiri.

Line 20 : Statement penutup untuk komen

Line 21 : Sleep digunakan untuk menunda dalam waktu tertentu. Secara default satuan dari waktunya adalah second, namun kita bisa menambahkan suffix m untuk minute, dan h untuk hours.

Line 22 : Menutup process looping for

Line 23 : Mengembalikan nilai 0

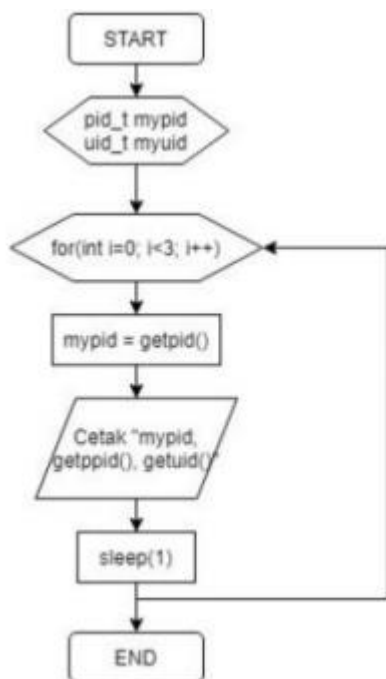
Line 24 : Penutup statement fungsi main

```

rheza@rheza-VirtualBox:~$ g++ -o fork01 fork01.cpp
rheza@rheza-VirtualBox:~$ ./fork01
I am process6424
My parent is process6409
The owner of this process has uid1000
I am process6424
My parent is process6409
The owner of this process has uid1000
I am process6424
My parent is process6409
The owner of this process has uid1000

```

Pada percobaan fork01, pada dasarnya memanggil pid dari child process dengan fungsi getpid() kemudian memanggil parent process dengan fungsi getppid(), kemudian memanggil uid / user id dengan menggunakan perintah getuid().



Percobaan 2

```
1. #include <iostream>
2. using namespace std;
3. #include <sys/types.h>
4. #include <unistd.h>
5.
6. /* getpid() dan fork() adalah system call yg dideklarasikan
7. pada unistd.h. menghasilkan suatu nilai dengan tipe pid_t.
8. pid_t adalah tipe khusus untuk process id yg ekuivalen dg int
9. */
10.
11. int main(void)
12. {
13.     pid_t childpid;
14.     int x = 5;
15.     childpid = fork();
16.
17.     while (1)
18.     {
19.         cout << "This is process " << getpid() << endl;
20.         cout << "x is " << x << endl;
21.         sleep(1);
22.         x++;
23.     }
24.     return 0;
25. }
```

Line 1 : input output operasi yang digunakan oleh bahasa C++, fungsi keluaran pada C++ yang menampilkan data dengan tipe data apapun ke layar.

Line 2 : memanggil **namespace** yang memiliki nama 'std'. **Namespace 'std'** merupakan standar **namespace** dari C++ yang dapat digunakan untuk memanggil class/object/fungsi yang terdapat di dalam **namespace**

Line 3 : program mengimport file header bernama *sys/types.h* yang berisi nama nama variable yang ada pada system linux. Misalnya uid_t, pid_t dan lain lain

Line 4 : program mengimport file header bernama *unistd.h* yang mendefinisikan konstanta simbolik dan jenis lainnya, dan mendeklarasikan fungsi lain-lain.

Line 11 : Fungsi main () adalah fungsi utama dalam sebuah program. Fungsi ini yang akan dieksekusi pertama kali.

Line 13 : pid_t childpid; childpid merupakan variabel dengan tipe data khusus untuk process ID yaitu pid_t dimana tipe data ini ekuivalen dengan int.

Line 14 : Terdapat variabel x bernilai 5 dengan tipe data int.

Line 15 : Variabel childpid ini berisi return value dari pemanggilan sistem call fork(). Sistem call fork() sendiri digunakan untuk membuat proses baru dengan menduplikasi proses yang sedang dijalankan. Proses hasil duplikasi disebut child process, return value = 0, sedangkan proses yang diduplikasi disebut parent process, return value > 0. Jika return value nya <0, maka proses fork() gagal.

Line 17 : Looping while dengan kondisi 1 artinya looping dilakukan tanpa berhenti atau infinite loop. Looping ini akan menjalankan baris 19 hingga 22.

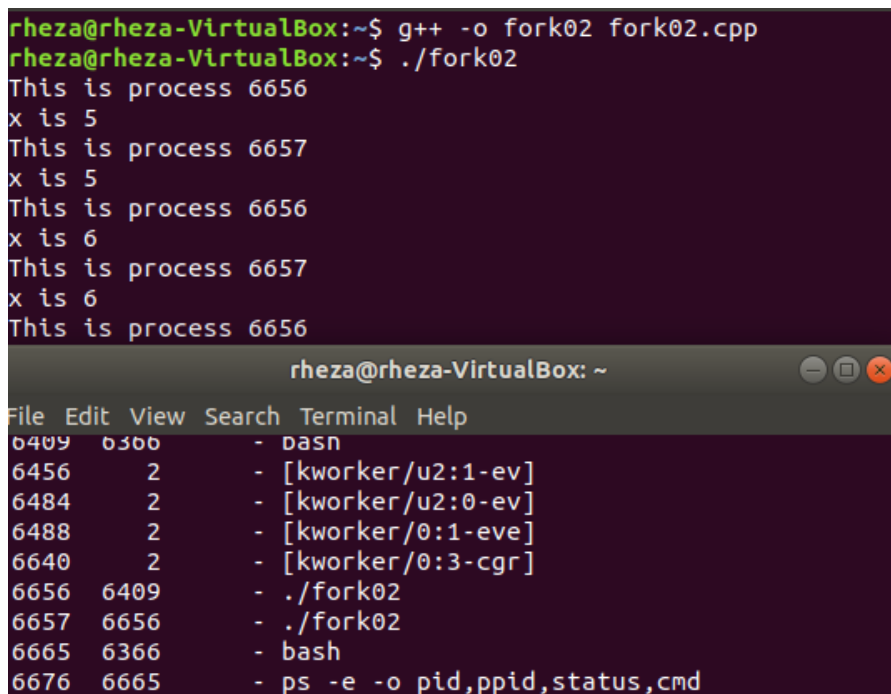
Line 19 : fungsi cout akan mencetak tulisan “This is process “ disertai return value dari pemanggilan sistem call getpid(). Fungsi getpid() adalah untuk mengembalikan/mereturn process id dari proses yang sedang dijalankan.

Line 20 : fungsi cout akan mencetak tulisan “x is “ disertai dengan return value dari variabel x.

Line 21 : Terdapat pemanggilan sistem call sleep() dengan parameter berupa angka dalam satuan second (waktu). Dalam hal ini, parameternya 1 detik, tujuannya adalah untuk menghentikan program secara sementara selama 1 detik.

Line 22 : Variabel x mengalami increment (x++), yaitu penambahan nilai dengan rumus $x=x+1$.

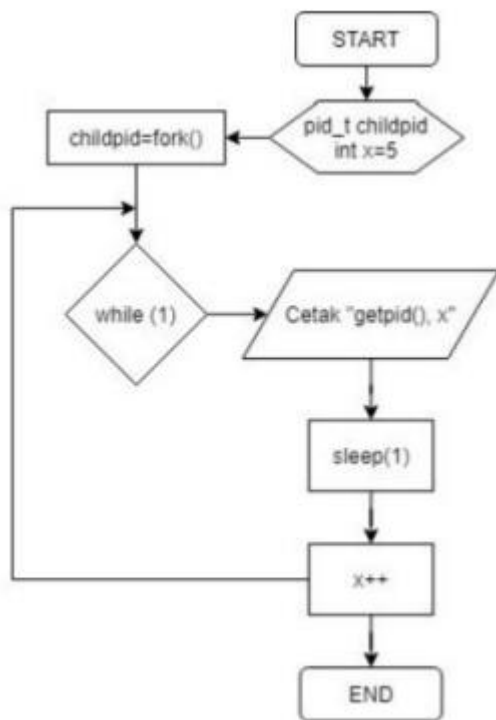
Line 24 : Fungsi main() akan mengembalikan nilai 0 jika program berjalan dengan baik.



```
rheza@rheza-VirtualBox:~$ g++ -o fork02 fork02.cpp
rheza@rheza-VirtualBox:~$ ./fork02
This is process 6656
x is 5
This is process 6657
x is 5
This is process 6656
x is 6
This is process 6657
x is 6
This is process 6656

rheza@rheza-VirtualBox: ~
File Edit View Search Terminal Help
6409 6366 - dash
6456 2 - [kworker/u2:1-ev]
6484 2 - [kworker/u2:0-ev]
6488 2 - [kworker/0:1-eve]
6640 2 - [kworker/0:3-cgr]
6656 6409 - ./fork02
6657 6656 - ./fork02
6665 6366 - bash
6676 6665 - ps -e -o pid,ppid,status,cmd
```

Berdasarkan gambar tersebut dapat disimpulkan bahwa sistem call fork() berhasil membuat 2 buah proses dalam satu kali iterasi dengan parent process nya memiliki PID 6656 dan child process nya memiliki PID 6657. Kemudian, sistem call sleep(1) akan dijalankan untuk mencetak proses dari iterasi kedua. Karen program ini bersifat infinite loop, gunakan ctrl+c untuk menghentikannya.



Percobaan 3

```

1. #include <iostream>
2. using namespace std;
3. #include <sys/types.h>
4. #include <unistd.h>
5.
6. /* getpid() dan fork() adalah system call yg dideklarasikan
7. pada unistd.h. menghasilkan suatu nilai dengan type pid_t.
8. pid_t adalah type khusus untuk process id yg ekuivalen dg int
9. */
10.
11. int main(void)
12. {
13.     pid_t childpid;
14.     childpid = fork();
15.     for(int i=0 ; i<5 ; i++)
16.     {
17.         cout << "This is process " << getpid() << endl;
18.         sleep(2);
19.     }
20.     return 0;
21. }
  
```

Line 1 : #include merupakan Preprocessor Directive yang mana berfungsi sebagai sebuah intruksi untuk memanggil file-header yang akan di gunakan. iostream adalah file header yang merupakan kepanjangan dari Input Output Stream yang digunakan untuk memanggil fungsi yang ada pada library file-header tersebut seperti fungsi input yaitu cin dan fungsi output yaitu cout.

Line 2 : Using namespace std digunakan untuk memanggil namespace yang memiliki nama 'std'. Namespace 'std' merupakan standar namespace dari C++ yang dapat kita gunakan untuk memanggil class/object/fungsi yang terdapat di dalam namespace tersebut.

Line 3 : Program mengimport file header bernama sys/types.h yang berisi beberapa nama variable yang ada pada system Linux. Misalnya uid_t, pid_t dan lain lain

Line 4 : <unistd.h> digunakan untuk mendeklarasikan library untuk beberapa konstanta symbolic yang memiliki fungsi yang berbeda beda

Line 5 : -

Line 6 : Merupakan komentar

Line 7 : Merupakan komentar

Line 8 : Merupakan komentar

Line 9 : Merupakan komentar

Line 10 : -

Line 11 : Fungsi main () adalah fungsi utama dalam sebuah program. Fungsi ini yang akan dieksekusi pertama kali.

Line 12 : -

Line 13 : Merupakan deklarasi variabel childpid dengan tipe data bernilai pid_t. Tipe data ini merupakan hasil dari proses systemcall getpid() dan fork(). Nilai pengembalian dari salah dua proses tersebut sama seperti tipe data integer.

Line 14 : Melakukan assign dari proses system call fork() ke variabel childpid, nilai yang didapat kemudian akan digunakan untuk proses proses selanjutnya

Line 15 : Proses looping yang dilakukan sebanyak lima kali dengan indeks iterasi dimulai dari 0 dan diakhiri kurang dari lima dengan lompatan iterasi sebanyak satu kali tiap looping .

Line 16 : -

Line 17 : Melakukan print "This is process" dengan menambahkan nilai dari proses systemcall getpid().

Line 18 : Menjalankan fungsi sleep, proses ini seperti menunggu rangkaian proses yang terjadi dalam kurun waktu tertentu. Dalam contoh ini waktunya adalah dua second.

Line 19 : -

Line 20 : Return akan mengembalikan nilai 0 jika fungsi main berhasil dijalankan.

Line 21 : -

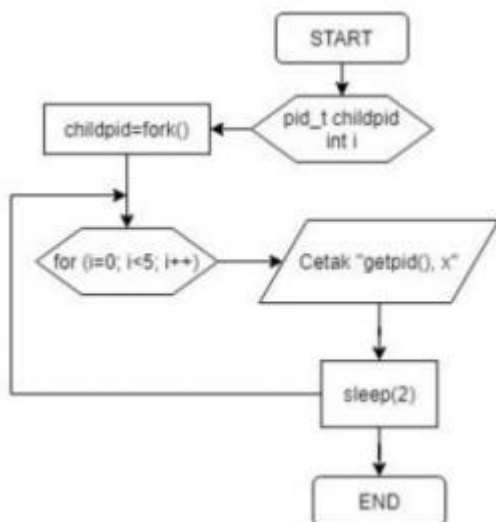
```

rheza@rheza-VirtualBox:~$ g++ -o fork03 fork03.cpp
rheza@rheza-VirtualBox:~$ ./fork03
This is process6947
This is process6948
This is process6947
This is process6948
This is process6947
This is process6948
This is process6947

rheza@rheza-VirtualBox: ~
File Edit View Search Terminal Help
0720 2 - [kworker/u2:1-ev]
6765 2 - [kworker/u2:2-ev]
6779 2 - [kworker/0:0-mm_]
6930 2 - [kworker/u2:0-ev]
6939 2 - [kworker/0:2-cgr]
6940 2 - [kworker/0:3-eva]
6947 6409 - ./fork03
6948 6947 - ./fork03
6950 6665 - ps -e -o pid,ppid,status,cmd

```

Analisis : Bisa dilihat bahwa pada percobaan fork03, kita menciptakan sebuah proses dan kita memanggil pid dengan fungsi getpid() dan dilakukan selama 5 kali. Akan tetapi yang ditampilkan lebih dari 5 kali, hal tersebut dikarenakan yang ditampilkan adalah child process dan parent process.



Percobaan 4

```
1.  #include <iostream>
2.  using namespace std;
3.  #include <sys/types.h>
4.  #include <unistd.h>
5.  #include <sys/wait.h>
6.
7.  /* pid_t fork() dideklarasikan pada unistd.h.
8.  pid_t adalah type khusus untuk process id yg ekuivalen dg int
9.  */
10.
11. int main(void)
12. {
13.     pid_t child_pid;
14.     pid_t wait_result;
15.     int status;
16.
17.     child_pid = fork();
18.     if(child_pid == 0){
19.         /* kode ini hanya dieksekusi proses child */
20.         cout << "I am a child and my pid = " << getpid() << endl;
21.         cout << "My parent is " << getppid() << endl;
22.         /* keluar if akan menghentikan hanya proses child */
23.     }
24.     else if(child_pid > 0){
25.         /* kode ini hanya mengeksekusi proses parent */
26.         cout << "I am the parent and my pid = " << getpid() << endl;
27.         cout << "My child has pid = " << child_pid << endl;
28.     }
29.     else {
30.         cout << "The fork system call failed to create a new process"
31.         << endl;
32.         exit(1);
33.     }
34.     /* kode ini dieksekusi baik oleh proses parent dan child */
35.     cout << "I am a happy, healthy process and my pid = " << getpid() <<
36.     endl;
37.     if(child_pid == 0){
38.         /* kode ini hanya dieksekusi oleh proses child */
39.         cout << "I am a child and I am quitting work now!" << endl;
40.     }
41.     else{
42.         /* kode ini hanya dieksekusi oleh proses parent */
43.         cout << "I am a parent and I am going to wait for my child" <<
44.         endl;
45.         do {
46.             /* parent menunggu sinyal SIGCHLD mengirim tanda bahwa
47.             proses child diterminasi */
48.             wait_result = wait(&status);
49.             } while (wait_result != child_pid);
50.             cout << "I am a parent and I am quitting." << endl;
51.         }
52.     }
53.     return 0;
54. }
```

Line 1 : #include merupakan Preprocessor Directive yang mana berfungsi sebagai sebuah intruksi untuk memanggil file-header yang akan di gunakan. iostream adalah file header yang merupakan kepanjangan dari Input Output Stream yang digunakan untuk memanggil fungsi yang ada pada library file-header tersebut seperti fungsi input yaitu cin dan fungsi output yaitu cout.

Line 2 : Using namespace std digunakan untuk memanggil namespace yang memiliki nama 'std'. Namespace 'std' merupakan standar namespace dari C++ yang dapat kita gunakan untuk memanggil class/object/fungsi yang terdapat di dalam namespace tersebut.

Line 3 : Program mengimport file header bernama sys/types.h yang berisi beberapa nama variable yang ada pada system Linux. Misalnya uid_t, pid_t dan lain lain

Line 4 : <unistd.h> digunakan untuk mendeklarasikan library untuk beberapa konstanta symbolic yang memiliki fungsi yang berbeda beda

Line 5 : digunakan untuk memakai fungsi wait() yang digunakan untuk proses parent.

Line 6 : -

Line 7 : Merupakan komentar

Line 8 : Merupakan komentar

Line 9 : Merupakan komentar

Line 10 : -

Line 11 : Fungsi main () adalah fungsi utama dalam sebuah program. Fungsi ini yang akan dieksekusi pertama kali.

Line 12 : -

Line 13 : Merupakan deklarasi variabel childpid dengan tipe data bernilai pid_t. Tipe data ini merupakan hasil dari proses systemcall getpid() dan fork(). Nilai pengembalian dari salah dua proses tersebut sama seperti tipe data integer.

Line 14 : Melakukan deklarasi variabel wait_result bertipe data pid_t, yang nantinya variabel tersebut akan dipakai untuk menampung return dari fungsi wait().

Line 15 : Melakukan deklarasi variabel status bertipe data integer. Digunakan untuk parameter dari fungsi wait().

Line 16 : -

Line 17 : Melakukan inisialisasi variabel child_pid, kita assign fork() ke variabel tersebut. Supaya nanti variabel tersebut bisa kita jadikan ekspresi kondisional untuk menentukan proses parent dan child. Fungsi fork() tadi akan me-return pid yang nantinya akan ditampung oleh variabel child_pid.

Line 18 : ketika fork() me-return 0 maka jalankan proses child di dalam kondisional if ini. Dan proses childnya adalah pada baris ke 20 dan 21

Line 19 : -

Line 20 : Menjalankan instruksi `getpid()` dengan tujuan untuk menunjukkan pid dari child process tersebut

Line 21 : Menjalankan instruksi `getppid()` dengan tujuan untuk menunjukkan parent's pid dari child process ini.

Line 22 : -

Line 23 : -

Line 24 : ketika `fork()` me-return pid lebih besar dari 0 maka jalankan proses parent di dalam kondisional `else if` ini. Maka proses parent tersebut yaitu ada baris ke 26 dan 27.

Line 25 : -

Line 26 : Menjalankan instruksi `getpid()` dengan tujuan untuk menunjukkan pid dari proses parent ini

Line 27 : Menjalankan instruksi untuk mencetak `child_pid`. Kita tahu bahwa `child_pid` adalah variabel penampung return value dari `fork()`. Maka hasilnya sama saja dengan menunjukkan pid dari child.

Line 28 : -

Line 29 : -

Line 30 : Ketika `child_pid` tidak sama dengan 0 dan tidak lebih besar dari 0 maka proses forking tidak dapat berjalan. Maka dari itu kita buat sebuah pesan error atau failed dalam proses forking tersebut.

Line 31 : kita jalankan instruksi `exit(1)` untuk langsung keluar setelahnya.

Line 32 : -

Line 33 : -

Line 34 : -

Line 35 : kita membuat instruksi `getpid()` lagi. Pada baris ini nantinya bisa dilakukan oleh proses parent maupun child karena tidak dalam kondisional.

Line 36 : -

Line 37 : kita tuliskan kondisional `if` dengan ekspresi `if(child_pid == 0){}` maksudnya adalah ketika `child_pid` sama dengan 0, maka dijalankan instruksi pada baris 39

Line 38 : -

Line 39 : mencetak pesan bahwa child telah selesai.

Line 40 : -

Line 41 : kita menuliskan kondisional `else` yang maksudnya adalah ketika nilai dari `child_pid` selain 0, maka jalankan instruksi pada baris ke 43 sampai 49 dan tentu kondisional ini hanya dijalankan oleh proses parent saja karena pid-nya selain 0.

Line 42 : -

Line 43 : kita membuat instruksi mencetak pesan untuk menunjukkan bahwa parent sedang dalam proses menunggu child supaya terminasi

Line 44 : kita membuat looping do while. Di dalam looping tersebut kita akan membuat proses dimana parent akan menunggu child hingga terminasi

Line 45 : -

Line 46 : memanggil fungsi wait(&status), dengan tujuan menunggu sinyal SIGCHILD yaitu tanda bahwa child diterminasi. Lalu hasilnya kita assign ke variabel yang kita buat yaitu wait_result.

Line 47 : kita buat ekspresi untuk do while nya yaitu while(wait_result != child_pid);

Line 48 : -

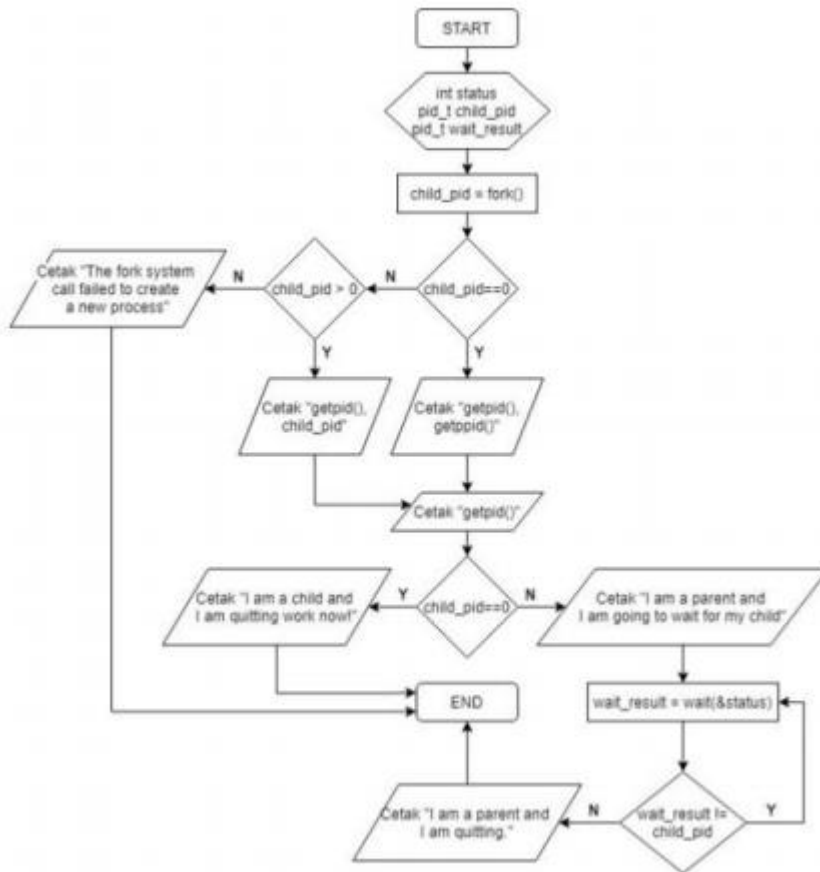
Line 49 : -

Line 50 : menyatakan hasil keluaran dari fungsi main() dan juga bahwa program berakhir dengan normal

Line 51 : -

```
rheza@rheza-VirtualBox:~$ g++ -o fork04 fork04.cpp
rheza@rheza-VirtualBox:~$ ./fork04
I am the parent and my pid = 7295
My child has pid = 7296
I am a happy, healthy process and my pid = 7295
I am a parent and I am going to wait for my child
I am a child and my pid = 7296
My parent is 7295
I am a happy, healthy process and my pid = 7296
I am a child and I am quitting work now!
I am a parent and I am quitting.
```

Analisis : Pada percobaan fork04, langsung ditampilkan sebuah parent process, hal tersebut berarti bahwa pada pemanggilan fungsi fork() yang merupakan pembuatan proses baru, return value nya adalah > 0 sehingga yang dieksekusi adalah parent process. Kemudian karena return lebih dari 0, maka kita mengeksekusi fungsi wait yang berfungsi untuk menunggu child process untuk diakhiri, kemudian kita akan berganti ke child process di mana return pada fungsi fork() adalah 0 dan kode untuk child process dieksekusi, dan kemudian pada kode tersebut terdapat perintah exit yang mana akan mengakhiri process tersebut, dan pada akhirnya setelah diakhiri, maka ditampilkan “I am a parent and I am quitting.” Hal tersebut dikarenakan child process telah diakhiri dan parent process bisa mengakhiri.



Percobaan 5

```

1. #include <iostream>
2. using namespace std;
3. #include <sys/types.h>
4. #include <unistd.h>
5. #include <sys/wait.h>
6.
7. /* pid_t fork() dideklarasikan pada unistd.h.
8. pid_t adalah type khusus untuk process id yg ekuivalen dg int
9. */
10. int main(void) {
11. pid_t child_pid;
12. int status;
13. pid_t wait_result;
14. child_pid = fork();
15. if (child_pid == 0) {
16.     /* kode ini hanya dieksekusi proses child */
17.     cout << "I am a child and my pid = " << getpid() << endl;
18.     execl("/bin/ls", "ls", "-l", "/home", NULL);
19.     /* jika execl berhasil kode ini tidak pernah digunakan */
20.     cout << "Could not execl file /bin/ls" << endl;
21.     exit(1);
22.     /* exit menghentikan hanya proses child */
23. } else if (child_pid > 0) {
24.     /* kode ini hanya mengeksekusi proses parent */
25.     cout << "I am the parent and my pid = " << getpid() << endl;

```

```

26.  cout << "My child has pid = " << child_pid << endl;
27.  } else {
28.  cout << "The fork system call failed to create a new process " << endl;
29.  exit(1);
30.  }
31.  /* kode ini hanya dieksekusi oleh proses parent karena
32.  child mengeksekusi dari "/bin/ls" atau keluar */
33.  cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
34.  if (child_pid == 0) {
35.  /* kode ini tidak pernah dieksekusi */
36.  printf("This code will never be executed!\n");
37.  } else {
38.  /* kode ini hanya dieksekusi oleh proses parent */
39.  cout << "I am a parent and I am going to wait for my
40.  child " << endl;
41.  do {
42.  /* parent menunggu sinyal SIGCHLD mengirim tanda
43.  bila proses child diterminasi */
44.  wait_result = wait( & status);
45.  } while (wait_result != child_pid);
46.  cout << "I am a parent and I am quitting." << endl;
47.  }
48.  return 0;
49. }

```

Line 1 : supaya dapat melakukan input output maka harus menyertakan file library iostream, caranya dengan mengincludekan iostream itu. include adalah sebuah cara agar processor menyertakan file yang menjadi parameter nya. kalau di baris ini filenya adalah iostream

Line 2 : Using namespace std digunakan untuk memanggil namespace yang memiliki nama 'std'. Namespace 'std' merupakan standar namespace dari C++ yang dapat kita gunakan untuk memanggil class/object/fungsi yang terdapat di dalam namespace tersebut.

Line 3 : file sys/types.h yang terdapat di library c/c++ yang menjadi parameter include ini digunakan supaya program ini dapat membaca tipe data pid_t

Line 4 : file unistd.h ini di includekan karena program ini memerlukan sebuah fungsi forking untuk pembuatan process child, sehingga untuk melakukan forking diperlukan pemanggilan file ini dari library c supaya program bisa jalan

Line 5 : sebuah file di librari c/c++ dimana file tersebut terdapat fungsi fungsi yang berurusan dengan waktu. namun di program kita ini, kita memanggil file tersebut dari library dengan #include supaya kita dapat menggunakan fungsi wait();

Line 7 : komentar

Line 8 : komentar

Line 9 : komentar

Line 10 : sebuah main dari program dengan parameter void

Line 11 : mendeklarasikan sebuah variabel bernama child_pid dengan tipe data pid_t. pid_t adalah sebuah tipe data int yang digunakan khusus untuk menampung PID

Line 12 : mendeklarasikan variabel status dengan tipe data int

Line 13 : mendeklarasikan variabel wait_result dengan tipe data pid_t. pid_t adalah sebuah tipe data int yang digunakan khusus untuk menampung PID

Line 14 : melakukan forking yang dimana nantinya return value atau output dari fungsi fork() itu nanti akan disimpan dalam sebuah variabel yang bernama child_pid.

Line 15 : melakukan pengondisian atau pengecekan variabel child_pid. jika child_pid == 0, maka forking pembuatan child process berhasil. return value dari forking = 0 menandakan PID dari process child. jika child terbentuk, otomatis forking berhasil.

Line 16 : komentar

Line 17 : Mendapatkan pid dari child menggunakan getpid() kemudian mencetaknya.

Line 18 : Execl merupakan system call yang berfungsi untuk mengeksekusi file. Pada kasus ini, child process mengeksekusi perintah ls yang filenya berada di /bin/ls dengan argumen -l dan /home. Fungsi execl dapat dimasukkan banyak parameter. Namun, parameter utama yang harus dimasukkan ada 3, yaitu path dari file yang akan dieksekusi, argumen perintah (bisa lebih dari satu), dan NULL (sebagai penanda akhiran dari argumen). System call execl akan mengganti process image sebelumnya dengan process image yang baru. Sehingga jika execl berhasil dijalankan, maka setelah execl selesai dijalankan oleh proses child dan diterminasi, proses child tersebut juga akan ikut diterminasi. Namun jika execl gagal dijalankan, maka proses child akan tetap berjalan.

Line 19 : komentar

Line 20 : Mencetak "Could not execl file /bin/ls" (hanya dicetak jika execl gagal)

Line 21 : Keluar dari program dengan exit status 1.

Line 22 : komentar

Line 23 : Jika pengondisian ada line 15 tidak terpenuhi dan value dari child_pid > 0 (parent process), jalankan line 24-26.

Line 24 : komentar

Line 25 : Mendapatkan pid dari parent menggunakan getpid() kemudian mencetaknya.

Line 26 : Mencetak child_pid (pid dari child process)

Line 27 : Jika pengondisian pada line 15 dan 23 tidak terpenuhi (berarti child process gagal dibentuk), maka jalankan line 28-29.

Line 28 : Mencetak "The fork system call failed to create a new process "

Line 29 : Keluar dari program dengan exit status 1.

Line 31 : komentar

Line 32 : komentar

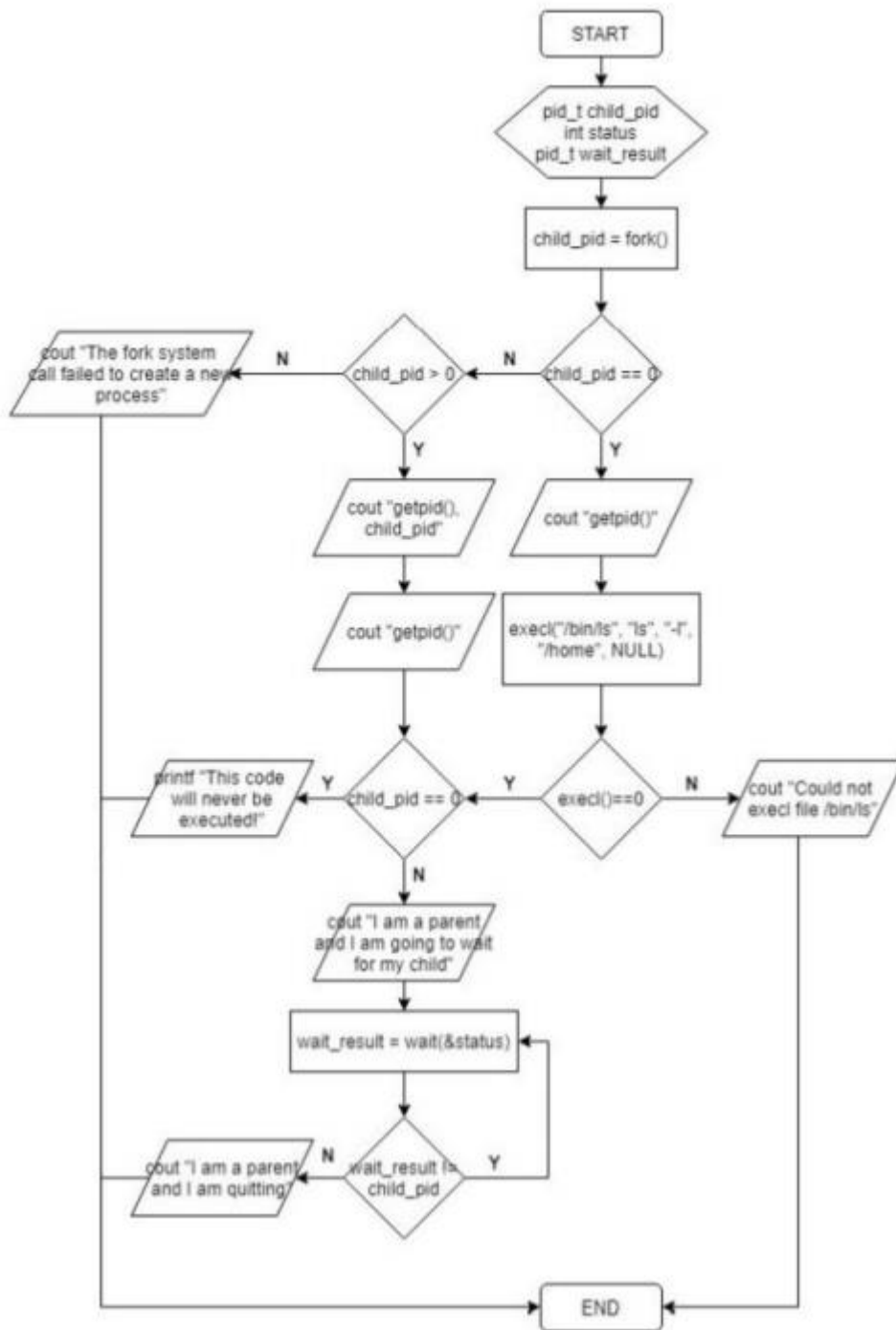
- Line 33 : Mendapat pid dari parent menggunakan getpid() lalu dicetak " "I am a happy, healthy process and my pid = "
- Line 34 : Jika child_pid nya sama dengan 0
- Line 36 : Jika kondisi pada line 34 terpenuhi maka dicetak bahwa kode ini tidak akan pernah dieksekusi
- Line 37 : Jika child_pid tidak sama dengan 0 (dalam arti, kondisi selain sama dengan 0)
- Line 39 : Jika sesuai dengan kondisi pada line 37, maka akan dicetak bahwa proses parent akan menunggu proses child.
- Line 41 : ada perintah 'do' untuk melakukan perulangan terlebih dahulu baru akan memeriksa kondisi diakhir
- Line 44 : parent akan menunggu sinyal yang mengirimi tanda bila proses child diterminasi
- Line 45 : Memeriksa kondisi apakah wait_result tidak sama dengan child_pid, jika belum memenuhi maka akan melakukan perulangan lagi, namun jika sudah maka akan masuk ke line selanjutnya.
- Line 46 : setelah proses wait selesai maka akan dicetak bahwa proses parent telah selesai.
- Line 48 : Menandakan bahwa proses parent telah selesai

```

rheza@rheza-VirtualBox:~$ g++ -o fork05 fork05.cpp
rheza@rheza-VirtualBox:~$ ./fork05
I am the parent and my pid = 7576
My child has pid = 7577
I am a happy, healthy process and my pid = 7576
I am a parent and I am going to wait for my child
I am a child and my pid = 7577
total 16
drwxr-xr-x  2 lika  lika  4096 Apr 12 12:08 lika
drwxr-xr-x  2 pulo  pulo  4096 Apr 12 12:24 pulo
drwxr-xr-x 19 rheza rheza 4096 Mei  4 16:27 rheza
drwxr-xr-x  3 rika  rika  4096 Apr 12 12:24 rika
I am a parent and I am quitting.

```

Analisis : Pada percobaan fork05, langsung ditampilkan sebuah parent process, hal tersebut berarti bahwa pada pemanggilan fungsi fork() yang merupakan pembuatan proses baru, return value nya adalah > 0 sehingga yang dieksekusi adalah parent process. Kemudian karena return lebih dari 0, maka kita mengeksekusi fungsi wait yang berfungsi untuk menunggu child process untuk diakhiri, kemudian kita akan berganti ke child process di mana return pada fungsi fork() adalah 0 dan kode untuk child process dieksekusi, dan akan menampilkan list yang ada di directoy bin yang berisi permission dari setiap user dan juga user itu sendiri, dan kapan terakhir kali melakukan login ke dalam linux, setelah menampilkan itu maka child process diakhiri dengan perintah exit. Dan setelah child process diakhiri, maka parent process pun akan mengakhiri dirinya juga.



Percobaan 6

```
1. #include <iostream>
2. using namespace std;
3. #include <sys/types.h>
4. #include <unistd.h>
5. #include <sys/wait.h>
6. /* pid_t fork() dideklarasikan pada unistd.h.
7. pid_t adalah type khusus untuk process id yg ekuivalen dg int
8. */
9.
10. int main(void) {
11. pid_t child_pid;
12. int status;
13. pid_t wait_result;
14. child_pid = fork();
15. if (child_pid == 0) {
16. /* kode ini hanya dieksekusi proses child */
17. cout << "I am a child and my pid = " << getpid() << endl;
18. execl("fork3", "goose", NULL);
19. /* jika execl berhasil kode ini tidak pernah digunakan */
20. cout << "Could not execl file fork3" << endl;
21. exit(1);
22. /* exit menghentikan hanya proses child */
23. }
24. else if (child_pid > 0) {
25. /* kode ini hanya mengeksekusi proses parent */
26. cout << "I am the parent and my pid = " << getpid() << endl;
27. cout << "My child has pid = " << child_pid << endl;
28. }
29. else {
30. cout << "The fork system call failed to create a new process" <<
endl;
31. exit(1);
32. }
33.
34. /* kode ini hanya dieksekusi oleh proses parent karena
35. child mengeksekusi dari "fork3" atau keluar */
36. cout << "I am a happy, healthy process and my pid = " << getpid() <<
endl;
37. if (child_pid == 0) {
38. /* kode ini tidak pernah dieksekusi */
39. printf("This code will never be executed!\n");
40. }
41. else {
42. /* kode ini hanya dieksekusi oleh proses parent */
43. cout << "I am a parent and I am going to wait for mychild" << endl;
44. do {
45. /* parent menunggu sinyal SIGCHLD mengirim tanda
```

```

46.  bila proses child diterminasi */
47.  wait_result = wait(&status);
48.  } while (wait_result != child_pid);
49.  cout << "I am a parent and I am quitting." << endl;
50.  }
51.  return 0;
52.  }
53.

```

Line 1 : Import library iostream yang berfungsi sebagai input dan output seperti cout, cin, dll include artinya berarti library ini disertakan dalam program

Line 2 : Digunakan untuk memanggil namespace yang memiliki nama 'std'. misal jika kita tidak mendeklarasikan line-2 ini, maka setiap kali kita ingin cout, harus diawali dengan std::

Line 3 : Untuk mengenali tipe data yang digunakan dalam system source code

Line 4 : Standard symbolic untuk constants dan types

Line 5 : Deklarasi fungsi untuk mengenali system call waiting, wait().

Line 6 : komentar

Line 7 : komentar

Line 8 : komentar

Line 9 : -

Line 10 : awal fungsi bernama main, parameter void.

Line 11 : Deklarasi variable child_pid bertipe pid_t, valuenya ekuivalen dengan int, hanya saja untuk proses.

Line 12 : Deklarasi variable status bertipe int.

Line 13 : Deklarasi variable wait_result bertipe pid_t, valuenya ekuivalen dengan int, hanya saja untuk proses.

Line 14 : Assignment yang melakukan system call fork(), sekaligus mereturn value kepada variable child_pid. System call fork() berfungsi untuk menduplikasi program yang kemudian berjalan sebagai child proses

Line 15 : If statement, jika value dari child_pid = 0 (child process yang dimaksud) maka lakukan code yang berada didalam brackets (jika lebih dari satu baris, jika == 1 baris, maka eksekusi baris dibawahnya).

Line 16 : komentar

Line 17 : Cout, built in function dari library iostream. Dan juga bentuk sederhana dari std::cout, karena program sudah menggunakan using namespace std. menampilkan pesan pada layer. Getpid(), anggota dari library sys/types.h dan unistd.h. berfungsi untuk mereturn

value dari process id yang berjalan. Endln, anggota dari library iostream. Berfungsi sebagai end line (enter)

Line 18 : Anggota dari library unistd.h, yang berfungsi mereplace proses yang saat itu berjalan ke proses baru, dengan parameter fork3, berarti mencari file/path fork3

Line 19 : komentar

Line 20 : Menampilkan pesan di layer, yang berfungsi ketika line sebelumnya tidak menemukan file yang dimaksud

Line 21 : Exit(1), terminate program secara paksa.

Line 22 : komentar

Line 23 : Tutup bracket if

Line 24 : Else if statement, dimana statement ini berjalan ketika forking berhasil. Maka jalankan kode yang berada di dalam brackets

Line 25 : komentar

Line 26 : Cout, built in function dari library iostream. Dan juga bentuk sederhana dari std::cout, karena program sudah menggunakan using namespace std. menampilkan pesan pada layer. Getpid(), anggota dari library sys/types.h dan unistd.h. berfungsi untuk mereturn value dari process id yang berjalan. Endln, anggota dari library iostream. Berfungsi sebagai end line (enter)

Line 27 : Sama seperti line 26, hanya saja child_pid sudah memiliki nilai yaitu dari return value system call fork() kepada process parent.

Line 28 : Tutup brackets

Line 29 : Else statement, statement ini berjalan ketika forking gagal.

Line 30 : Menampilkan pesan error

Line 31 : Exit(1) , terminate process secara paksa

Line 32 : Tutup brackets

Line 33 : -

Line 34 : komentar

Line 35 : komentar

Line 36 : Cout, built in function dari library iostream. Dan juga bentuk sederhana dari std::cout, karena program sudah menggunakan using namespace std. menampilkan pesan pada layer. Getpid(), anggota dari library sys/types.h dan unistd.h. berfungsi untuk mereturn value dari process id yang berjalan. Endln, anggota dari library iostream. Berfungsi sebagai end line (enter)

Line 37 : If statement, dijalankan pada child proses

Line 38 : komentar

Analisis : Pada percobaan fork06, langsung ditampilkan sebuah parent process, hal tersebut berarti bahwa pada pemanggilan fungsi fork() yang merupakan pembuatan proses baru, return value nya adalah > 0 sehingga yang dieksekusi adalah parent process. Kemudian karena return lebih dari 0, maka kita mengeksekusi fungsi wait yang berfungsi untuk menunggu child process untuk diakhiri, kemudian kita akan berganti ke child process di mana return pada fungsi fork() adalah 0 dan kode untuk child process dieksekusi, kemudian setelah itu kita akan menampilkan pid dan ppid dari process goose yang merupakan argument dari perintah execl.

Percobaan 7

1. Dmesg | more

```
rheza@rheza-VirtualBox:~$ dmesg | more
[ 0.000000] Linux version 5.3.0-46-generic (build@lcy01-amd64-013) (gcc ver
sion 7.5.0 (Ubuntu 7.5.0-3ubuntu1~18.04)) #38-18.04.1-Ubuntu SMP Tue Mar 31 04:
17:56 UTC 2020 (Ubuntu 5.3.0-46.38-18.04.1-generic 5.3.18)
[ 0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.3.0-46-generic root=UUI
D=0a71eb4f-cec8-44d2-a2a9-b60cad097e56 ro quiet splash
[ 0.000000] KERNEL supported cpus:
[ 0.000000] Intel GenuineIntel
[ 0.000000] AMD AuthenticAMD
[ 0.000000] Hygon HygonGenuine
[ 0.000000] Centaur CentaurHauls
[ 0.000000] zhaoxin Shanghai
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x001: 'x87 floating point reg
isters'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x002: 'SSE registers'
[ 0.000000] x86/fpu: Supporting XSAVE feature 0x004: 'AVX registers'
[ 0.000000] x86/fpu: xstate_offset[2]: 576, xstate_sizes[2]: 256
[ 0.000000] x86/fpu: Enabled xstate features 0x7, context size is 832 bytes,
using 'standard' format.
[ 0.000000] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[ 0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
[ 0.000000] BIOS-e820: [mem 0x0000000001000000-0x0000000007ffefffff] usable
[ 0.000000] BIOS-e820: [mem 0x0000000007fff0000-0x0000000007fffff] ACPI data
[ 0.000000] BIOS-e820: [mem 0x00000000fec00000-0x00000000fec00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000fee00000-0x00000000fee00fff] reserved
[ 0.000000] BIOS-e820: [mem 0x00000000fffc0000-0x00000000fffff] reserved
```

Analisis : Mencetak atau mengontrol kernel ring buffer

2. Free

```
rheza@rheza-VirtualBox:~$ free
              total        used         free       shared  buff/cache   available
Mem:           2035480        988320        166292         16204         880868         865168
Swap:          483800          46816         436984
```

Analisis : Perintah free digunakan untuk menampilkan jumlah total dari memori fisik dan memori swap yang tersedia dan yang telah digunakan, begitu juga dengan buffer dan cache yang digunakan oleh kernel. Dan kita melihat bahwa memory yang tersedia adalah senilai 865168 dan Swap yang tersedia adalah 436984.

3. Cat /proc/meminfo

```
rheza@rheza-VirtualBox:~$ cat /proc/meminfo
MemTotal:        2035480 kB
MemFree:         166332 kB
MemAvailable:    865236 kB
Buffers:         53428 kB
Cached:          759064 kB
SwapCached:      1036 kB
Active:          834144 kB
Inactive:        838348 kB
Active(anon):    416576 kB
Inactive(anon):  459628 kB
Active(file):    417568 kB
Inactive(file):  378720 kB
Unevictable:     32 kB
Mlocked:         32 kB
SwapTotal:       483800 kB
SwapFree:        436984 kB
Dirty:           4 kB
Writeback:       0 kB
AnonPages:      859496 kB
```

Analisis : Dan setelah kita gunakan `cat /proc/meminfo` kita melihat bahwa `MemAvailable` adalah 865236 yang mana mempunyai perbedaan yang kecil dari yang ditampilkan di `free`, dan `SwapFree` mempunyai angka yang sama.

4. Ls -lr /.

```
rheza@rheza-VirtualBox:~$ ls -lr /.
total 483908
lrwxrwxrwx  1 root root          29 Mei  4 01:56 vmlinuz.old -> boot/vmlinuz-5.3
.0-46-generic
lrwxrwxrwx  1 root root          29 Mei  4 01:56 vmlinuz -> boot/vmlinuz-5.3.0-5
1-generic
drwxr-xr-x 14 root root        4096 Feb  4 01:30 var
drwxr-xr-x 11 root root        4096 Feb  4 01:25 usr
drwxrwxrwt 14 root root        4096 Mei  4 16:30 tmp
dr-xr-xr-x 13 root root           0 Mei  4 00:12 sys
-rw-----  1 root root 495416320 Mar 18 12:10 swapfile
drwxr-xr-x  2 root root        4096 Feb  4 01:22 srv
drwxr-xr-x 12 root root        4096 Mei  4 01:24 snap
drwxr-xr-x  2 root root       12288 Mei  4 01:53 sbin
drwxr-xr-x 28 root root         900 Mei  4 11:39 run
drwx-----  4 root root        4096 Apr 14 18:56 root
dr-xr-xr-x 199 root root           0 Mei  4 00:12 proc
drwxr-xr-x  2 root root        4096 Feb  4 01:22 opt
drwxr-xr-x  2 root root        4096 Feb  4 01:22 mnt
drwxr-xr-x  2 root root        4096 Feb  4 01:22 media
drwx-----  2 root root       16384 Mar 18 12:10 lost+found
drwxr-xr-x  2 root root        4096 Feb  4 01:22 lib64
drwxr-xr-x 21 root root        4096 Mar 18 12:28 lib
lrwxrwxrwx  1 root root          32 Mei  4 01:56 initrd.img.old -> boot/initrd.i
```

5. Free

```
rheza@rheza-VirtualBox:~$ free
              total        used         free       shared  buff/cache   available
Mem:           2035480        987568        166592         16204         881320         865936
Swap:           483800          46816        436984
```

6. Free

```
rheza@rheza-VirtualBox:~$ free
              total        used         free       shared  buff/cache   available
Mem:           2035480       1060704        115164         19364         859612         790804
Swap:           483800          49904        433896
```

7. Ps -uax

```
rheza@rheza-VirtualBox:~$ ps -uax
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.4 225684  8272 ?        Ss   01:42   0:09 /lib/systemd/s
root         2  0.0  0.0     0     0 ?        S    01:42   0:00 [kthreadd]
root         3  0.0  0.0     0     0 ?        I<   01:42   0:00 [rcu_gp]
root         4  0.0  0.0     0     0 ?        I<   01:42   0:00 [rcu_par_gp]
root         6  0.0  0.0     0     0 ?        I<   01:42   0:00 [kworker/0:0H-
root         8  0.0  0.0     0     0 ?        I<   01:42   0:00 [mm_percpu_wq]
root         9  0.0  0.0     0     0 ?        S    01:42   0:01 [ksoftirqd/0]
root        10  0.0  0.0     0     0 ?        I    01:42   0:02 [rcu_sched]
root        11  0.0  0.0     0     0 ?        S    01:42   0:00 [migration/0]
root        12  0.0  0.0     0     0 ?        S    01:42   0:00 [idle_inject/0
root        14  0.0  0.0     0     0 ?        S    01:42   0:00 [cpuhp/0]
root        15  0.0  0.0     0     0 ?        S    01:42   0:00 [kdevtmpfs]
root        16  0.0  0.0     0     0 ?        I<   01:42   0:00 [netns]
root        17  0.0  0.0     0     0 ?        S    01:42   0:00 [rcu_tasks_kth
root        18  0.0  0.0     0     0 ?        S    01:42   0:00 [kauditd]
root        19  0.0  0.0     0     0 ?        S    01:42   0:00 [khungtaskd]
root        20  0.0  0.0     0     0 ?        S    01:42   0:00 [oom_reaper]
root        21  0.0  0.0     0     0 ?        I<   01:42   0:00 [writeback]
root        22  0.0  0.0     0     0 ?        S    01:42   0:00 [kcompactd0]
root        23  0.0  0.0     0     0 ?        SN   01:42   0:00 [ksmd]
root        24  0.0  0.0     0     0 ?        SN   01:42   0:00 [khugepaged]
```

Latihan

1. Ubahlah program fork5.cpp pada percobaan 5 untuk mengeksekusi perintah yang ekuivalen dengan

- Ls -al /etc

```
rheza@rheza-VirtualBox:~$ g++ -o fork052 fork052.cpp
rheza@rheza-VirtualBox:~$ ./fork052
I am the parent and my pid = 7698
My child has pid = 7699
I am a happy, healthy process and my pid = 7698
I am a parent and I am going to wait for my child
I am a child and my pid = 7699
total 1092
drwxr-xr-x 123 root root 4096 Mei 4 11:50 .
drwxr-xr-x 24 root root 4096 Mei 4 01:56 ..
drwxr-xr-x 3 root root 4096 Feb 4 01:25 acpi
-rw-r--r-- 1 root root 3028 Feb 4 01:22 adduser.conf
drwxr-xr-x 2 root root 4096 Mei 4 11:50 alternatives
-rw-r--r-- 1 root root 401 Mei 29 2017 anacrontab
-rw-r--r-- 1 root root 433 Okt 2 2017 apg.conf
drwxr-xr-x 6 root root 4096 Feb 4 01:24 apm
drwxr-xr-x 3 root root 4096 Feb 4 01:25 apparmor
drwxr-xr-x 8 root root 4096 Mei 4 01:57 apparmor.d
drwxr-xr-x 4 root root 4096 Apr 12 11:38 appport
-rw-r--r-- 1 root root 769 Apr 4 2018 appstream.conf
drwxr-xr-x 7 root root 4096 Mar 18 12:29 apt
drwxr-xr-x 3 root root 4096 Feb 4 01:26 avahi
-rw-r--r-- 1 root root 2319 Apr 5 2018 bash.bashrc
-rw-r--r-- 1 root root 45 Apr 2 2018 bash_completion
drwxr-xr-x 2 root root 4096 Mei 4 01:56 bash_completion.d
-rw-r--r-- 1 root root 367 Jan 27 2016 bindresvport.blacklist
drwxr-xr-x 2 root root 4096 Apr 20 2018 binfmt.d
```

- Cat fork2

```
rheza@rheza-VirtualBox:~$ g++ -o fork052 fork052.cpp
rheza@rheza-VirtualBox:~$ ./fork052
I am the parent and my pid = 7726
My child has pid = 7727
I am a happy, healthy process and my pid = 7726
I am a parent and I am going to wait for my child
I am a child and my pid = 7727
This is process 7727
x is 5
This is process 7728
x is 5
This is process 7727
x is 6
This is process 7728
x is 6
This is process 7727
x is 7
This is process 7728
x is 7
```

- ./fork2

```

rheza@rheza-VirtualBox:~$ g++ -o fork052 fork052.cpp
rheza@rheza-VirtualBox:~$ ./fork052
I am the parent and my pid = 7726
My child has pid = 7727
I am a happy, healthy process and my pid = 7726
I am a parent and I am going to wait for my child
I am a child and my pid = 7727
This is process 7727
x is 5
This is process 7728
x is 5
This is process 7727
x is 6
This is process 7728
x is 6
This is process 7727
x is 7
This is process 7728
x is 7

```

2. Informasi apa saja mengenai manajemen memory yang ditampilkan pada perintah dmesg pada percobaan Anda ?

- Pesan Kernel
- Pesan tingkat user
- Mail system
- System daemons
- Pesan keamanan
- Pesan syslogd internal
- Line printer subsystem
- Network News subsystem

3. Bagaimana informasi yang ditampilkan dengan perintah free pada percobaan Anda ?

- Total, used, free, shared, buff/cache, dan available memory
- Total, used, free, shared, buff/cache, dan available swap

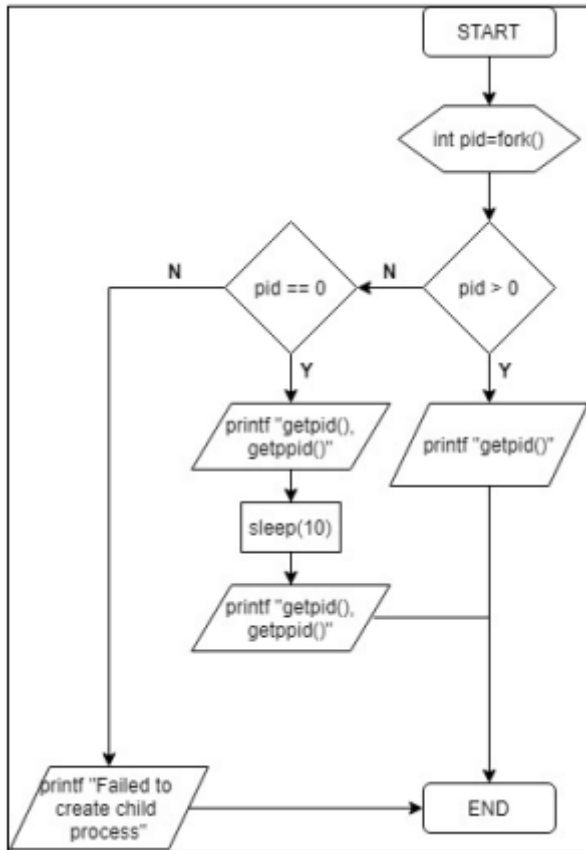
4. Apa isi file /proc/meminfo pada percobaan yang Anda lakukan ?

- MemTotal
- MemFree
- MemAvailable
- Buffers
- Cached
- SwapCached
- Active
- Inactive
- Active(anon)
- Inactive(anon)
- Active(file)
- Inactive(file)
- Unevictable
- Mlocked
- SwapTotal
- SwapFree

5. Berapa besar memory yang digunakan setelah percobaan 7 dengan perintah ps -uax ?
 - 1018016
6. Lakukan hal yang sama dengan percobaan 7 untuk melihat perubahan memory setelah dilakukan beberapa proses pada shell. Tentukan perintah yang dilakukan misalnya membuka browser dan perhatikan hal – hal berikut :
 - Informasi apa saja yang ditampilkan dengan perintah free ?
 - a.Total, used, free, shared, buff/cache, dan available memory
 - b.Total, used, free, shared, buff/cache, dan available swap
 - Informasi apa saja yang disimpan file /proc/meminfo ?
 - a.Jumlah total RAM fisik
 - b.Jumlah RAM fisik, tidak digunakan oleh system
 - c.Jumlah RAM fisik digunakan untuk buffer file
 - d.Jumlah RAM fisik digunakan untuk memori cache
 - e.Jumlah swap digunakan untuk memori cache
 - Berapa besar kapasitas memory total ?
 - a.2035480
 - Berapa kapasitas memory yang sudah terpakai ?
 - a.1025016
 - Berapa kapasitas memory yang belum terpakai ?
 - a.88976
 - Berapa kapasitas memory yang digunakan sharing beberapa proses ?
 - a.17664
 - Berapa kapasitas buffer cache ?
 - a.921488

Orphan proses

```
1. #include <stdio.h>
2. #include <sys/types.h>
3. #include <unistd.h>
4.
5. int main()
6. {
7.     // fork() Create a child process
8.
9.     int pid = fork();
10.    if (pid > 0)
11.    {
12.        //getpid() returns process id
13.        // while getppid() will return parent process id
14.        printf("Parent process\n");
15.        printf("ID : %d\n\n",getpid());
16.    }
17.    else if (pid == 0)
18.    {
19.        printf("Child process\n");
20.        // getpid() will return process id of child process
21.        printf("ID: %d\n",getpid());
22.        // getppid() will return parent process id of child process
23.
24.        printf("Parent -ID: %d\n\n",getppid());
25.
26.        sleep(10);
27.
28.        // At this time parent process has finished.
29.        // So if u will check parent process id
30.        // it will show different process id
31.        printf("\nChild process \n");
32.        printf("ID: %d\n",getpid());
33.        printf("Parent -ID: %d\n",getppid());
34.    }
35.    else
36.    {
37.        printf("Failed to create child process");
38.    }
39.    return 0;
40. }
41.
42. /*
43. https://www.includehelp.com/c-programs/orphan-process.aspx
44. */
```



Line 1 : program mengimport file header bernama *stdio.h* yang berisi fungsi fungsi dasar operasi input dan ouput

Line 2 : program mengimport file header bernama *sys/types.h* yang berisi nama nama variable yang ada pada system llinux. Misalnya uid_t, pid_t dan lain lain

Line 3 : program mengimport file header bernama *unistd.h* yang mendefinisikan konstanta simbolik dan jenis lainnya, dan mendeklarasikan fungsi lain-lain.

Line 4 : -

Line 5 : Fungsi main() adalah fungsi utama dalam program. Fungsi ini akan dieksekusi pertamakali saat program dijalankan.

Line 6 : { merupakan tanda awal suatu block kode, dalam baris ini merupakan tanda { menandakan awal dari fungsi main

Line 7 : -

Line 8 : // menandakan sebuah komentar. Baris ini tidak akan dieksekusi ketika program dijalankan. Komentar pada baris ini menyatakan bahwa fungsi fork akan memnuat child proses

Line 9 : mendeklarasikan sebuah variable bernama *pid* dengan tipe data int. Niali dari varibel ini sama dengan return value dari fungsi fork(); Fungsi fork sendiri adalah sebuah fungsi yang akan membuat proses baru dengan menduplikasi proses yang sedang dipanggil. Proses hasil duplikasi disebut child proses dan proses yang diduplikasi disebut parent proses

Line 10 : merupakan seleksi if yang akan menguji apakah nilai dari variable *pid* lebih besar dari nol

Line 11 : { merupakan tanda awal block program yang akan dieksekusi bila nilai variable *pid* lebih dari nol

Line 12 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa fungsi `getpid()` akan mengembalikan nilai berupa process id. `Getpid()` adalah sebuah fungsi yang digunakan untuk menreturn / mengembalikan process id dari proses yang sedang dipanggil

Line 13 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa `getppid()` akan mengembalikan nilai berupa parent process id. `Getppid()` adalah sebuah fungsi yang digunakan untuk menreturn / mengembalikan process id dari parent proses dari proses yang sedang dipanggil

Line 14 : `printf` akan menampilkan kalimat “Parent process”

Line 15 : `printf` akan menampilkan “ID : %d“ dengan %d adalah return value dari fungsi `getpid()` dengan tipe integer yaitu pid dari proses yang sedang berjalan

Line 16 : } merupakan tanda akhir block program yang akan dieksekusi bila nilai variable *pid* lebih dari nol

Line 17 : merupakan seleksi else if yang akan menguji apakah nilai dari variable *pid* sama dengan nol

Line 18 : { merupakan tanda awal block program yang akan dieksekusi bila nilai variable *pid* sama dengan nol

Line 19 : `printf` akan menampilkan kalimat “Child Process”

Line 20 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa “fungsi `getpid()` akan mengembalikan nilai berupa process id dari child proses”

Line 21 : `printf` akan menampilkan “ID : %d“ dengan %d adalah return value dari fungsi `getpid()` dengan tipe integer yaitu pid dari proses yang sedang berjalan

Line 22 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa “fungsi `getppid()` akan mengembalikan nilai berupa parent process id dari child proses”

Line 23 : `printf` akan menampilkan “Parent -ID : %d“ dengan %d adalah return value dari fungsi `getppid()` dengan tipe integer yaitu pid dari proses yang sedang berjalan

Line 24 : -

Line 25 : fungsi `sleep(10)` akan menghentikan program selama 10 detik.

Line 26 :-

Line 27 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa “pada saat ini parent proses telah selesai”

Line 28 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa “jika kita melakukan pengecekan parent proses id”

Line 29 : // merupakan komentar. Pada baris ini komentar menyatakan bahwa “akan ditampilkan proses id yang berbeda”

Line 30 : printf akan menampilkan kalimat “Child Process”

Line 31 : printf akan menampilkan “ID : %d“ dengan %d adalah return value dari fungsi getpid() dengan tipe integer yaitu pid dari proses yang sedang berjalan

Line 32 : printf akan menampilkan “Parent -ID : %d“ dengan %d adalah return value dari fungsi getppid() dengan tipe integer yaitu pid dari proses yang sedang berjalan

Line 33 : } merupakan tanda akhir block program yang akan dieksekusi bila nilai variable *pid* sama dengan nol

Line 34 : merupakan seleksi else yang akan dilakukan ketika nilai pid tidak memenuhi seleksi if dan else if sebelumnya

Line 35 : { merupakan tanda awal block program yang akan dieksekusi bila nilai variable *pid* tidak lebih dari dan sama dengan nol

Line 36 : printf akan menampilkan “Failed to create child process”

Line 37 : } merupakan tanda akhir block program yang akan dieksekusi bila nilai variable *pid* tidak lebih dari dan sama dengan nol

Line 38 : -

Line 39 : fungsi main akan mengembalikan nilai 0

Line 40 : } merupakan tanda akhir block program dari fungsi main

Line 41 : -

Line 42 : /* merupakan tanda awal komentar

Line 43 : <https://www.includehelp.com/c-programs/orphan-process.aspx> merupakan link sumber program ini

Line 44 : */ merupakan tanda akhir komentar

```
rheza@rheza-VirtualBox:~$ g++ orphan.c -o orphan
rheza@rheza-VirtualBox:~$ ./orphan
Parent process
ID : 3134

Child process
ID: 3135
Parent -ID: 1043

rheza@rheza-VirtualBox:~$
Child process
ID: 3135
Parent -ID: 1043
```

Program orphan akan membuat proses child dengan menggunakan fungsi fork. Return value dari fungsi ini kemudian di assign ke dalam variable bernama pid.

Karena nilai pid lebih dari nol maka ditampilkan parent proses id yaitu 3134

Karena nilai pid sama dengan nol maka program juga akan menampilkan child proses id dan parent proses id yaitu ID : 3135 dan parent -ID : 1043

Kemudian program dihentikan sementara selama 10 detik

Pada saat ini parent proses telah selesai dieksekusi, oleh karena itu, ketika ditampilkan parent -ID maka nilainya akan berbeda dari sebelumnya. (sekarang 1034, sebelumnya 3134).

Inilah yang dinamakan orphan process yaitu sebuah proses dimana parent procese yang selesai/ sudah diterminasi tanpa menunggu untuk child prosesnya terminasi. Proses orphan akan segera di adopsi oleh proses init, jika parent proses mati/gagal.

Zombie.cpp

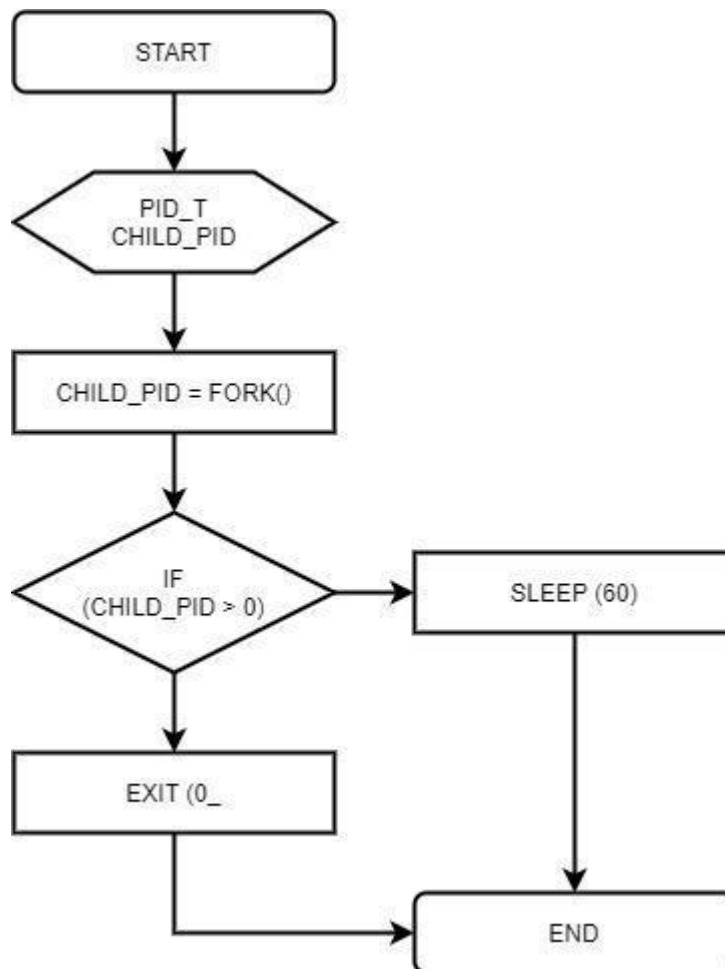
```
1  #include <stdlib.h>
2  #include <sys/types.h>
3  #include <unistd.h>
4  int main ()
5  {
6      pid_t child_pid;
7
8      /* Create child/
9      child_pid = fork ();
10     if (child_pid > 0) {
11
12         / Parent process /
13         sleep (60);
14     }
15     else {
16
17         /Child process. Exit immediately. /
18         exit (0);
19     }
20     return 0;
21 }
```

Analisa :

Baris -	Code	Penjelasan
1	ude <stdlib.h>	h adalah header dari <i>standart library</i> bahasa pemrograman C yang fungsinya untuk alokasi memori, kotrol proses, konversi, dll.
2	ude <sys/types.h>	pes.h adalah header dari <i>standart library</i> bahasa pemrograman C yang fungsinya untuk mendefinisikan tipe data, seperti pid_t.
3	ude <unistd.h>	h adalah header dari <i>standart library</i> bahasa pemrograman C yang fungsinya untuk mendeklarasikan variable dan fungsi dari user yang terdiri dari API sitem operasi POSIX, seperti size_t dan fork().
4	ain ()) adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.

5		untuk membuka deklarasi fungsi main().
6	<code>pid_t child_pid;</code>	digunakan untuk ID proses dan ID grup proses. pid adalah variable yang digunakan untuk menyimpan ID proses dari proses fork(), dimana hasilnya akan menjadi child process.
7		
8	<code>/* Create child/</code>	eri komentar, untuk membuar child process.
9	<code>child_pid = fork();</code>	digunakan untuk membuat proses baru yang disebut chld process, yang berjalan bersamaan dengan proses yang membuat panggilan fork() (parent process), Setelah child process dibuat, kedua proses akan menjalankan instruksi berikutnya mengikuti fork() system call. aris ini, system call fork diassign pada child_pid, sehingga child_pid akan menyimpan child process.
10	<code>if (child_pid > 0) {</code>	bat return value untuk menunjukkan apakah proses fork() berhasil atau tidak : ve : pembuatan child process gagal. nbali ke child process yang baru dibuat. ve : kembali ke parent process dan PID berisi ID proses dari child process yang dibuat. aris ini, jika child_pid (child process) berhasil dibuat, maka

		perintah di bawahnya akan dijalankan.
11		
12	<code>/ Parent process /</code>	eri komentar, parent process berjalan.
13	<code>Sleep (60);</code>	digunakan untuk membuat pekerjaan tiruan. Pekerjaan tiruan membantu memnunda eksekusi. Perintah ini menjeda eksekusi untuk sejumlah waktu yang ditentukan. baris ini, eksekusi ditunda selama 60 detik.
14	<code>}</code>	untuk menutup kondisi pertama.
15	<code>else {</code>	digunakan untuk memberi kondisi. untuk membuka kondisi kedua.
16		
17	<code>/ Child process. Exit immediately./</code>	eri komentar, child process berjalan dan akan segera keluar.
18	<code>exit (0);</code>	digunakan untuk keluar dari shell dimana ia sedang berjalan. baris ini, proses akan keluar dan mengembalikan nilai 0.
19	<code>}</code>	untuk menutup kondisi kedua.
20	<code>return 0;</code>	n digunakan untuk keluar dari fungsi shell. baris ini, proses akan keluar dari fungsi dan mengembalikan nilai 0.
21	<code>}</code>	untuk menutup deklarasi fungsi main().



```

rheza@rheza-VirtualBox: ~
File Edit View Search Terminal Help
rheza@rheza-VirtualBox:~$ ./zombie

rheza@rheza-VirtualBox: ~
File Edit View Search Terminal Help
.13 /org/gtk/gvfs/ex
1571 1045 - /usr/lib/gvfs/gvfsd-dnssd --spawner :1.1
3 /org/gtk/gvfs/exec
1581 836 - /usr/lib/gvfs/gvfsd-metadata
1604 1499 - ./zombie
1605 1604 - [zombie] <defunct>
1610 2 - [kworker/u2:0-ev]
1612 1499 - ps -e -o pid,ppid,status,cmd
rheza@rheza-VirtualBox:~$
  
```

Analisa : Proses zombie adalah proses yang sudah selesai eksekusi namun masih berada di entry tabel proses untuk di laporkan ke proses parent. Pada percobaan di atas, terlihat bahwa proses child sudah selesai melakukan eksekusi namun parent process masih melakukan child. Padahal, apabila child process sudah selesai seharusnya parent process ikut selesai, karena kasus ini parent process masih menjalani proses sleep dimana masih terdapat pada entry tabel proses,

Zombiesafe.cpp

```

1. /*Here is a method to avoid the process from becoming Zombie. The cleanup is done b
   y using signal handler */
2. #include <stdlib.h>
3. #include <sys/types.h>
4. #include <unistd.h>
5. #include <signal.h>
6. #include <string.h>
7. #include <sys/wait.h>
8. sig_atomic_t child_exit_status;
9. void clean_up_child_process (int signal_number)
10. {
11.     /* Clean up the child process. */
12.     int status;
13.     wait (&status);
14.     /* Store its exit status in a global variable. */
15.     child_exit_status = status;
16. }
17. int main ()
18. {
19.     pid_t child_pid;
20.     /* Handle SIGCHLD by calling clean_up_child_process. */
21.     struct sigaction sigchld_action;
22.     memset (&sigchld_action, 0, sizeof (sigchld_action));
23.     sigchld_action.sa_handler = &clean_up_child_process;
24.     sigaction (SIGCHLD, &sigchld_action, NULL);
25.     /* Create child*/
26.     child_pid = fork ();
27.     if (child_pid > 0) {
28.         /* Parent process */
29.         sleep (60);
30.     }
31.     else {
32.         /*Child process. Exit immediately. */
33.         exit (0);
34.     }
35.     return 0;
36. }

```

Line -	Code	Penjelasan
1	/*Here is a method to avoid the process from becoming Zombie. The cleanup is done by using signal handler */	Memberi komentar bahwa ini adalah metode untuk menghindari proses zombie terjadi. Proses dilakukan dengan menggunakan pengatur sinyal.
2	#include <stdlib.h>	stdlib.h adalah header dari <i>standart</i> library bahasa pemrograman C yang fungsinya untuk alokasi memori, kontrol proses, konversi, dll.

3	<pre>#include <sys/types.h></pre>	sys/types.h adalah header dari <i>standart library</i> bahasa pemrograman C yang fungsinya untuk mendefinisikan tipe data, seperti <code>pid_t</code> .
4	<pre>#include <unistd.h></pre>	unistd.h adalah header dari <i>standart library</i> bahasa pemrograman C yang fungsinya untuk mendeklarasikan variable dan fungsi dari user yang terdiri dari API sitem operasi POSIX, seperti <code>size_t</code> dan <code>fork()</code> .
5	<pre>#include <signal.h></pre>	signal.h adalah header dari <i>standart library</i> bahasa pemrograman C yang mendefinisikan tipe variabel <code>sig_atomic_t</code> , dua panggilan fungsi, dan beberapa makro untuk menangani sinyal berbeda yang dilaporkan selama eksekusi program.
6	<pre>#include <string.h></pre>	string.h merupakan header yang berisi fungsi-fungsi, makro dan tipe yang digunakan untuk pengoprasian string dan array
7	<pre>#include <sys/wait.h></pre>	<sys/wait.h> merupakan header untuk mendefinisikan system call <code>wait()</code> .
8	<pre>sig_atomic_t child_exit_status</pre>	sig_atomic_t adalah tipe data untuk signal handler. child_exit_status merupakan variabel dari tipe data sig_atomic_t .
9	<pre>clean_up_child_ process (int signal_number)</pre>	tipe data void() digunakan ketika fungsi ini tidak mengembalikan nilai output. clean_up_child_process adalah nama variabel dari tipe data void() . (int signal_number) parameter <code>signal_number</code> dengan tipe data <code>int</code> . tipe data parameter <code>int</code> digunakan untuk menjamin komunikasi dengan <i>signal handler</i> .
10		Syntax untuk membuka deklarasi fungsi <code>void()</code> .

11	<pre>lean up the child process */</pre>	Memberi komentar pembersihan child process
12	<pre>status</pre>	Mendeklarasikan variabel status dengan tipe data int.
13	<pre>(&status)</pre>	Menunggu hingga ada <i>child process</i> yang sudah diterminasi , status berisi exit status dari <i>child process</i> .
14	<pre>ore its exit status in a global variable. */</pre>	Memberi komentar untuk menyimpan status exit nya dalam variabel global.
15	<pre>_exit_status = status;</pre>	Me-assignment nilai yang ada di variabel status ke variabel child_exit_status
16		Syntax untuk menutup deklarasi fungsi void().
17	<pre>main()</pre>	main() adalah fungsi utama dalam program, dimana fungsi ini akan dieksekusi pertama kali saat program dijalankan.
18		Syntax untuk membuka deklarasi fungsi main().
19	<pre>pid_t child_pid;</pre>	pid_t digunakan untuk ID proses dan ID grup proses. child_pid adalah variable yang digunakan untuk menyimpan ID proses dari proses fork(), dimana hasilnya akan menjadi child process.
20	<pre>/* Handle SIGCHLD by calling clean_up_child_ process. */</pre>	Memberi komentar “Menangani SIGCHLD dengan memanggil clean_up_child_process”
21	<pre>struct sigaction sigchld_action;</pre>	Membuat sebuah struct sigaction dengan nama sigchld_action dimana struct tersebut berisi beberapa fungsi lainnya.
22	<pre>memset (&sigchld_actio n, 0, sizeof (sigchld_action));</pre>	Memset atau bisa disebut memory setter adalah sebuah fungsi untuk mengisi memory yang telah dipesan dengan nilai tertentu yang dimasukkan pada

		parameter keduanya agar memory yang dipesan tidak bernilai undefined.
23	<pre>sigchld_action. sa_handler = &clean_up_child _process;</pre>	Memanggil sebuah fungsi <code>clean_up_child_process</code> kemudian diassign ke <code>sigchld_action.sa_handler</code>
24	<pre>sigaction (SIGCHLD, &sigchld_action , NULL);</pre>	Sigaction() digunakan untuk merubah aksi yang dilakukan oleh proses ketika menerima sebuah sinyal.
25	<pre>/* Create child */</pre>	Memberi komentar, untuk membuar child process.
26	<pre>child_pid = fork();</pre>	fork() digunakan untuk membuat proses baru yang disebut chld process, yang berjalan bersamaan dengan proses yang membuat panggilan <code>fork()</code> (parent process), Setelah child process dibuat, kedua proses akan menjalankan instruksi berikutnya mengikuti <code>fork()</code> system call. Pada baris ini, system call <code>fork</code> diassign pada <code>child_pid</code> , sehingga <code>child_pid</code> akan menyimpan child process.
27	<pre>if (child_pid > 0) {</pre>	Terdapat return value untuk menunjukkan apakah proses <code>fork()</code> berhasil atau tidak : Negative : pembuatan child process gagal. 0 : kembali ke child process yang baru dibuat. Positive : kembali ke parent process dan PID berisi ID proses dari child process yang dibuat. Pada baris ini, jika <code>child_pid</code> (child process) berhasil dibuat, maka perintah di bawahnya akan dijalankan.
28	<pre>/* Parent process */</pre>	Memberi komentar, parent process berjalan.

29	<code>sleep (60);</code>	<p>Sleep digunakan untuk membuat pekerjaan tiruan. Pekerjaan tiruan membantu memnunda eksekusi. Perintah ini menjeda eksekusi untuk sejumlah waktu yang ditentukan.</p> <p>Pada baris ini, eksekusi ditunda selama 60 detik.</p>
30	<code>}</code>	Syntax untuk menutup kondisi pertama.
31	<code>else {</code>	<p>If else digunakan untuk memberi kondisi.</p> <p>Syntax untuk membuka kondisi kedua.</p>
32	<code>/* Child process. Exit immediately.*/</code>	Memberi komentar, child process berjalan dan akan segera keluar.
33	<code>exit (0);</code>	<p>Exit digunakan untuk keluar dari shell dimana ia sedang berjalan.</p> <p>Pada baris ini, proses akan keluar dan mengembalikan nilai 0.</p>
34	<code>}</code>	Syntax untuk menutup kondisi kedua.
35	<code>return 0;</code>	<p>Return digunakan untuk keluar dari fungsi shell.</p> <p>Pada baris ini, proses akan keluar dari fungsi dan mengembalikan nilai 0.</p>
36	<code>}</code>	Syntax untuk menutup deklarasi fungsi main().

```

rheza@rheza-VirtualBox:~$ g++ -o zombie_safe zombie_safe.cpp
rheza@rheza-VirtualBox:~$ ./zombie_safe
rheza@rheza-VirtualBox:~$ ps -e -o pid,ppid,status,cmd
  PID  PPID  STATUS  CMD
    1     0      -  /sbin/init splash
    2     0      -  [kthreadd]
    3     2      -  [rcu_gp]
    4     2      -  [rcu_par_gp]
    6     2      -  [kworker/0:0H-kb]
    8     2      -  [mm_percpu_wq]
    9     2      -  [ksoftirqd/0]
   10     2      -  [rcu_sched]
   11     2      -  [migration/0]
   12     2      -  [idle_inject/0]
   13     2      -  [kworker/0:1-mm_]
   14     2      -  [cpuhp/0]
   15     2      -  [kdevtmpfs]
   16     2      -  [netns]
   17     2      -  [rcu_tasks_kthre]
   18     2      -  [kauditd]

```

Analisa : Pada percobaan di atas, metode zombiesafe bisa digunakan untuk menghindari proses menjadi zombie yaitu dengan menyimpan exit status dari child process ke dalam variabel global dan memberikan perubahan status di dalam fungsi. Lalu, perubahan status pada child process di kirim menggunakan signal dan diterima oleh variabel penampung yang ada didalam main. Sehingga proses zombie dapat di hindari dengan ditandai nya parent proses tidak terjadi sleep.