Tugas Besar - Beneath The Skin
IF2230 Sistem Operasi
Operating System Development

Dipersiapkan oleh:
Asisten Laboratorium Sistem Terdistribusi

Didukung oleh:

START: 29 Maret 2017
END: 18 April 2017

I. Latar Belakang

"Yellow Light"

I'm looking for a place to start
And everything feels so different now
Just grab a hold of my hand
I will lead you through this wonderland
Water up to my knees
But sharks are swimming in the sea
Just follow my yellow light
And ignore all those big warning signs

Somewhere deep in the dark
A howling beast hears us talk
I dare you to close your eyes
And see all the colors in disguise
Running into the night
The earth is shaking and I see a light
The light is blinding my eyes
As the soft walls eat us alive

II. Milestone 3 : Processes and Multiprogramming

Pada umumnya, sebuah OS dapat menjalankan beberapa proses dalam satu waktu. Tahap ini akan memperkenalkan bagaimana sebuah OS mengelola proses yang berjalan. Pada milestone ini, Anda akan mengimplementasi algoritma scheduling Round Robin dalam OS Anda sehingga OS dapat menjalankan banyak proses sekaligus.

Spesifikasi Tugas

1. OS dapat menjalankan banyak proses sekaligus dengan Round Robin
2. Mengimplementasi prosedur **showProcesses**, untuk menunjukkan proses apa yang berjalan dan segment berapa
3. Mengimplementasi fungsi **kill** untuk menghentikan proses di segment tertentu

Background

Ada tiga komponen utama dalam mengimplementasi multiprogramming, yaitu :

1. Memory Management
2. Time Sharing
3. Process Management

Memory Management

Sistem operasi Anda akan mengelola memory secara *fixed size*. Memory dibagi dalam 10 segmen (0x0000, 0x1000, 0x2000, ….., 0x9000). Segmen 0x0000 dipakai untuk interrupt vector, segmen 0x1000 dipakai untuk kernel, sementara sisanya akan digunakan sebagai segmen program berjalan. Dengan demikian, maksimal program yang bisa berjalan secara konkuren hanya 8. Setiap ada program yang akan dieksekusi, OS Anda akan mencari segmen yang kosong untuk diisi. Setelah program selesai, segmen yang digunakan oleh program tersebut akan dikosongkan kembali.

Time Sharing

Agar program bisa berjalan seolah-olah konkuren, OS akan mengganti proses yang berjalan dalam setiap sepersekian detik. Dalam tugas ini, OS akan melakukan interrupt sebanyak 12 kali dalam 1 detik. Setiap proses akan memanfaatkan waktu 1/12 detik tersebut sebelum diubah ke proses lain.

Process Management

Selain membagi memory dan waktu, OS juga harus mengelola proses dan mengetahui proses apa yang berjalan, proses berikutnya yang akan berjalan, dan

konteks program setiap proses. OS membutuhkan Process Control Block (PCB) untuk menyimpan informasi proses, dan ready queue, list of PCB untuk menandakan proses yang siap dijalankan berikutnya.

Getting Started

Dalam tugas ini anda tidak wajib menggunakan *Tree File System* pada tugas sebelumnya karena tidak berhubungan tersebut. File yang dibutuhkan ada di dalam **milestone3-kit.zip** yang akan diberikan oleh asisten:

- kernel.asm, berisi fungsi baru untuk mengimplementasi **TimerInterrupt**
- lib.asm, berisi prosedur baru yaitu **enableInterrupts** untuk membuat user program bisa menerima interrupt
- pcb.h, berisi struktur data PCB yang akan diimplementasi

III. Langkah Pengerjaan

Timer Interrupt

Jika diperhatikan kernel.asm mempunyai tiga prosedur baru, yaitu **`makeTimerInterrupt`**, **`timer_ISR`**, dan **`returnFromTimer`**.

`makeTimerInterrupt`: Membuat Timer untuk melakukan interrupt sebanyak 12 kali dalam 1 detik. Setiap interrupt akan memanggil prosedur timer_ISR.

`timer_ISR`: Menyimpan konteks program yang sedang berjalan (registers) ke dalam stack, lalu memanggil prosedur handleTimerInterrupt yang akan diimplementasi dalam kernel.c. timer_ISR akan memberikan pointer menuju stack. Prosedur handleTimerInterrupt akan menyimpan stack pointer tersebut ke struktur data dan memilih proses berikutnya untuk dijalankan.

`returnFromTimer`: Setelah proses yang akan dijalankan telah dipilih, maka prosedur returnFromTimer akan mempop stack milik proses berikutnya untuk me-*load* konteks program.

Lakukan langkah berikut untuk menguji Timer:

1. Panggil fungsi **makeTimerInterrupt** dalam main kernel.c sebelum shell
2. Untuk mengetes, implementasi prosedur **handleTimerInterrupt** untuk melakukan print “tic”, kemudian di akhir prosedur, panggil **returnFromTimer**.

```
void handleTimerInterrupt(int segment, int stackPointer);
```

```
void returnFromTimer(int segment, int stackPointer);
```

Untuk sementara, isi argumen returnFromTimer dengan argumen dari handleTimerInterrupt. Nantinya kalian akan mengisi dengan segmen dan stackPointer dari proses yang akan berjalan selanjutnya.

3. Jalankan program, jika berhasil maka shell kalian akan dibanjiri pesan "tic".

Struktur Data

1. Implementasi struktur data pcb.h pada file pcb.c.
2. (Opsional) Buatlah file testpcb.c untuk mengetes apakah struktur data kalian sudah terimplementasi dengan benar. (Opsional[2]) Tes program kalian dengan standar unit test yang kalian ketahui.

Implementasi Multiprogramming

1. **Setup**

   Define label MAIN pada kernel.c, include pcb.h, lalu panggil prosedur initializeProcStructures di fungsi utama (main). (modifikasi juga script compileOS agar mengcompile dan melink pcb.c)

2. **Starting program**

   Ubah prosedur **`executeProgram(char *fname, int segment)`** menjadi **`executeProgram(char *fname)`**, lalu dalam prosedurnya, buatlah agar variabel segment diisi dengan free segment yang didapat dari prosedur pcb.c. Jika dicompile, program seharusnya berjalan sama seperti sebelumnya.

   Untuk multiprogramming, pada implementasi executeProgram inisialisasi PCB untuk proses tersebut, masukkan ke ready queue, isi nama dengan filename program, isi status dengan *state* STARTING, isi segment, lalu set Stack Pointer ke 0xFF00.

   Terakhir, prosedur executeProgram tidak lagi jump ke instruksi program. Ganti pemanggilan prosedur launchProgram dengan initializeProgram yang ada di kernel.asm.

   ```
   void initializeProgram(int segment);
   ```

   Prosedur ini bertujuan untuk menyimpan konteks program.

3. **Prosedur handleTimerInterrupt**

   Implementasi prosedur handleTimerInterrupt agar Prosedur ini menyimpan segmen dan stackPointer ke PCB dari proses running, tandai statusnya dengan WAITING, lalu tambahkan ke tail dari ready queue. Kemudian *remove head* dari *ready queue*, tandai statusnya dengan RUNNING, set running variable ke PCB itu, lalu panggil prosedur returnFromTimer. Jika ready queue kosong, maka pilihlah proses idle.

4. **Terminate program**

   Ubahlah implementasi prosedur terminate dari tugas sebelumnya, sehingga tidak perlu memanggil shell lagi. Setiap suatu program diterminasi, free memory dari segmen tersebut, free PCB, set statusnya menjadi DEFUNCT, lalu buat infinite while loop.

   Note : prosedur handleTimerInterrupt juga harus mengelola proses yang statusnya DEFUNCT.

5. **Data Segment**

   Ada masalah kecil saat pemanggilan executeProgram dan terminate. Fungsi ini biasa dipanggil dengan system call (interrupt 0x21). Pada saat itu Data Segment (DS) register pointernya mengarah ke data segment system call, sementara global variable dari struktur data proses disimpan dalam kernel data segment. Karena itu DS register harus diarahkan terlebih dahulu sebelum mengakses global variable tersebut, dan setelah mengakses, DS register juga diarahkan kembali tempat semula.

   Karena itu pada kernel.asm disediakan prosedur:

```
void setKernelDataSegment();
void restoreDataSegment();
```

   Setiap pengaksesan global variable atau fungsi proses management, maka perlu diapit kedua prosedur di atas. Contohnya dapat dilihat sebagai berikut:

```
void setKernelDataSegment();
int segment = getFreeMemorySegment();
void restoreDataSegment();
```

   Ubah fungsi **executeProgram** dan **terminate** sehingga data segmentnya diapit dengan kedua prosedur tersebut.

6. **Enabling Interrupt**
   Anda juga perlu melakukan sedikit perubahan pada user program. Dalam 16-bit real mode, secara default interrupt di-disable. Karena itu di main user program perlu memanggil prosedur `enableInterrupts()`, yang implementasinya ada di lib.asm.

7. **Testing**
   Jika implementasi Anda sudah benar, kernel akan berjalan seperti tugas sebelumnya, hanya saja proses internal yang berjalan berbeda. Untuk menguji multiprogramming, buat userprogram Hello.

```
main()
{
    int i=0;
    int j=0;
    int k=0;
    enableInterrupts();
    for(i=0; i<1000; i++) {
        print("Hello\n\r\0");
        for(j=0; j<1000; j++) {
            for(k=0; k<1000; k++) {
            }
        }
    }
}
```

   Buat juga userprogram World, lalu eksekusi kedua program tersebut.

   Saat program berjalan, Anda bisa menjalankan fungsi shell juga seperti dir, yang outputnya akan tercampur dengan print dari user program.

Implementasi fungsi

1. **Prosedur showProcesses**
   Implementasi prosedur **`showProcesses`** sehingga bisa menampilkan nama proses yang sedang berjalan beserta dengan segmen dari proses.

   Prototype prosedur **showProcesses** adalah sebagai berikut:

```
void showProcesses();
```

Tambahkan pada *shell* Anda sehingga bisa menerima perintah ps untuk memanggil showProcesses.

2. **Fungsi kill**
   Implementasi fungsi **`kill`** untuk menghentikan proses dengan segmen tertentu. Tambahkan pada shell Anda sehingga bisa menerima perintah kill <nomor segmen>. Fungsi akan mengembalikan 1 jika proses berhasil dihentikan, dan -1 jika tidak ada proses yang berjalan pada segmen tersebut. *Shell* akan memberikan pesan bahwa proses berhasil dibunuh atau tidak.

   Prototype fungsi **kill** adalah sebagai berikut:

```
int kill(int segment);
```

IV. Bonus

Sekarang program *shell* akan berjalan secara konkuren dan tidak menghalangi *shell*, sementara *shell* pada umumnya justru *blocking*. Implementasikan dalam OS Anda agar anda dapat memilih apakah suatu eksekusi program akan berjalan konkuren atau tidak.

Contoh :

1. **`execute <nama program>`** : *shell* berhenti berjalan dan akan mengeksekusi program sampai selesai
2. **`execute <nama program> &`** : eksekusi program berjalan secara konkuren dan asynchronous, sehingga *shell* dapat menerima input.

V. Pengumpulan dan Deliverables

1. Buatlah sebuah **zip/rar** dengan nama **TB_M3_KX_YY**, dengan X adalah nomor kelas dan YY adalah nomor kelompok (2 digit). File zip/rar ini terdiri dari 2 folder sebagai berikut:
   - Folder **src**
   - Folder **doc**, berisi file laporan dengan format **Laporan_TB_M3_KX_YY.pdf** dengan penamaan seperti file yang dizip.

2. Laporan yang berisi
   - Cover
   - *Screenshot* hasil eksekusi program Hello dan World, dilanjutkan dengan prosedur showProcesses dan fungsi kill. Jika anda mengerjakan bonus, tampilkan *screenshotnya* dengan keterangan agar terlihat bahwa anda mengerjakan bonusnya.

- Pembagian tugas, dengan mencantumkan NIM dan Nama setiap anggota kelompok, dengan rincian sebagai berikut:
    a) Apa saja yang dikerjakan
    b) Perkiraan persentase pengerjaan pada tugas ini
- *Feedback* mengenai tugas ini

3. Deadline dari pengumpulan milestone ini adalah tanggal 18 April 2017 pukul 20:17:00 waktu server. File yang telah di-zip harap dikumpulkan ke milestone.if.itb.ac.id. **Keterlambatan pengumpulan akan menyebabkan pengurangan nilai**.

Referensi

- users.dickinson.edu/~braught/courses/cs354s10/proj/p1.pdf
- users.dickinson.edu/~braught/courses/cs354s10/proj/p2.pdf
- users.dickinson.edu/~braught/courses/cs354s10/proj/p3.pdf
- users.dickinson.edu/~braught/courses/cs354s10/proj/p4.pdf
- users.dickinson.edu/~braught/courses/cs354s10/proj/p5.pdf
- users.dickinson.edu/~braught/courses/cs354s10/proj/p6.pdf