

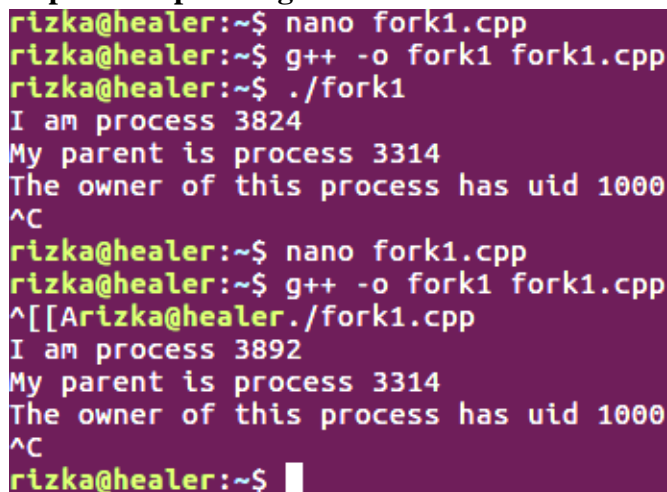
Percobaan 1 : Melihat proses parent dan proses child

1. Dengan menggunakan editor vi, buatlah file fork1.cpp dan ketikkan program berikut :

```
#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
/* getpid() adalah system call yg dideklarasikan pada unistd.h.
Menghasilkan suatu nilai dengan type pid_t.
pid_t adalah type khusus untuk process id yg ekuivalen dg int
*/
int main(void) {
    pid_t mypid;
    uid_t myuid;
    for (int i = 0; i < 3; i++) {
        mypid = getpid();
        cout << "I am process " << mypid << endl;
        cout << "My parent is process " << getppid() << endl;
        cout << "The owner of this process has uid " << getuid() << endl;
        /* sleep adalah system call atau fungsi library yang menghentikan proses ini
        dalam detik
        */
        sleep(30);
    }
    return 0;
}
```

2. Jalankan Program

- Capture Ouput Program



```
rizka@healer:~$ nano fork1.cpp
rizka@healer:~$ g++ -o fork1 fork1.cpp
rizka@healer:~$ ./fork1
I am process 3824
My parent is process 3314
The owner of this process has uid 1000
^C
rizka@healer:~$ nano fork1.cpp
rizka@healer:~$ g++ -o fork1 fork1.cpp
^[[Arizka@healer:~$ ./fork1.cpp
I am process 3892
My parent is process 3314
The owner of this process has uid 1000
^C
rizka@healer:~$
```

- Analisis :

Dari hasil program, program digunakan untuk melihat proses parent dan child dimana untuk melihat PID parent menggunakan fungsi `getppid()`, sedangkan untuk

melihat PID dari proses child menggunakan *mypid* dimana *mypid* ini sudah di isi nilai dengan fungsi *getpid()* dan juga didapatkan dengan deklarasi dengan *pid_t mypid*;. Untuk UID menggunakan fungsi *getuid()*, dengan mendeklarasikan terlebih dahulu *uid_t myuid*. Dengan Program tersebut dan fungsi – fungsi yang digunakan diperoleh PID dan UID yang digunakan oleh proses yang mengeksekusi suatu job.

3. Tambahkan Baris :

Tambahkan baris

```
mypid = getpid();
```

```
cout << "I am process " << mypid << endl;
```

- **Analisis :**

Setelah ditambahkan sebuah baris pada program dibawah return maka akan terjadi perubahan PID yang di gunakan oleh child namun PID proses tidak ada perubahan.

4. Jawab Pertanyaan:

- PID Proses adalah 3824.
PID Proses sesudahnya di tambahkan baris adalah 3892
- PID parent adalah 3314
- PID parent sesudahnya di tambahkan baris adalah 3314
- UID yang dibangkitkan adalah 1000
- Parent tersebut adalah proses dengan PID 3314

Percobaan 2 : Membuat dua proses terus menerus dengan sebuah system call fork()

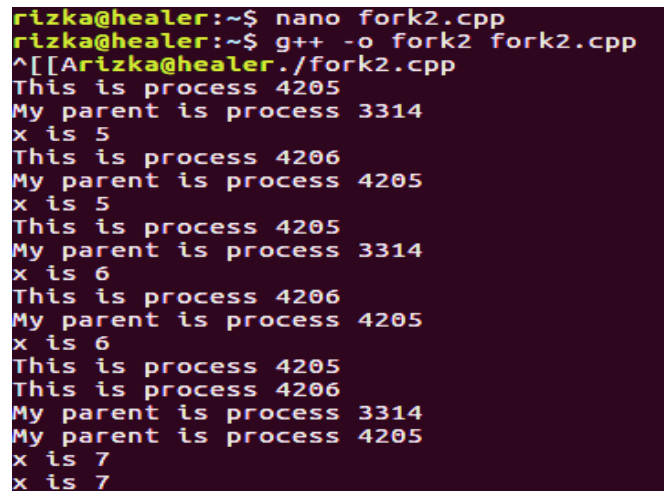
1. Dengan menggunakan editor vi, buatlah file *fork2.cpp* dan ketikkan program berikut :

```
#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
/* getpid() dan fork() adalah system call yg dideklarasikan pada unistd.h.
Menghasilkan suatu nilai dengan type pid_t. pid_t adalah type khusus untuk
process id yg ekuivalen dg int */

int main(void) {
    pid_t childpid;
    int x = 5;
    childpid = fork();
    while (1) {
        cout << "This is process " << getpid() << endl;
        cout << "x is " << x << endl;
        sleep(10);
        x++;
    }
    return 0;
}
```

2. Jalankan Program

- Capture Ouput Program



```
rizka@healer:~$ nano fork2.cpp
rizka@healer:~$ g++ -o fork2 fork2.cpp
^[[Arizka@healer:~/fork2.cpp
This is process 4205
My parent is process 3314
x is 5
This is process 4206
My parent is process 4205
x is 5
This is process 4205
My parent is process 3314
x is 6
This is process 4206
My parent is process 4205
x is 6
This is process 4205
This is process 4206
My parent is process 3314
My parent is process 4205
x is 7
x is 7
```

- Analisis

Fork2 Membuat dua Proses Terus menerus Dengan System Call Fork() dari program di dapatkan hasil output diatas bahwa program akan membuat dua proses yang tidak akan berhenti dan hanya dapat diberhentikan dengan ctrl-C. Program tersebut membuat program membuat dua child/proses. getpid() dan fork() adalah system call yg dideklarasikan pada unistd.h. Menghasilkan suatu nilai dengan type pid_t. pid_t adalah type khusus untuk process id yang ekuivalen dengan int, ha ini akan menghasilkan nomor PID dengan getppid untuk child dan UID serta getppid untuk parent,

3. Jawab Pertanyaan :

- a) Dua Proses yang dibangkitkan namun tidak bisa berhenti setelah menekan Ctrl-C, akan bias berhenti.
- b) PID child yang dibangkitkan pertama adalah 4205
PID child yang dibangkitkan kedua adalah 4206
PID child akan terus di tambah 1 jika dibangkitkan kembali.
PID Parent adalah 3314
- c) Child dan Parent tersebut akan dibangkitkan sebanyak diberhentikan menggunakan Ctrl-C

Percobaan 3 : Membuat dua proses sebanyak lima kali

1. Dengan menggunakan editor vi, buatlah file fork1.cpp dan ketikkan program berikut :

```
#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
/* getpid() dan fork() adalah system call yg dideklarasikan pada unistd.h.
Menghasilkan suatu nilai dengan type pid_t. pid_t adalah type khusus untuk process
id yg ekuivalen dg int */
int main(void) {
    pid_t childpid;
```

```
childpid = fork();
for (int i=0; i<5; i++) {
cout << "This is process " << getpid() << endl;
sleep(2);
}
return 0;
```

2. Jalankan Program

- Capture Ouput Program

```
rizka@healer:~$ nano fork3.cpp
rizka@healer:~$ g++ -o fork3 fork3.cpp
rizka@healer:~$ ./fork3
This is process 2882
This is process 2883
This is process 2882
This is process 2883
This is process 2882
This is process 2883
This is process 2883
This is process 2882
This is process 2883
This is process 2882
rizka@healer:~$
```

- Analisis :

Program akan membuat dua proses sebanyak 5 kali, jadi proses akan dibuat menjadi dua dan dibangkitkan selama lima kali dengan looping *for* (*int i=0; i<5; i++*), Untuk mengetahui berapa nilai PID yang telah dibuat oleh proses maka menggunakan fungsi *getpid()*.

3. Jawab Pertanyaan:

- a. Proses yang dibangkitkan adalah 10 proses
- b. PID child yang dibangkitkan adalah proses 1 2882, proses 2 2883
- c. Child dan Parent tersebut akan di bangkitkan selama 5 kali.

Tugas : Child adalah parent dari proses berikut-berikutnya

```
root@healer:/home/rizka# nano fork3tugas.cpp
root@healer:/home/rizka# g++ -o fork3tugas fork3tugas.cpp
root@healer:/home/rizka# ./fork3tugas
This is process 2401This is Parent2383
This is process 2402This is Parent2401
This is process 2401This is Parent2383
This is process 2402This is Parent2401
root@healer:/home/rizka#
```

Tugas : Membentuk 4 buah child dari 1 parent

```
root@healer:/home/rizka# nano fork3tugas.cpp
root@healer:/home/rizka# g++ -o fork3tugas fork3tugas.cpp
root@healer:/home/rizka# ./fork3tugas
This is process 2602This is Parent2338
This is process 2603This is Parent2602
This is process 2602This is Parent2338
This is process 2603This is Parent2602
```

Percobaan 4 : Proses parent menunggu sinyal dari proses child dengan system call wait

1. Dengan menggunakan editor vi, buatlah file fork4.cpp dan ketikkan program berikut :

```
#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk
process id yg ekuivalen dg int */
int main(void) {
    pid_t child_pid;
    int status;
    pid_t wait_result;
    child_pid = fork();
    if (child_pid == 0) {
        /* kode ini hanya dieksekusi proses child */
        cout << "I am a child and my pid = " << getpid() << endl;
        cout << "My parent is " << getppid() << endl;
        /* keluar if akan menghentikan hanya proses child */
    } else if (child_pid > 0) {
        /* kode ini hanya mengeksekusi proses parent */
        cout << "I am the parent and my pid = " << getpid() << endl;
        cout << "My child has pid = " << child_pid << endl;
    } else {
        cout << "The fork system call failed to create a new process" << endl;
        exit(1);
    }
    /* kode ini dieksekusi baik oleh proses parent dan child */
    cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
    if (child_pid == 0) {
        /* kode ini hanya dieksekusi oleh proses child */
        cout << "I am a child and I am quitting work now!" << endl;
    } else {
        /* kode ini hanya dieksekusi oleh proses parent */
        cout << "I am a parent and I am going to wait for my child" << endl;
        do {
            /* parent menunggu sinyal SIGCHLD mengirim tanda bahwa proses child
determinasi */
```

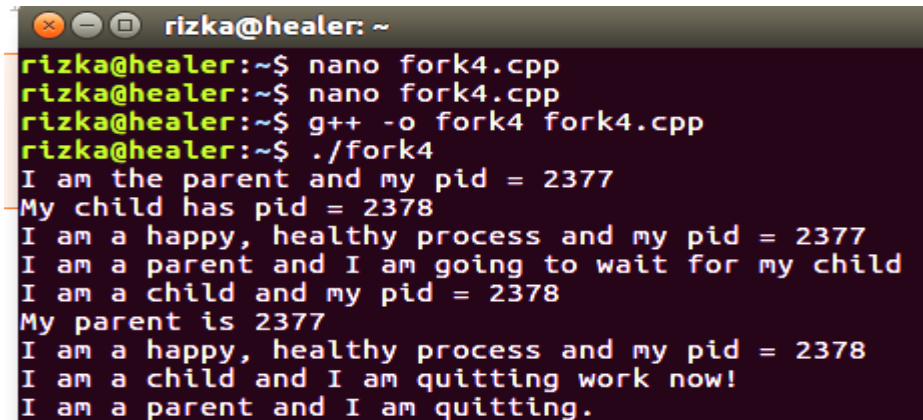
```

wait_result = wait(&status);
}
while (wait_result != child_pid);
cout << "I am a parent and I am quitting." << endl;
}
return 0;
}

```

2. Jalankan Program

- Capture Output Program



```

rizka@healer:~$ nano fork4.cpp
rizka@healer:~$ nano fork4.cpp
rizka@healer:~$ g++ -o fork4 fork4.cpp
rizka@healer:~$ ./fork4
I am the parent and my pid = 2377
My child has pid = 2378
I am a happy, healthy process and my pid = 2377
I am a parent and I am going to wait for my child
I am a child and my pid = 2378
My parent is 2377
I am a happy, healthy process and my pid = 2378
I am a child and I am quitting work now!
I am a parent and I am quitting.

```

- Analisis :

Program diatas untuk melihat bagaimana Proses parent menunggu sinyal dari proses child dengan system call wait . Proses parent menunggu sinyal dari proses childnya dimana parent menunggu sinyal SIGCHLD mengirim tanda bahwa proses child diterminasi dengan mengirimkan hasil dari fungsi wait (&status) dengan parameter pointer `wait_result = wait(&status)`. Hal ini akan memberikan status jika child sedang proses determinasi jika sudah terminas (mengakhiri proses) maka parent tidak menunggu childnya lagi karena childnya sudah mati.

3. Jawab Pertanyaan:

- Proses yang dibangkitkan adalah 1 proses
- PID child yang di bangkitkan adalah 2378 dan PID parent yang dibangkitkan adalah 2377
- Child dan Parent yang dibangkitkan sebanyak 1 kali.

Percobaan 5 :Menggunakan fork/exec dan wait System call fork/exec dan wait mengeksekusi program bernama ls, menggunakan file executable /bin/ls dengan parameter -l (ls -l)

1. Dengan menggunakan editor yang ada, buatlah file fork5.cpp dan ketikkan program berikut :

```

#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

```

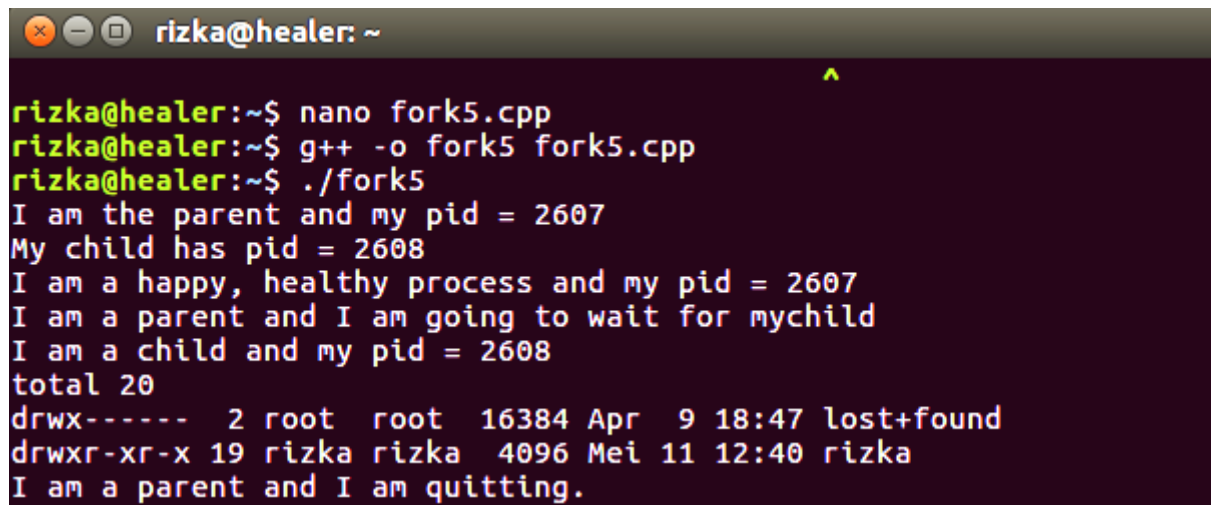
```

#include <stdlib.h>
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk
process id
yg ekuivalen dg int */
int main(void) {
pid_t child_pid;
int status;
pid_t wait_result;
child_pid = fork();
if (child_pid == 0) {
/* kode ini hanya dieksekusi proses child */
cout << "I am a child and my pid = " << getpid() << endl;
execl("/bin/ls", "ls", "-l", "/home", NULL);
/* jika execl berhasil kode ini tidak pernah digunakan */
cout << "Could not execl file /bin/ls" << endl;
exit(1);
/* exit menghentikan hanya proses child */
}
else if (child_pid > 0) {
/* kode ini hanya mengeksekusi proses parent */
cout << "I am the parent and my pid = " << getpid() << endl;
cout << "My child has pid = " << child_pid << endl;
}
else {
cout << "The fork system call failed to create a new process" << endl;
exit(1);
}
/* kode ini hanya dieksekusi oleh proses parent karena
child mengeksekusi dari "/bin/ls" atau keluar */
cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
if (child_pid == 0) {
/* kode ini tidak pernah dieksekusi */
printf("This code will never be executed!\n");
}
else {
/* kode ini hanya dieksekusi oleh proses parent */
cout << "I am a parent and I am going to wait for my child" << endl;
do {
/* parent menunggu sinyal SIGCHLD mengirim tanda bila proses child
diterminasi*/
wait_result = wait(&status);
}
while (wait_result != child_pid);
cout << "I am a parent and I am quitting." << endl;
}
return 0;
}

```

2. Jalankan Program

- Capture Output Program



```
rizka@healer: ~  
rizka@healer:~$ nano fork5.cpp  
rizka@healer:~$ g++ -o fork5 fork5.cpp  
rizka@healer:~$ ./fork5  
I am the parent and my pid = 2607  
My child has pid = 2608  
I am a happy, healthy process and my pid = 2607  
I am a parent and I am going to wait for mychild  
I am a child and my pid = 2608  
total 20  
drwx----- 2 root root 16384 Apr  9 18:47 lost+found  
drwxr-xr-x 19 rizka rizka 4096 Mei 11 12:40 rizka  
I am a parent and I am quitting.
```

- Analisis :

Untuk membuat perintah juga bisa melalui program dengan menggunakan fork/exec dan wait System call fork/execl dimana wait mengeksekusi perintah bernama ls yang ada dalam fungsi execl yang menggunakan file executable /bin/ls dengan parameter -l (ls -l) yang disisipkan pada program diatas.

Percobaan 6 : System call fork/exec dan wait mengeksekusi program lain

1. Dengan menggunakan editor vi, buatlah file fork6.cpp dan ketikkan program berikut :

```
#include <iostream>  
using namespace std;  
#include <sys/types.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <stdlib.h>  
#include <stdio.h>  
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk process id yg  
ekuivalen dg int  
*/  
int main(void) {  
    pid_t child_pid;  
    int status;  
    pid_t wait_result;  
    child_pid = fork();  
    if (child_pid == 0) {  
        /* kode ini hanya dieksekusi proses child */  
        cout << "I am a child and my pid = " << getpid() << endl;  
        execl("fork3", "goose", NULL);  
        /* jika execl berhasil kode ini tidak pernah digunakan */  
        cout << "Could not execl file fork3" << endl;  
    }
```



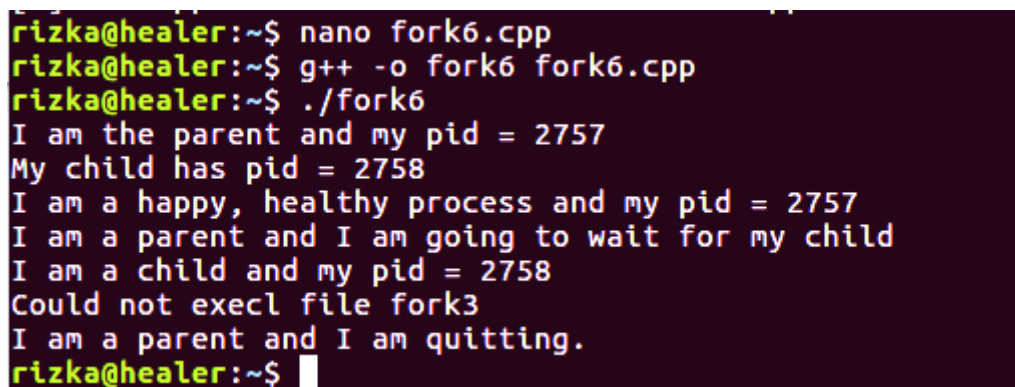
```

exit(1);
/* exit menghentikan hanya proses child */
}
else if (child_pid > 0) {
/* kode ini hanya mengeksekusi proses parent */
cout << "I am the parent and my pid = " << getpid() << endl;
cout << "My child has pid = " << child_pid << endl;
}
else {
cout << "The fork system call failed to create a new process" << endl;
exit(1);
}
/* kode ini hanya dieksekusi oleh proses parent karena child mengeksekusi dari "fork3"
atau keluar */
cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
if (child_pid == 0) {
/* kode ini tidak pernah dieksekusi */
printf("This code will never be executed!\n");
}
else {
/* kode ini hanya dieksekusi oleh proses parent */
cout << "I am a parent and I am going to wait for my child" << endl;
do {
/* parent menunggu sinyal SIGCHLD mengirim tanda bila proses child diterminasi*/
wait_result = wait(&status);
}
while (wait_result != child_pid);
cout << "I am a parent and I am quitting." << endl;
}
return 0;
}

```

2. Jalankan Program

- Capture Ouput Program



```

rizka@healer:~$ nano fork6.cpp
rizka@healer:~$ g++ -o fork6 fork6.cpp
rizka@healer:~$ ./fork6
I am the parent and my pid = 2757
My child has pid = 2758
I am a happy, healthy process and my pid = 2757
I am a parent and I am going to wait for my child
I am a child and my pid = 2758
Could not execl file fork3
I am a parent and I am quitting.
rizka@healer:~$

```

- Analisis :

Pada program digunakan untuk membuka file dan menjalankan/ membuka isi file tersebut menggunakan System call fork/exec dan wait mengeksekusi dimana hal

ini program akan langsung dieksekusi melalui child yang membangkitkan dengan fork() sehingga juga dapat diketahui berapa PID yang digunakan parent dan child untuk memproses file tersebut dan di peroleh PID parent 2757, PID child 2758 namun program tidak dapat memproses fork3() sehingga parent langsung keluar dari proses.

LATIHAN:

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

a. ls -al /etc.

- **Program**

```
#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk
process id
yg ekuivalen dg int */
int main(void) {
    pid_t child_pid;
    int status;
    pid_t wait_result;
    child_pid = fork();
    if (child_pid == 0) {
        /* kode ini hanya dieksekusi proses child */
        cout << "I am a child and my pid = " << getpid() << endl;
```

```

execl("/bin/ls", "ls", "-al", "/etc", NULL);
/* jika execl berhasil kode ini tidak pernah digunakan */
cout << "Could not execl file /bin/ls" << endl;
exit(1);
/* exit menghentikan hanya proses child */
}
else if (child_pid > 0) {
/* kode ini hanya mengeksekusi proses parent */
cout << "I am the parent and my pid = " << getpid() << endl;
cout << "My child has pid = " << child_pid << endl;
}
else {
cout << "The fork system call failed to create a new process" << endl;
exit(1);
}
/* kode ini hanya dieksekusi oleh proses parent karena
child mengeksekusi dari "/bin/ls" atau keluar */
cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
if (child_pid == 0) {
/* kode ini tidak pernah dieksekusi */
printf("This code will never be executed!\n");
}
else {
/* kode ini hanya dieksekusi oleh proses parent */
cout << "I am a parent and I am going to wait for my child" << endl;
do {
/* parent menunggu sinyal SIGCHLD mengirim tanda bila proses child
determinasi*/
wait_result = wait(&status);
}
while (wait_result != child_pid);
cout << "I am a parent and I am quitting." << endl;
}
return 0;
}

```

- **Capture**

```
rizka@healer: ~  
rizka@healer:~$ nano fork5.cpp  
rizka@healer:~$ g++ -o fork5 fork5.cpp  
^[[Arizka@healer:~/fork5  
I am the parent and my pid = 3094  
My child has pid = 3095  
I am a happy, healthy process and my pid = 3094  
I am a parent and I am going to wait for mychild  
I am a child and my pid = 3095  
total 1152  
drwxr-xr-x 130 root root 12288 Mei 4 16:11 .  
drwxr-xr-x 24 root root 4096 Apr 9 18:52 ..  
drwxr-xr-x 3 root root 4096 Feb 16 03:34 acpi  
-rw-r--r-- 1 root root 3028 Feb 16 03:19 adduser.conf  
drwxr-xr-x 2 root root 4096 Apr 9 18:56 alternatives  
-rw-r--r-- 1 root root 401 Dec 29 2014 anacrontab  
-rw-r--r-- 1 root root 112 Jan 10 2014 apg.conf  
drwxr-xr-x 6 root root 4096 Feb 16 03:25 apm  
drwxr-xr-x 3 root root 4096 Feb 16 03:34 apparmor  
drwxr-xr-x 8 root root 4096 Feb 16 03:36 apparmor.d  
drwxr-xr-x 5 root root 4096 Feb 16 03:34 appport  
-rw-r--r-- 1 root root 389 Apr 18 2016 appstream.conf  
drwxr-xr-x 6 root root 4096 Apr 9 18:58 apt  
drwxr-xr-x 3 root root 4096 Feb 16 03:30 aptdaemon  
-rw-r--r-- 1 root root 4942 Jun 14 2016 wgetrc  
drwxr-xr-x 2 root root 4096 Feb 16 03:34 wpa_supplicant  
drwxr-xr-x 11 root root 4096 Feb 16 03:34 X11  
drwxr-xr-x 5 root root 4096 Feb 16 03:36 xdg  
drwxr-xr-x 2 root root 4096 Feb 16 03:34 xml  
-rw-r--r-- 1 root root 477 Jul 20 2015 zsh_command_not_found  
I am a parent and I am quitting.
```

- **Aalisis**
Untuk membuat perintah juga bisa melalui program dengan menggunakan fork/exec dan wait System call fork/execl dimana wait mengeksekusi perintah bernama ls yang ada dalam fungsi execl yang menggunakan file executable /bin/ls dengan parameter argument -al (ls -al) yang disisipkan pada program diatas, sehingga jika memenuhi kondisi maka child akan memproses execl tersebut dan menampilkan isi dari file direktori /etc.

b. cat fork2

- **Program**

```
#include <iostream>  
using namespace std;  
#include <sys/types.h>  
#include <unistd.h>  
#include <sys/wait.h>  
#include <stdlib.h>  
#include <stdio.h>  
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk  
process id yg
```

```

ekuivalen dg int
*/
int main(void) {
    pid_t child_pid;
    int status;
    pid_t wait_result;
    child_pid = fork();
    if (child_pid == 0) {
        /* kode ini hanya dieksekusi proses child */
        cout << "I am a child and my pid = " << getpid() << endl;
        execl("fork2", "cat", NULL);
        /* jika execl berhasil kode ini tidak pernah digunakan */
        cout << "Could not execl file fork2" << endl;
        exit(1);
        /* exit menghentikan hanya proses child */
    }
    else if (child_pid > 0) {
        /* kode ini hanya mengeksekusi proses parent */
        cout << "I am the parent and my pid = " << getpid() << endl;
        cout << "My child has pid = " << child_pid << endl;
    }
    else {
        cout << "The fork system call failed to create a new process" << endl;
        exit(1);
    }
    /* kode ini hanya dieksekusi oleh proses parent karena child mengeksekusi dari
    "fork2"
    atau keluar */
    cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
    if (child_pid == 0) {
        /* kode ini tidak pernah dieksekusi */
        printf("This code will never be executed!\n");
    }
    else {
        /* kode ini hanya dieksekusi oleh proses parent */
        cout << "I am a parent and I am going to wait for my child" << endl;
        do {
            /* parent menunggu sinyal SIGCHLD mengirim tanda bila proses child
            diterminasi*/
            wait_result = wait(&status);
        }
        while (wait_result != child_pid);
        cout << "I am a parent and I am quitting." << endl;
    }
    return 0;
}

```

- **Capture**

```

I am a parent and I am quitting.
rizka@healer:~$ nano fork5.cpp
rizka@healer:~$ g++ -o fork5 fork5.cpp
^[[Arizka@healer:~/fork5cpp
I am the parent and my pid = 3368
My child has pid = 3369
I am a happy, healthy process and my pid = 3368
I am a parent and I am going to wait for mychild
I am a child and my pid = 3369
lost+found rizka
I am a parent and I am quitting.
rizka@healer:~$

```

- **Analisis**

Pada program digunakan untuk membuka file dan menjalankan/ membuka isi file tersebut menggunakan System call fork/exec dan wait mengeksekusi dimana hal ini program akan langsung dieksekusi melalui child yang membangkitkan dengan fork() sehingga juga dapat diketahui berapa PID yang digunakan parent dan child untuk memproses file tersebut dan di peroleh PID parent 3368, PID child 3369 namun program tidak dapat memproses fork2() sehingga parent langsung keluar dari proses dan tidak menampilkan isi dari proses karena tidak memenuhi kondisi dank arena execl tidak berhasil di eksekusi.

c. ./fork2

- **Program**

```

#include <iostream>
using namespace std;
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <stdlib.h>
/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk
process id
yg ekuivalen dg int */
int main(void) {
    pid_t child_pid;
    int status;
    pid_t wait_result;
    child_pid = fork();
    if (child_pid == 0) {
        /* kode ini hanya dieksekusi proses child */
        cout << "I am a child and my pid = " << getpid() << endl;
        execl("/bin/ls", "./fork2", "-l", "/home", NULL);
        /* jika execl berhasil kode ini tidak pernah digunakan */
        cout << "Could not execl file /bin/ls" << endl;
        exit(1);
        /* exit menghentikan hanya proses child */
    }
    else if (child_pid > 0) {
        /* kode ini hanya mengeksekusi proses parent */
        \\cout << "I am the parent and my pid = " << getpid() << endl;
    }
}

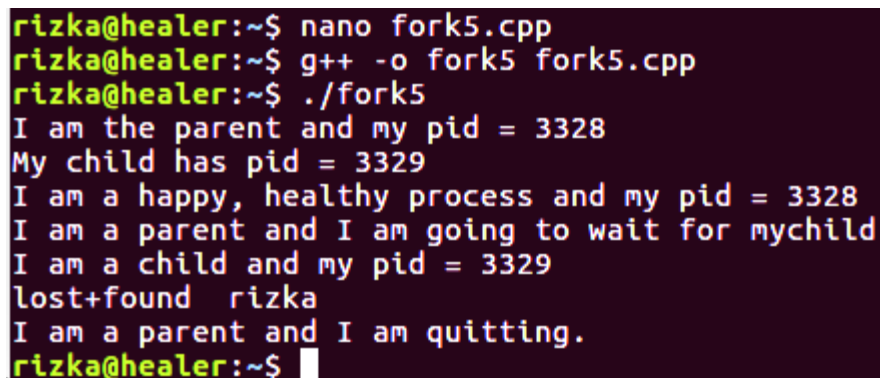
```

```

cout << "My child has pid = " << child_pid << endl;
}
else {
cout << "The fork system call failed to create a new process" << endl;
exit(1);
}
/* kode ini hanya dieksekusi oleh proses parent karena
child mengeksekusi dari "/bin/ls" atau keluar */
cout << "I am a happy, healthy process and my pid = " << getpid() << endl;
if (child_pid == 0) {
/* kode ini tidak pernah dieksekusi */
printf("This code will never be executed!\n");
}
else {
/* kode ini hanya dieksekusi oleh proses parent */
cout << "I am a parent and I am going to wait for my child" << endl;
do {
/* parent menunggu sinyal SIGCHLD mengirim tanda bila proses child
diterminasi*/
wait_result = wait(&status);
}
while (wait_result != child_pid);
cout << "I am a parent and I am quitting." << endl;
}
return 0;
}

```

- **Capture**



```

rizka@healer:~$ nano fork5.cpp
rizka@healer:~$ g++ -o fork5 fork5.cpp
rizka@healer:~$ ./fork5
I am the parent and my pid = 3328
My child has pid = 3329
I am a happy, healthy process and my pid = 3328
I am a parent and I am going to wait for mychild
I am a child and my pid = 3329
lost+found rizka
I am a parent and I am quitting.
rizka@healer:~$

```

- **Analisis**

Pada program digunakan untuk membuka file dan menjalankan/ membuka isi file tersebut menggunakan System call fork/exec dan wait mengeksekusi dimana hal ini program akan langsung dieksekusi melalui child yang dibangkitkan dengan fork() sehingga juga dapat diketahui berapa PID yang digunakan parent dan child untuk memproses file tersebut dan di peroleh PID parent 3328, PID child 3329 namun program tidak dapat memproses perintah ./fork2() sehingga parent langsung keluar dari proses dan tidak dapat mengeksekusi file fork2() untuk mengetahui hasilnya..

