Praktikum 7

Unix System Call dan Manajemen Memory

POKOK BAHASAN:

- ✓ UNIX System Call
- ✓ Manajemen Memory

TUJUAN BELAJAR:

Setelah mempelajari materi dalam bab ini, mahasiswa diharapkan mampu:

- ✓ Menggunakan system call fork, wait dan execl pada Linux.
- ✓ Menggunakan perintah-perintah untuk manajemen memory.

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, execl 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 parend 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 konkuren.

Nilai yang dihasilkan oleh `fork()` disimpan dalam variable bertipe `pid_t`, yang berupa nilai integer. Karena nilai dari variable 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 variable 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 variable 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 diterminasi. System call wait menghasilkan pid dari proses yang mengirimi sinyal. Untuk melihat dokumentasi wait gunakan perintah `man 2 wait`.

System Call Execl

Misalnya kita ingin proses baru mengerjakan sesuatu yang berbeda dari proses parent, sebutlah menjalankan program yang berbeda. Sistem call execl meletakkan program executable baru ke memory 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 memory demand-paged. Proses mempunyai besar memory virtual yang besar (4 gigabyte). Pada virtual memory dilakukan transfer page antara disk dan memory fisik.

Jika tidak terdapat cukup memory 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 memory fisik. Jika memory total page lebih dari memory fisik yang tersedia, kernel lebih banyak melakukan swapping dibandingkan eksekusi kode program, sehingga terjadi thrashing dan mengurangi utilitas.

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

Pertama kali sistem melakukan booting, ROM BIOS membentuk memory 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 memory (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 memory yang tersedia. Atau menggunakan `cat /proc/meminfo`. Memory fisik dan ruang swap ditampilkan disini. Contoh output pada sistem :

```
total used free shared buffers
Mem:  7096  5216 1880   2328     2800
Swap: 19464 0    19464
```

Informasi ditampilkan dalam kilobyte (1024 byte). Memory "`total`" adalah jumlah tersedia setelah load kernel. Memory digunakan untuk proses atau disk bufferring sebagai "`used`". Memory yang sedang tidak digunakan ditampilkan pada kolom "`free`". Memory total sama dengan jumlah kolom "`used`" dan "`free`".

Memory diindikasikan "`shared`" yaitu berapa banyak memory yang digunakan lebih dari satu proses. Program seperti shell mempunyai lebih dari satu proses yang berjalan. Kode executable read-only dan dapat disharing oleh semua proses yang berjalan pada shell. Kolom "`buffers`" menampilkan berapa banyak memory digunakan untuk disk buffering.

Perintah `free` juga menunjukkan dengan jelas bagaimana swap space dilakukan dan berpa banyak swapping yang terjadi.

Percobaan berikut untuk mengetahui manajemen memory :

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
```

   Terdapat free memory (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 berakibat pula pada kolom "`used`" dan memory "`free`" juga berkurang.

Perintah `top` dan `ps -u` juga sangat berguna untuk menunjukkan bagaimana penggunaan memory berubah secara dinamis dan bagaimana proses individu menggunakan memory. 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 ?
2. Apa yang dimaksud dengan sistem call `fork(), execl()` dan `wait().` Jawablah dengan menggunakan perintah man (contoh : `man 2 fork`, `man 2 execl` dan `man 2 wait`) ?
3. Apa yang dimaksud sistem virtual memory, proses swapping dan buffer cache pada manajemen memory ?
4. Apa yang dimaksud perintah free dan cat /proc/meminfo ?
5. Apa yang dimaksud perintah ps ?

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 `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(1);
   }
   return 0;
}
```

2. Gunakan g++ compiler untuk menjalankan program diatas

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

3. Amati output yang dihasilkan

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(1);
       x++;
   }
   return 0;
}
```

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
```

3. Amati output yang dihasilkan

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
   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. Gunakan g++ compiler untuk menjalankan program diatas

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

3. Amati output yang dihasilkan

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;
}
```

2. Gunakan g++ compiler untuk menjalankan program diatas

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

3. Amati output yang dihasilkan

Percobaan 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;
}
```

2. Gunakan g++ compiler untuk menjalankan program diatas

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

3. Amati output yang dihasilkan

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>

/* 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. Gunakan g++ compiler untuk menjalankan program diatas

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

3. Amati output yang dihasilkan

Percobaan 7 : Melihat Manajemen Memory

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

   ```
   $ dmesg | more
   ```

2. Dengan perintah `free` perhatikan jumlah memory ”free”, ”used”, “share” dan “buffer” .

   ```
   $ free
   ```

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

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

4. Gunakan perintah dibawah ini

   ```
   $ ls –lR /.
   ```

5. Perhatikan perubahan manajemen memory

   ```
   $ free
   ```

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

   ```
   $ free
   ```

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

   ```
   $ ps -uax
   ```

LATIHAN:

1. Ubahlah program fork5.cpp pada percobaan 5 untuk mengeksekusi perintah yang ekuivalen dengan
   a. `ls –al /etc.`
   b. `cat fork2`
   c. `./fork2`
2. Informasi apa saja mengenai manajemen memory yang ditampilkan pada perintah `dmesg` pada percobaan Anda ?
3. Bagaimana informasi yang ditampilkan dengan perintah `free` pada percobaan Anda ?
4. Apa isi file `/proc/meminfo` pada percobaan yang Anda lakukan ?
5. Berapa besar memory yang digunakan setelah percobaan 7 dengan perintah `ps –uax` ?
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 :
   a. Informasi apa saja yang ditampilkan dengan perintah `free` ?
   b. Informasi apa saja yang disimpan file `/proc/meminfo` ?
   c. Berapa besar kapasitas memory total ?
   d. Berapa kapasitas memory yang sudah terpakai ?
   e. Berapa kapasitas memory yang belum terpakai ?
   f. Berapa kapasitas memory yang digunakan sharing beberapa proses ?
   g. Berapa kapasitas buffer cache ?

LAPORAN RESMI:

1. Analisa hasil percobaan yang Anda lakukan.
2. Kerjakan latihan diatas dan analisa hasil tampilannya.
3. Berikan kesimpulan dari praktikum ini.