Tugas Besar 1 - IF2230 Sistem Operasi

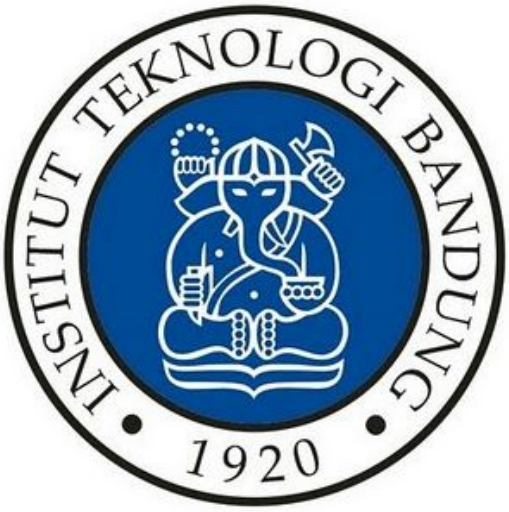
Petunjuk Pengerjaan Tugas Besar 1

Memory Management Unit

Dipersiapkan oleh:

Asisten Lab Sistem Terdistribusi

Didukung oleh:



START: 2 Maret 2016

END: 21 Maret 2016

A. Latar Belakang

Selamat datang di Lovelyz in Wonderland!

Saat ini, Lovelyz sedang dalam masa istirahat. Untuk itu, mereka berlibur ke suatu tempat yang merupakan impian mereka. Apa saja yang mereka lakukan di sana? Hanya di sini mereka dapat melakukan hal-hal yang dilarang ataupun tidak dapat dilakukan di asrama, seperti menonton televisi, tidur di kasur yang nyaman, dan makan makanan ringan karena mereka diharuskan untuk menjaga berat badan ideal.

Namun, meski sedang berlibur, mereka tetap ingin menumpas kejahatan yang dilakukan Red▽elvet. Sejak Sujeong kembali dalam pelukan, seluruh anggota Lovelyz sepakat untuk menggagalkan rencana Red▽elvet bersama-sama. Pada chapter sebelumnya, Sujeong telah memasang *rootkit* pada salah satu komputer mereka. Sekarang mereka ingin melakukan SSH ke *rootkit* tersebut. Untungnya, di Wonderland ini terdapat sebuah komputer dan koneksi internet. Sayangnya komputer tersebut tidak terawat dan *Memory Management Unit* (MMU)-nya rusak. Kebetulan, Jin membawa MMU yang sudah ia kembangkan sebelumnya bersamanya. Hanya saja ia memerlukan sistem operasi khusus untuk menguji apakah MMUnya sudah *stable* atau belum.

Seperti dalam kuliah yang telah diberikan, sebuah proses akan diberikan sebuah alokasi memori virtual. *Virtual memory* ini menjadikan proses dapat memiliki P page di memori virtual dan F frame di memori fisik, meskipun F lebih kecil dari P.

Hal tersebut dapat terjadi karena karena penggunaan page table yang memetakan page pada *virtual memory* ke *frame* pada memori fisik, dan mekanisme *pagefault* yang dilakukan MMU yang memerintahkan *operating system* (OS) untuk mengambil kembali page dari disk ke memori fisik. *Page table* harus dapat diakses oleh OS maupun MMU, dan IPC diperlukan untuk komunikasi antara MMU dan OS.

Kebetulan Anda juga sedang berlibur di Wonderland, sehingga kalian ditugaskan untuk membantu mereka. Dalam tugas ini, Anda diberikan sebuah simulator MMU. Buatlah sebuah program yang akan melakukan simulasi manajemen memori yang dilakukan OS pada memori fisik dan *page table*. *Page table* berupa *shared memory* dan signal digunakan untuk IPC. Anda tidak harus melakukan simulasi pada memori fisik. Namun, untuk membuat tugas ini lebih mirip dengan apa yang terjadi pada OS, setiap melakukan akses pada *disk*, Anda harus menambahkan perintah `sleep(1)`.

B. Detail Tugas

Tugas ini dikerjakan dalam kelompok 3 orang.

Page Table

Anda diberikan sebuah struktur data *page table*:

```
1. //PageTable.h
2. typedef struct {
3.     int Valid;
4.     int Frame;
5.     int Dirty;
6.     int Requested;
7.     } page_table_entry;
8.
9. typedef page_table_entry* page_table_pointer;
```

Tiap entry *page table* memiliki beberapa field:

- Sebuah *Boolean* `Valid` yang menyatakan apakah page tersebut ada di memori fisik.
- Sebuah *integer* `Frame` yang menyatakan indeks frame page tersebut di memori fisik.
- Sebuah *Boolean* `Dirty` yang menyatakan apakah page tersebut telah ditulis.
- Sebuah *integer* `Requested` yang bernilai bukan-nol hanya jika page tersebut tidak pada memori fisik dan dipesan oleh MMU. Pada kasus tersebut, nilai ini diisi dengan ID proses (*PID*) dari MMU.

Proses simulasi OS harus membuat *page table* dalam *shared memory*, dan page table tersebut dinisiasi untuk menyatakan tidak ada page yang di load ke memori fisik (seluruh field `Valid` dinisiasi sebagai 0). Mungkin Anda perlu menambahkan beberapa field pada struktur data page table, sehingga mungkin Anda perlu mengubah kode MMU juga.

Memory Management Unit

Dalam tugas ini, Anda diberikan sebuah *source code* template untuk MMU.

Simulator MMU akan memerlukan 3 argumen untuk menjalankannya.

- Jumlah page pada sebuah proses.

- Beberapa kode simulasi pembacaan dan penulisan memori dengan bentuk <mode><page>, misalkan `W5` berarti proses menulis ke *page* 5.
- PID dari proses OS.

Dalam eksekusinya, MMU akan melakukan *attach shared-memory* dengan menggunakan key PID dari proses OS, kemudian menjalankan serangkaian proses simulasi.

1. Mengecek apakah page ada dalam memori fisik.
2. Jika tidak, MMU akan menuliskan PID ke field `Requested` untuk page tersebut.
3. Mengirimkan sinyal `SIGUSR1` ke proses OS.
4. Blocking, sampai mendapatkan sinyal `SIGCONT` dari proses OS yang menyatakan bahwa page telah dimuat ke memori fisik.
5. Jika proses ini merupakan proses penulisan, melakukan update pada `Dirty` menjadi `true`.
6. Melakukan penulisan status page table sekarang.

Ketika seluruh proses simulasi telah dilakukan, MMU melakukan *deattach* dari *shared-memory* dan mengirimkan sinyal pada OS untuk terakhir kali, tanpa melakukan update pada field `Requested` pada page manapun, yang kemudian akan diterima oleh OS untuk kemudian melakukan penghapusan *shared-memory* dan berhenti.

OS

OS simulator memerlukan 2 argumen untuk menjalankannya:

- Jumlah *page* dalam sebuah proses
- Jumlah *frame* yang digunakan oleh sebuah proses

Asumsikan indeks page dan frame menggunakan 0, 1, 2, dst.

Setelah membuat dan menginisiasi page table di *shared memory*, OS akan melakukan loop untuk menunggu sinyal `SIGUSR1` dari MMU. Ketika sinyal itu diterima, OS akan melakukan hal berikut:

1. Scan pada page table untuk menemukan `Requested` field yang diisi oleh MMU.
2. Jika ditemukan, artinya *page* tersebut dibutuhkan oleh MMU.
3. Jika terdapat *frame* yang tidak ditempati, alokasikan.
4. Jika tidak ada seluruh *frame* ditempati, pilih *page* ‘victim’ yang akan di *swap*.

- Jika *page* 'victim' tersebut memiliki `dirty` bernilai `true`, simulasikan penulisan pada disk, lakukan `sleep(1)` dan tambahkan nilai pengaksesan pada disk.
- Update *page table* untuk menyatakan page tersebut sudah tidak berada pada memori fisik.

5. Simulasikan pemuatan page dengan `sleep(1)` dan tambahkan nilai pengaksesan pada disk.
6. Update *page table* untuk menyatakan page tersebut telah di load ke memori fisik dalam *frame* tersebut, tentu dengan `dirty` bernilai `false`, dan kembalikan nilai `Requested` ke 0.
7. Cetak *page table* setelah diupdate.
8. Mengirimkan sinyal `SIGCONT` ke MMU untuk menyatakan page telah dimuat.
9. Jika tidak ditemukan field `Requested` dengan nilai bukan nol, berhenti.

Anda harus memilih algoritma dan mengimplementasikannya untuk menentukan 'victim' page. Mungkin Anda perlu menambahkan beberapa field pada struktur data *page table*, sehingga mungkin Anda perlu mengubah MMU juga. Sebelum OS berhenti, OS harus menuliskan berapa kali pengaksesan *disk*, dan kemudian melakukan penghapusan *shared-memory*.

Contoh Interaksi

```
> OS 5 2
The shared memory key (PID) is 78801
Initialized page table
                                > MMU 5 W2 R3 W3 R4 78801
                                Initialized page table:
                                0: Valid=0 Frame=-1 Dirty=0 Requested=0
                                1: Valid=0 Frame=-1 Dirty=0 Requested=0
                                2: Valid=0 Frame=-1 Dirty=0 Requested=0
                                3: Valid=0 Frame=-1 Dirty=0 Requested=0
                                4: Valid=0 Frame=-1 Dirty=0 Requested=0

                                Request for page 2 in W mode
                                It's not in RAM - page fault
Process 78803 has requested page 2
Put it in free frame 0
Unblock MMU
                                Set the dirty bit for page 2
```

```
                                0: Valid=0 Frame=-1 Dirty=0 Requested=0
                                1: Valid=0 Frame=-1 Dirty=0 Requested=0
                                2: Valid=1 Frame= 0 Dirty=1 Requested=0
                                3: Valid=0 Frame=-1 Dirty=0 Requested=0
                                4: Valid=0 Frame=-1 Dirty=0 Requested=0

                                Request for page 3 in R mode
                                It's not in RAM - page fault
Process 78803 has requested page 3
Put it in free frame 1
Unblock MMU
                                0: Valid=0 Frame=-1 Dirty=0 Requested=0
                                1: Valid=0 Frame=-1 Dirty=0 Requested=0
                                2: Valid=1 Frame= 0 Dirty=1 Requested=0
                                3: Valid=1 Frame= 1 Dirty=0 Requested=0
                                4: Valid=0 Frame=-1 Dirty=0 Requested=0

                                Request for page 3 in W mode
                                It's in RAM
                                Set the dirty bit for page 3
                                0: Valid=0 Frame=-1 Dirty=0 Requested=0
                                1: Valid=0 Frame=-1 Dirty=0 Requested=0
                                2: Valid=1 Frame= 0 Dirty=1 Requested=0
                                3: Valid=1 Frame= 1 Dirty=1 Requested=0
                                4: Valid=0 Frame=-1 Dirty=0 Requested=0

                                Request for page 4 in R mode
                                It's not in RAM - page fault
Process 78803 has requested page 4
Chose a victim page 2
Victim is dirty, write out
Put in victim's frame 0
Unblock MMU
                                0: Valid=0 Frame=-1 Dirty=0 Requested=0
                                1: Valid=0 Frame=-1 Dirty=0 Requested=0
                                2: Valid=0 Frame=-1 Dirty=0 Requested=0
                                3: Valid=1 Frame= 1 Dirty=1 Requested=0
                                4: Valid=1 Frame= 0 Dirty=0 Requested=0
                                Tell OS that I'm finished
```

```
The MMU has finished
 0: Valid=0 Frame=-1 Dirty=0 Requested=0
 1: Valid=0 Frame=-1 Dirty=0 Requested=0
 2: Valid=0 Frame=-1 Dirty=0 Requested=0
 3: Valid=1 Frame= 1 Dirty=1 Requested=0
 4: Valid=1 Frame= 0 Dirty=0 Requested=0
4 disk accesses required
```

C. *Deliverables*

- Source sistem operasi diimplementasikan dengan C atau C++
- Source MMU.c apabila diubah
- Header PageTable.h apabila diubah
- Dokumentasi singkat

D. Kriteria Penilaian

- 10% dokumentasi
- 10% coding style
- 30% efisiensi (akses disk tiap kasus)
- 50% keberhasilan implementasi