

Unix System Call dan Manajemen Memori

Nama: Rizqi Bagus Andrean

Kelas: TI-1D

Absen: 25

Pokok Bahasan:

- Unix system call
- Manajemen memori

Tujuan Belajar

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

- Menggunakan system call fork, wait dan exec pada linux
- Menggunakan perintah-perintah untuk manajemen memori

Dasar Teori

1. Unix System Call

Pada praktikum ini akan dilakukan percobaan menggunakan system call yang berhubungan dengan proses pada system operasi UNIX yang biasa disebut UNIX System Call, yaitu system call fork, exec dan wait. Pada percobaan yang dilakukan akan dibuat program yang didalamnya terdapat fungsi system call. Untuk menjalankannya pada Linux gunakan g++.

System Call Fork

System call fork adalah suatu system call yang membuat suatu proses baru pada system operasi UNIX. Pada percobaan ini menggunakan mesin Linux dan beberapa program yang berisi system call fork().

Bila suatu program berisi sebuah fungsi fork(), eksekusi dari program menghasilkan eksekusi dua proses. Satu proses dibuat untuk memulai eksekusi program. Bila system call fork() dieksekusi, proses lain dibuat. Proses asal disebut proses parent dan proses kedua disebut proses child. Proses child merupakan duplikat dari proses parent. Kedua proses melanjutkan eksekusi dari titik dimana system call fork() menghasilkan eksekusi pada program utama. Karena UNIX adalah system operasi time sharing, dua proses tersebut dapat mengeksekusi secara konkuen.

Nilai yang dihasilkan oleh fork() disimpan dalam variabel bertipe pid_t, yang berupa nilai integer. Karena nilai dari variabel ini tidak digunakan, maka hasil fork() dapat diabaikan.

- Untuk kill proses gunakan Ctrl+C.
- Untuk dokumentasi fork() dapat dilihat dengan ketikkan man 2 fork.
- Untuk melihat id dari proses, gunakan system call getpid().
- Untuk melihat dokumentasi dari getpid(), ketikkan man 2 getpid.

Perbedaan antara proses parent dan proses child adalah

- Mempunyai pid yang berbeda.
- Pada proses parent, `fork()` menghasilkan pid dari proses child jika sebuah proses child dibuat.
- Pada proses child, `fork()` selalu menghasilkan 0. Membedakan copy dari semua data, termasuk variabel dengan current value dan stack.
- Membedakan program counter (PC) yang menunjukkan eksekusi berikutnya meskipun awalnya keduanya mempunyai nilai yang sama tetapi setelah itu berbeda.
- Setelah `fork`, kedua proses tersebut tidak menggunakan variabel bersama.

System call `fork` menghasilkan :

- Pid proses child yang baru ke proses parent, hal ini sama dengan memberitahukan proses parent nama dari child-nya
- 0 : menunjukkan proses child
- -1 : 1 jika terjadi error, `fork()` gagal karena proses baru tidak dapat dibuat.

System Call Wait

System call wait menyebabkan proses menunggu sinyal (menunggu sampai sembarang tipe sinyal diterima dari sembarang proses). Biasanya digunakan oleh proses parent untuk menunggu sinyal dari system operasi ke parent bila child dieliminasi. System call wait menghasilkan pid dari proses yang mengiklaimi sinyal. Untuk melihat dokumentasi wait gunakan perintah `man 2 wait`.

System Call `execl`

Misalnya kita ingin proses baru menjalankan sesuatu yang berbeda dari proses parent, sebutlah menjalankan program yang berbeda. Sistem call `execl` meletakkan program executable baru ke memori dan mengasosiasikannya dengan proses saat itu. Dengan kata lain, mengubah segala sesuatunya sehingga program mulai mengeksekusi dari file yang berbeda.

2. MANAJEMEN MEMORY

Linux mengimplementasikan sistem virtual memori demand-paged. Proses mempunyai besar memori virtual yang besar (4 gigabyte). Pada virtual memori dilakukan transfer page antara disk dan memori fisik.

Jika tidak terdapat cukup memori fisik, kernel melakukan swapping beberapa page lama ke disk. Disk drive adalah perangkat mekanik yang membaca dan menulis ke disk yang lebih lambat dibandingkan mengakses memori fisik. Jika memori total page lebih dari memori fisik yang tersedia, kernel lebih banyak melakukan swapping dibandingkan eksekusi kode program, sehingga terjadi thrashing dan mengurangi utilitas.

Jika memori fisik ekstra tidak digunakan, kernel meletakkan kode program sebagai disk buffer cache. Disk buffer menyimpan data disk yang diakses di memori; jika data yang sama dibutuhkan lagi dapat dengan cepat diambil dari cache.

Pertama kali sistem melakukan booting, ROM BIOS membentuk memori test seperti terlihat berikut :

```
ROM BIOS (C) 1990
008192 KB OK WAIT.....
```

Kemudian informasi penting ditampilkan selama proses booting pada linux seperti terlihat berikut:

```
Memory: 7100k/8192k available (464k
kernel code, 384k reserved, 244k data) ...
Adding Swap: 19464k swap-space
```

Informasi diatas menampilkan jumlah RAM tersedia setelah kernel di-load ke memori (dalam hal ini 7100K dari 8192K). Jika ingin melihat pesan saat booting kernel yang terlalu cepat dibaca dapat dilihat kembali dengan perintah dmesg.

Setiap Linux dijalankan, perintah free digunakan untuk menampilkan total memori yang tersedia. Atau menggunakan cat /proc/meminfo. Memori fisik dan ruang swap ditampilkan disini. Contoh output pada sistem :

```
total used free shared buffers
Mem: 7096 52161880 2328 2800
Swap: 194640 19464
```

Informasi ditampilkan dalam kilobyte (1024 byte). Memori "total" adalah jumlah tersedia setelah load kernel. Memori digunakan untuk proses atau disk buffering sebagai "used". Memori yang sedang tidak digunakan ditampilkan pada kolom "free".

Memori total sama dengan jumlah kolom "used" dan "free". Memori diindikasikan "shared" yaitu berapa banyak memori yang digunakan lebih dari satu proses. Program seperti shell mempunyailebih dari satu proses yang berjalan. Kode executable read-only dan dapat disharing oleh semua proses yang berjalan pada shell. Kolom "buffer" menampilkan berapa banyak memori digunakan untuk disk buffering.

Perintah free juga menunjukkan dengan jelas bagaimana swap space dilakukan dan berapa banyak swapping yang terjadi. Percobaan berikut untuk mengetahui manajemen memori :

1. Pada saat bootup, dengan satu user log in, dengan perintah free sistem melaporkan berikut :

```
total      used      free     shared    buffers   cached
Mem:   247184    145772    101412         0      10872    57564
-/+ buffers/cache:  77336    169848
Swap:   522072         0     522072
```

Didapat free memori (4.4MB) dan sedikit disk buffer (1.1MB).

2. Situasi berubah setelah menjalankan perintah yang membaca data dari disk (command ls -lR /.)

```
total      used      free     shared    buffers   cached
Mem:   247184    230604    16580         0      45260    59748
-/+ buffers/cache: 125596    121588
Swap:   522072        308     522072
```

Disk buffer bertambah menjadi 2 MB. Hal ini akibat pula pada kolom "used" dan memori "free" juga berkurang.

Perintah top dan ps -u juga sangat berguna untuk menunjukkan bagaimana penggunaan memori berubah secara dinamis dan bagaimana proses individu menggunakan memori. Contoh tampilannya :

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
student	4581	0.0	0.3	4316	856	pts/0	S	10:25	0:00	bash
student	4699	0.0	0.2	2604	656	pts/0	R	10:39	0:00	ps -u

TUGAS PENDAHULUAN:

Jawablah pertanyaan-pertanyaan berikut ini:

1. Apa yang dimaksud dengan system call?

System call merupakan penyedia antarmuka dari pelayanan-pelayanan yang tersedia dengan system operasi. Umumnya system call menggunakan bahasa C dan C++, meskipun tugas-tugas seperti hardware yang harus diakses langsung, maka menggunakan bahasa assembly. Pada sistem operasi UNIX akan menggunakan UNIX system call yaitu call fork, exec, dan wait.

2. Apa yang dimaksud dengan system call fork(), exec() dan wait(). Jawablah dengan menggunakan perintah man (contoh : man 2 fork, man 2 exec dan man 2 wait)?

- System call fork adalah suatu system call yang membuat suatu proses baru pada system operasi UNIX.
- Sistem call exec meletakkan program executable baru ke memori dan mengasosiasikannya dengan proses saat itu. Dengan kata lain, mengubah segala sesuatunya sehingga program mulai mengeksekusi dari file yang berbeda.
- System call wait menyebabkan proses menunggu sinyal (menunggu sampai sembarang tipe sinyal diterima dari sembarang proses). Biasanya digunakan oleh prosesparent untuk menunggu sinyal dari system operasi ke parent bila child ditekminasi. System call wait menghasilkan pid dari proses yang mengirimkan sinyal. Untuk melihat dokumentasi wait gunakan perintah man 2 wait.

3. Apa yang dimaksud dengan virtual memori, proses swapping dan buffer cache pada manajemen memori?

Virtual memori adalah suatu teknik memisahkan antara memori logis dan memori fisiknya. Memori logis merupakan kumpulan keseluruhan halaman dari suatu program. Tanpa memori virtual, memori logis akan langsung dibawa ke memori fisik (memori utama). Disinilah memori virtual melakukan pemisahan dengan menaruh memori logis ke secondary storage (disk sekunder) dan hanya membawa halaman yang diperlukan ke memori utama (memori fisik). Swapping adalah manajemen memori dengan pemindahan proses antara memori utama dan disk selama eksekusi. Buffer cache dapat dianggap sebagai sumber daya memori, terutama sumber daya I/O karena penggunaannya dalam mediasi transfer.

4. Apa yang dimaksud dengan perintah free dan cat /proc/meminfo?

Free digunakan untuk mengetahui total memori yang digunakan dalam proses. Dalam

perintah free ditampilkan total kapasitas memori, memori yang terpakai, yang tidak sedang dipakai, yang dibagi, buffer, cache dan juga swap. Cat /proc/meminfo digunakan untuk mengetahui isi dari meminfo kemudian ditampilkan.

5. Apa yang dimaksud dengan perintah ps?

Perintah ps digunakan untuk menampilkan informasi proses yang sedang berjalan termasuk nomor PID dari proses tersebut.

PERCOBAAN:

1. Login sebagai user.
2. Bukalah Console Terminal dan lakukan percobaan-percobaan di bawah ini kemudian analisa hasil percobaan.
3. Selesaikan soal-soal latihan

Percobaan 1 : Melihat proses parent dan proses child

1. Dengan menggunakan editor vi, buatlah file foik1.cpp dan ketikkan program berikut :

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

/*
getpid() adalah system call yg dideklarasikan padaunistd.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(1);
    }
    return 0;
}
```

```

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

/* getpid() adalah system call yang dideklarasikan pada unistd.h. Menghasilkan
suatu nilai dengan type pid_t. pid_t adalah type khusus untuk process id yg eku
ivalen dgn 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 menghentik
an proses ini dalam detik
*/
        sleep(1);
    }
    return 0;
}
~
:wq

```

Analisis: Bila suatu program berisi sebuah fungsi fork(), eksekusi dari program akan menghasilkan eksekusi dua proses. Satu proses dibuat untuk memulai eksekusi program. Bila system call fork() dieksekusi, proses lain dibuat. Proses asal disebut proses parent dan proses kedua disebut proses child. Proses child merupakan duplikat dari proses parent. Kedua proses melanjutkan eksekusi dari titik dimana system call.

2. Gunakan g++ compiler untuk menjalankan program diatas

```

$ g++ -o fork1 fork1.cpp
$ ./fork1

```

```

root@bagusok:~/jobsheet8# nano fork1.cpp
root@bagusok:~/jobsheet8# rm fork1*
root@bagusok:~/jobsheet8# nano fork1.cpp
root@bagusok:~/jobsheet8# g++ -o fork1 fork1.cpp
root@bagusok:~/jobsheet8# ./fork1
I am process 18459
My parent is process 2020
The owner of this process has uid 0
I am process 18459
My parent is process 2020
The owner of this process has uid 0
I am process 18459
My parent is process 2020
The owner of this process has uid 0
root@bagusok:~/jobsheet8#

```

3. Amati output yang dihasilkan

Analisis: Setelah script program file foik1.cpp telah dibuat maka untuk menjalankannya menggunakan g++ compiler. Tahap jika pada laptop g++ belum terinstall, maka harus melakukan penginstallan terlebih dahulu dengan ketikkan pada terminal sudo apt-get install g++.

Peicobaan 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
padaunistd.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(1);
        x++;
    }
    return 0;
}
```



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

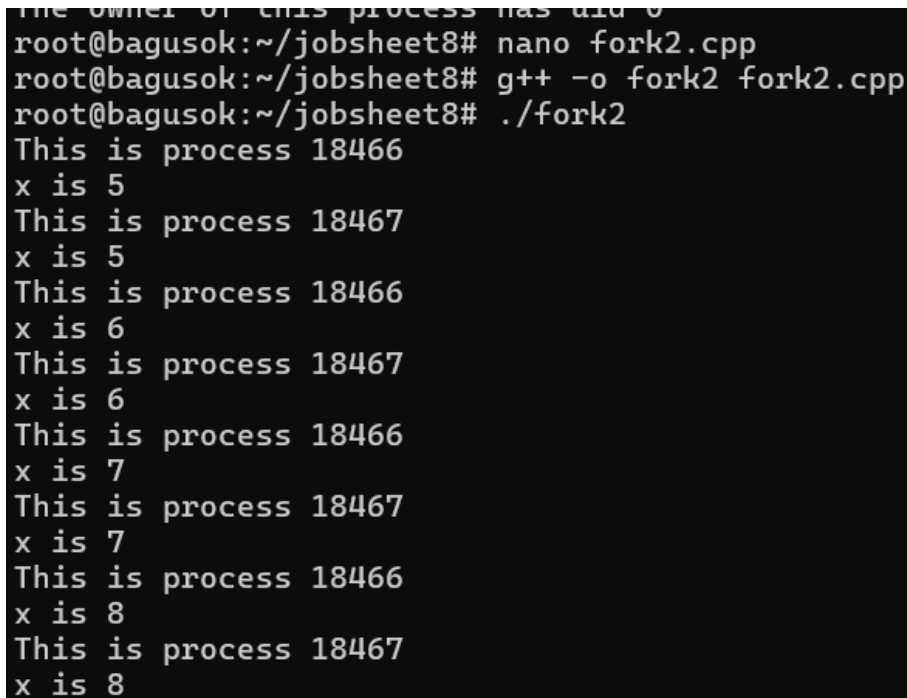
/* getpid() dan fork() adalah system call yang dideklarasikan pada unistd.h. Me
nghasilkan suatu nilai dengan type pid_t. pid_t adalah type khusus untuk proces
s id yg ekuivalen dgn int
*/

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(1);
        x++;
    }
    return 0;
}
~
~
~
~
~
~
:wq
```


Analisa: Menuliskan script seperti diatas pada program fork2.cpp yang telah dibuat dengan editor vi. System call fork adalah suatu system call yang membuat suatu proses baru pada system operasi UNIX.

2. Gunakan g++ compiler untuk menjalankan program diatas. Pada saat dijalankan, program tidak akan pernah berhenti. Untuk menghentikan program tekan Ctrl+C.

```
$ g++ -o fork2 fork2.cpp
$ ./fork2
```



```

The owner of this process has uid 0
root@bagusok:~/jobsheet8# nano fork2.cpp
root@bagusok:~/jobsheet8# g++ -o fork2 fork2.cpp
root@bagusok:~/jobsheet8# ./fork2
This is process 18466
x is 5
This is process 18467
x is 5
This is process 18466
x is 6
This is process 18467
x is 6
This is process 18466
x is 7
This is process 18467
x is 7
This is process 18466
x is 8
This is process 18467
x is 8
|
```

3. Amati output yang dihasilkan

Analisa: Compile file fork2.cpp yang sudah kita buat dengan menggunakan perintah `g++ -o fork2 fork2.cpp`, lalu jika tidak ada file yang error, maka ketikkan `./fork2` untuk menjalankan program fork2. Output dari program ini adalah membuat dua proses terus menerus yang dimulai dengan `x = 5` dengan sebuah system call `fork()`.

Percobaan 3 : Membuat dua proses sebanyak lima kali

1. Dengan menggunakan editor vi, buatlah file fork3.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
padaunistd.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;
}
```

Analisa: Menggunakan editor vi untuk membuat file foik3.cpp. Isi dari file foik3.cpp ialah untuk membuat dua proses sebanyak lima kali.

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

/* getpid() dan fork() adalah system call yang dideklarasikan pada unistd.h. Me
nghasilkan suatu nilai dengan type pid_t. pid_t adalah type khusus untuk proces
s id yg ekuivalen dgn 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. Gunakan g++ compiler untuk menjalankan program diatas

```
$ g++ -o fork3 fork3.cpp
```

```
$ ./fork3
```

```
root@bagusok:~/jobsheet8# nano fork3.cpp
root@bagusok:~/jobsheet8# g++ -o fork3 fork3.cpp
root@bagusok:~/jobsheet8# ./fork3
This is process 18474
This is process 18475
This is process 18474
This is process 18475
|
```

3. Amati output yang dihasilkan

Analisa: Percobaan ini membuat dua proses dalam satu terminal, yang dapat berjalan sebanyak 5 kali dengan file fork3.cpp. Untuk mendapatkan hasil seperti itu kita menggunakan pengulangan for yang akan menampilkan this is process (pid).

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>

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

```

```

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

/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk pr
ocess 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 dieksekusi 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() << en
dl;

    if (child_pid == 0) {
        /* kode ini hanya dieksekusi proses child */
        cout << "I am a child and I am quitting work now!" << endl;
    }
    else {
        /* kode ini hanya dieksekusi 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;
}

:wq

```

Analisa: File fork4.cpp berisi untuk membuat script yang nantinya akan digunakan untuk proses parent menunggu sinyal dari proses child dengan system call wait

2. Gunakan g++ compiler untuk menjalankan program diatas.

```

$ g++ -o fork4 fork4.cpp
$ ./fork4

```

```
root@bagusok:~/jobsheet8# nano fork4.cpp
root@bagusok:~/jobsheet8# nano fork4.cpp
root@bagusok:~/jobsheet8# g++ -o fork4 fork4.cpp
root@bagusok:~/jobsheet8# ./fork4
I am the parent and my pid = 18489
My child has pid = 18490
I am a happy, healthy process and my pid = 18489
I am a parent and I am going to wait for my child
I am a child and my pid = 18490
My parent is 18489
I am a happy, healthy process and my pid = 18490
I am a child and I am quitting work now!
I am a parent and I am quitting.
root@bagusok:~/jobsheet8# |
```

3. Amati output yang dihasilkan

Analisa: Digunakan untuk mencompile fork4.cpp yang dipakai untuk proses parent menunggu sinyal dari proses child dengan system call wait. Dan outputnya seperti gambar diatas.

Peicobaan 5 : System call fork/exec dan wait mengeksekusi program bernama ls, menggunakan file executable /bin/ls dengan satu parameter -l yang ekuivalen dengan ls -l

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

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

```

```

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

/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk pr
ocess 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 dieksekusi 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 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
diterminasi */
            wait_result = wait(&status);
        } while (wait_result != child_pid);
        cout << "I am a parent and I am quitting. " << endl;
    }
    return 0;
}
:wq

```

Analisa: Membuat script file fork5.cpp yang akan digunakan untuk mengeksekusi ls.

2. Gunakan g++ compiler untuk menjalankan program diatas.

```
$ g++ -o fork5 fork5.cpp
```

```
$ ./fork5
```

```
root@bagusok:~/jobsheet8# nano fork5.cpp
root@bagusok:~/jobsheet8# g++ -o fork5 fork5.cpp
root@bagusok:~/jobsheet8# ./fork5
I am the parent and my pid = 18497
My child has pid = 18498
I am a happy, healthy process and my pid = 18497
I am a parent and I am going to wait for my child
I am a child and my pid = 18498
total 4
drwxr-xr-x 3 bagus bagus 4096 Mar 26 04:11 bagus
I am a parent and I am quitting.
root@bagusok:~/jobsheet8# |
```

3. Amati output yang dihasilkan

Analisa: Compile program fork5.cpp dan program ini digunakan untuk mengeksekusi program bernama ls, menggunakan file executable /bin/ls dengan satu parameter -l yang ekuivalen dengan ls -l.

Peicobaan 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>

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

```

```

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

/* pid_t fork() dideklarasikan pada unistd.h. pid_t adalah type khusus untuk pr
ocess 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 dieksekusi 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 mengekseku
si dari "fork3" atau keluar */
    cout << "I am a happy, healthy process and my pid = " << getpid() << en
dl;

    if (child_pid == 0) {
        /* kode ini tidak pernah dieksekusi */
        printf("This code will never be executed!\n");
    }
    else {
        /* kode ini hanya dieksekusi 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;
}
:wq

```

Analisa: Membuat program fork6.cpp yang akan digunakan untuk menggunakan system call fork/exec dan wait untuk mengeksekusi program lain.

2. Gunakan g++ compiler untuk menjalankan program diatas.

```

$ g++ -o fork6 fork6.cpp
$ ./fork6

```

```
|
root@bagusok:~/jobsheet8# nano fork6.cpp
root@bagusok:~/jobsheet8# g++ -o fork6 fork6.cpp
root@bagusok:~/jobsheet8# ./fork6
I am the parent and my pid = 18510
My child has pid = 18511
I am a happy, healthy process and my pid = 18510
I am a parent and I am going to wait for my child
I am a child and my pid = 18511
This is process 18511
This is process 18512
This is process 18511
This is process 18512
|
```

3. Amati output yang dihasilkan

Analisa: Compile file fork6.cpp untuk menjalankan program system call fork/exec dan wait yang nantinya akan digunakan untuk mengeksekusi program lain. Yang isinya dalam program ini seperti diatas. Dan disana juga menampilkan 2 proses yang berbeda.

Percobaan 7 : Melihat Manajemen Memoi

1. Perhatikan dengan perintah dmesg jumlah memory tersedia dan proses swapping

```
$ dmesg | more
This is process 18512
I am a parent and I am quitting.
root@bagusok:~/jobsheet8# dmesg | more
[ 0.000000] Linux version 5.4.0-171-generic (buildd@lcy02-amd64-005) (gcc
version 9.4.0 (Ubuntu 9.4.0-1ubuntu1~20.04.2)) #189-Ubuntu SMP Fri Jan 5 14
:23:02 UTC 2024 (Ubuntu 5.4.0-171.189-generic 5.4.259)
[ 0.000000] Command line: BOOT_IMAGE=/vmlinuz-5.4.0-171-generic root=/dev
/mapper/ubuntu--vg-ubuntu--lv ro maybe-ubiquity
[ 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] BIOS-provided physical RAM map:
[ 0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[ 0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserv
ed
[ 0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000ffffff] reserv
ed
[ 0.000000] BIOS-e820: [mem 0x0000000000100000-0x00000000007ffeffff] usable
[ 0.000000] BIOS-e820: [mem 0x00000000007fff0000-0x00000000007fffffff] ACPI d
ata
[ 0.000000] BIOS-e820: [mem 0x0000000000fec00000-0x0000000000fec0ffff] reserv
```

Analisa: Perintah dmesg digunakan untuk melihat jumlah memory tersedia dan proses swapping.

2. Dengan perintah freeperhatikan jumlah memory "free", "used", "share" dan "buffer" .

```
$ free
[ 0.061052] setup_percpu: NR_CPUS:8192 nr_cpumask_bits:1 nr_cpu_ids:1 nr_
root@bagusok:~/jobsheet8# free
              total            used             free             shared  buff/cache          availa
ble
Mem:           2018932          152796           855908              1104          1010228           1692
532
Swap:           2002940               0          2002940
root@bagusok:~/jobsheet8# |
```

Analisa: Free digunakan untuk mengetahui total memori yang digunakan dalam proses. Dalam perintah free ditampilkan total kapasitas memori, memori yang terpakai, yang tidak sedang dipakai, yang dibagi, buffer, cache dan juga swap.

3. Dengan perintah dibawah ini apakah hasilnya sama dengan no 2 ?

```
$ cat /proc/meminfo
```



```

Swap:          2002940      0      2002940
root@bagusok:~/jobsheet8# cat /proc/meminfo
MemTotal:      2018932 kB
MemFree:       855916 kB
MemAvailable:  1692552 kB
Buffers:       52612 kB
Cached:        891728 kB
SwapCached:    0 kB
Active:        597388 kB
Inactive:      399852 kB
Active(anon):  62048 kB
Inactive(anon): 160 kB
Active(file):  535340 kB
Inactive(file): 399692 kB
Unevictable:   18644 kB
Mlocked:       18644 kB
SwapTotal:     2002940 kB
SwapFree:      2002940 kB
Dirty:         20 kB
Writeback:     0 kB
AnonPages:     71568 kB

```

Analisa: Dalam percobaan dengan perintah `cat /proc/meminfo` berbeda dengan nomer 2 dengan perintah `free` karena disini disk yang terpakai lebih terperinci dengan jelas, dan informasi memori total dan swab total sama, untuk yang lain sedikit berbeda dengan perintah `free`. sedangkan perintah `free` hanya secara global bukan secara khusus.

4. Gunakan perintah dibawah ini

```
$ ls -lR /.
```

Analisa: Perintah `ls -lR /.` digunakan untuk menampilkan isi dari suatu direktori dengan menampilkan informasi file tersebut,

```

DirectMap4k:    96192 kB
DirectMap2M:   2000896 kB
root@bagusok:~/jobsheet8# ls -lR /.
ls: cannot access '-lR': No such file or directory
/.:
bin    etc    lib32  lost+found  opt    run    srv    tmp
boot  home  lib64  media      proc   sbin   swap.img  usr
dev    lib    libx32  mnt      root   snap   sys     var
root@bagusok:~/jobsheet8# |

```

5. Perhatikan perubahan manajemen memory

```
$ free
```

Analisa: Perubahan terjadi pada memory used, shared, dan buff/cache menjadi bertambah, sedangkan memory free dan available menjadi berkurang.

```
dev  lib  libx32  mnt  root  snap  sys  var
root@bagusok:~/jobsheet8# free
              total        used        free      shared  buff/cache   availa
ble
Mem:         2018932      152772      855916         1104      1010244      1692
560
Swap:         2002940           0      2002940
```

6. Jalankan sebuah program, misalnya open Office. Perhatikan perubahan manajemen memory

```
$ free
```

Analisa: Perubahan terjadi pada memory shared menjadi bertambah, sedangkan memory used, free, buff/cache, dan available menjadi berkurang.

7. Dengan perintah ps bagaimana penggunaan memory untuk setiap proses diatas ?

```
$ ps-uax
```

```
root@bagusok:~/jobsheet8# ps -uax
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root           1  0.0  0.5 102748 11544 ?        Ss   05:50   0:03 /sbin/init
root           2  0.0  0.0      0     0 ?        S    05:50   0:00 [kthreadd
root           3  0.0  0.0      0     0 ?        I<   05:50   0:00 [rcu_gp]
root           4  0.0  0.0      0     0 ?        I<   05:50   0:00 [rcu_par_
root           6  0.0  0.0      0     0 ?        I<   05:50   0:00 [kworker/
root           8  0.0  0.0      0     0 ?        I<   05:50   0:00 [mm_percp
root           9  0.0  0.0      0     0 ?        S    05:50   0:03 [ksoftirq
root          10  0.0  0.0      0     0 ?        I    05:50   0:02 [rcu_sche
root          11  0.0  0.0      0     0 ?        S    05:50   0:00 [migratio
root          12  0.0  0.0      0     0 ?        S    05:50   0:00 [idle_inj
root          14  0.0  0.0      0     0 ?        S    05:50   0:00 [cpuhp/0]
root          15  0.0  0.0      0     0 ?        S    05:50   0:00 [kdevtmpf
root          16  0.0  0.0      0     0 ?        I<   05:50   0:00 [netns]
root          17  0.0  0.0      0     0 ?        S    05:50   0:00 [rcu_task
root          18  0.0  0.0      0     0 ?        S    05:50   0:00 [kauditd]
root          19  0.0  0.0      0     0 ?        S    05:50   0:00 [khungtas
root          20  0.0  0.0      0     0 ?        S    05:50   0:00 [oom_reap
root          21  0.0  0.0      0     0 ?        T<   05:50   0:00 [writebac
```

Analisa: Perintah ps -uax digunakan untuk menunjukkan bagaimana penggunaan memory berubah secara dinamis dan bagaimana proses individu menggunakan memory.



LATIHAN:

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

a. `ls -al /etc.`

```
root@bagusok:~/jobsheet8# nano fork5.cpp
root@bagusok:~/jobsheet8# g++ -o forkls fork5.cpp
root@bagusok:~/jobsheet8# ./forkls
I am the parent and my pid = 18528
My child has pid = 18529
I am a happy, healthy process and my pid = 18528
I am a parent and I am going to wait for my child
I am a child and my pid = 18529
total 872
drwxr-xr-x 104 root root      4096 Apr 23 06:34 .
drwxr-xr-x  19 root root      4096 Feb 20 03:20 ..
-rw-r--r--   1 root root      3028 Mar 14 2023 adduser.conf
-rw-r--r--   1 root root        51 Feb 27 06:27 aliases
-rw-r--r--   1 root root     12288 Feb 27 06:27 aliases.db
drwxr-xr-x   2 root root      4096 Apr 23 06:32 alternatives
drwxr-xr-x   3 root root      4096 Mar 14 2023 apparmor
drwxr-xr-x   7 root root      4096 Feb 20 06:40 apparmor.d
drwxr-xr-x   3 root root      4096 Feb 20 06:29 apport
drwxr-xr-x   7 root root      4096 Feb 20 03:16 apt
-rw-r-----   1 root daemon    144 Nov 12 2018 at.deny
-rw-r--r--   1 root root     2319 Feb 25 2020 bash.bashrc
-rw-r--r--   1 root root        45 Jan 26 2020 bash_completion
drwxr-xr-x   2 root root      4096 Feb 20 06:29 bash_completion.d
-rw-r--r--   1 root root       367 Apr 14 2020 bindresvport.blackl
drwxr-xr-x   2 root root      4096 Apr 22 2020 binfmt.d
drwxr-xr-x   2 root root      4096 Mar 14 2023 byobu
```

b. `cat fork2`

```
root@bagusok:~/jobsheet8# cat fork2  
@@@(((<(      ((   ((X-X=X=op-p=p888 XXXDDSdtd888 Pdt    TTQdtdRdtdX-X=X=((  
/lib64/ld-linux-x86-64.so.2GNUGNU(((azOR  
  
omknnGNUee m  
  
C'_'_@3,,,(U U ; (  
F"?@@libstdc++.so.6__gmon_start___ITM_deregisterTMCloneTable_ITM_registerTMC  
loneTable_ZNSolsEi_ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_  
_ZNSt8ios_base4InitD1Ev_ZNSolsEPFRSoS_E_ZStlsIS11char_traitsIcEERSt13basic_  
ostreamIcT_ES5_PKc_ZNSt8ios_base4InitC1Ev_ZSt4coutlibc.so.6forkgetpid__cxx_a  
textitsleep__cxx_finalize__libc_start_mainGLIBCXX_3.4GLIBC_2.2.5 t)gu?i sX `  
@@@@?  
??  
?  
?  
?@??? ????? ?H H /HottHH5R/?%S/hhhhhh h h h  
/D%%. D%%. D%%. D%%. D%%. D%%. D%%. D%%. D%%. D1^H  
H==r.H==.H..H9ttHN.Hott  HH=i.H5b.H)HHH?HHHHttH%.HHf  
DDDDD=e/u+UH=-Hott  
H=-^-^HHHEHHHHHHHH$-HHHHIIHHHH{UUHU}uu}u2}?  
~ ~ ~
```

```
root@bagusok:~/jobsheet8# ./fork2
This is process 18531
x is 5
This is process 18532
x is 5
This is process 18531
x is 6
This is process 18532
x is 6
```

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

Jawab: Perintah `dmesg` digunakan untuk melihat jumlah memori tersedia dan proses swapping.

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

Jawab: `Free` digunakan untuk mengetahui total memori yang digunakan dalam proses. Dalam perintah `free` ditampilkan total kapasitas memori, memori yang terpakai, yang tidak sedang dipakai, yang dibagi, buffer, cache dan juga swap.

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

Jawab: Dalam percobaan dengan perintah `cat /proc/meminfo` berbeda dengan nomor 2 percobaan 7 dengan perintah `free` karena disini disk yang terpakai lebih terperinci dengan jelas, dan informasi memori total dan swap total sama, untuk yang lain sedikit berbeda dengan perintah `free`. sedangkan perintah `free` hanya secara global bukan secara khusus.

5. Berapa besar memori yang digunakan setelah percobaan 7 dengan perintah `ps -uax` ?

Jawab: Besar memori yang digunakan setelah percobaan 7 dengan perintah `ps -uax` ialah 0,3 %.

6. Lakukan hal yang sama dengan percobaan 7 untuk melihat perubahan memori setelah dilakukan beberapa proses pada shell. Tentukan perintah yang dilakukan misalnya membuka browser dan perhatikan hal-hal berikut :

- a. Informasi apa saja yang ditampilkan dengan perintah `free` ?
- b. Informasi apa saja yang disimpan file `/proc/meminfo` ?

c. Berapa besar kapasitas memori total ? 1004628 kB atau 1 GB.

d. Berapa kapasitas memori yang sudah terpakai ? 158552 kB.

e. Berapa kapasitas memori yang belum terpakai ? 279496 kB.

f. Berapa kapasitas memori yang digunakan sharing beberapa proses ? 512 kB.

g. Berapa kapasitas buffer cache ? 16376 kB.

Kesimpulan:

- System call merupakan penyedia antarmuka dari pelayanan-pelayanan yang tersedia dengan system operasi. Umumnya system call menggunakan bahasa C dan C++, meskipun tugas-tugas seperti hardware yang harus diakses langsung, maka menggunakan bahasa

assembly. System call fork adalah suatu system call yang membuat suatu proses baru pada system operasi UNIX.

- Pada percobaan ini menggunakan mesin Linux dan beberapa program yang berisi system call fork(). Sistem call execl adalah peletakkan program executable baru ke memori dan mengasosiasikannya dengan proses saat itu. Dengan kata lain, mengubah segala sesuatunya sehingga program mulai mengeksekusi dari file yang berbeda.
- System call wait dapat menyebabkan proses menunggu sinyal (menunggu sampai sembarang tipe sinyal diterima dari sembarang proses). System call wait menghasilkan pid dari proses yang mengirimkan sinyal.
- Virtual memori adalah suatu teknik memisahkan antara memori logis dan memori fisiknya. Memori logis merupakan kumpulan keseluruhan halaman dari suatu program. Tanpa memori virtual, memori logis akan langsung dibawa ke memori fisik (memori utama).
- Swapping adalah manajemen memori dengan pemindahan proses antara memori utama dan disk selama eksekusi. Buffer cache dapat dianggap sebagai sumber daya memori, terutama sumber daya I/O karena penggunaannya dalam mediasi transfer.