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

Dipersiapkan oleh:
Asisten Laboratorium Sistem Terdistribusi

Didukung oleh:

START: 14 Februari 2017
END: 28 Februari 2017

I. Latar Belakang

Mengapa kita seringkali salah dalam berkomunikasi dengan orang lain? Mengapa kita sulit untuk mengerti pikiran orang? Hal ini terjadi karena kita hanya mengenal orang itu dari "kulit"-nya saja. Begitu kita mengenal apa yang ada di balik kulit itu (Anime, Idol, Fan Fiction, Drama Korea, dan komik Marvel), kita akan mudah mengerti dan menerima apa adanya.

Kalian tentunya sudah sering menemui sistem operasi tanpa mengenalinya. Pada tugas besar ini kami mengajak kalian untuk mengenal lebih dalam bagaimana sistem operasi bekerja, dan memotivasi kalian untuk melakukan eksplorasi untuk menemukan Pokemon Legendaris.

"Human" - Of Monsters And Men

When the words weigh heavy on the heart
I am lost and led only by the stars

Cage me like an animal
A crown with gems and gold
Eat me like a cannibal
Chase the neon throne

Breathe in, breathe out
Let the human in
Breathe in, breathe out
And let it in
Plants awoke and they slowly grow
Beneath the skin
So breathe in, breathe out
Let the human in

II. Deskripsi Tugas Besar

Dalam tugas besar ini, anda akan membangun Sistem Operasi 16-bit sederhana yang dijalankan di atas **Bochs Emulator**. Tugas besar ini mengandung file assembly, hanya saja pengembangan Sistem Operasi akan difokuskan di bagian C (**yeay ^_^**). Tugas besar bersifat incremental, dibagi dalam 3 milestone dengan rincian sebagai berikut.

1. Milestone 1 : OS Introduction & Booting
2. Milestone 2 : System Call, File System and Shell
3. Milestone 3 : Processes and Multiprogramming

Spesifikasi Milestone 2 dan Milestone 3 akan diberikan secara bertahap; Sementara itu, spesifikasi Milestone 1 dapat dilihat pada bab berikut.

III. Milestone 1 : OS Introduction & Booting

Tahap ini akan memperkenalkan bagaimana sebuah OS melakukan *boot*. Selain itu, anda akan membuat sebuah kernel kecil yang dapat melakukan print "Hello World" pada layar.

Spesifikasi Tugas

1. OS dapat melakukan *boot* dan menampilkan **Hello World** ke layar.
2. Mengimplementasi fungsi **printString**, dengan parameter karakter, baris, kolom, dan warna.

Booting

Saat komputer melakukan booting, bootstrap program pada BIOS akan menjalankan sektor pertama pada disk yang disebut ***bootloader***. *Bootloader* kemudian akan menjalankan kernel sistem operasi.

Tools

Tools yang dibutuhkan selama tugas besar ini adalah :

- **bochs**, x86 processor simulator, dapat di-install dengan menggunakan command:
  sudo apt-get install bochs bochs-x bochs-wx
- **bcc**, 16-bit C compiler
- **as86**, **ld86**, 16 bit assembler dan linker
- **gcc**

- **nasm**, Netwide Assembler
- **hexedit**, untuk mengedit file secara hexadecimal
- **dd**, standard low-level copying utility
- gedit/atom/text editor lainnya

IV. Langkah Pengerjaan

Getting Started

1. Download zip yang disediakan, lalu un-zip file tersebut. Di dalamnya akan terdapat beberapa file :
   - **bootload.asm**, assembly code untuk bootloader
   - **kernel.asm**, code yang akan dipakai kernel
   - **opsys.bxrc**, bochs configuration files
   - **test.img**, bootable 1.44 MB floppy image
2. Rename test.img menjadi floppya.img
3. Jalankan bochs dengan command

   bochs -f opsys.bxrc

   Lalu ketik 'c' pada shell bochs untuk melanjutkan eksekusi.

   Bochs akan berjalan sesuai konfigurasi file **opsys.bxrc.** Konfigurasi file akan mengatur driver, memory, dan file yang dijalankan bochs. Konfigurasi ini akan menginstruksikan bochs untuk melakukan booting terhadap floppya.img.

   Karena floppya.img masih kosong, maka bochs tidak akan mengeluarkan output apa pun.

Bootloader

Ketika PC pertama kali melakukan *boot*, PC akan mencari *sector* 0 dari *disk* dan melakukan boot pada sector tersebut. *Floppy disk* sebesar 1.44MB dengan sector size 512 bytes, mempunyai 2880 *sector*. Program *bootloader* akan diletakkan di sector 0 dengan size 512 bytes.

Karena bootloader hanya bisa menampung 512 bytes, maka bahasa yang digunakan adalah bahasa assembly. Karena kebaikan asisten, kami telah menyiapkan bootload.asm untuk digunakan oleh anda.

Terdapat tiga hal yang dilakukan oleh bootload.asm:

1. Mempersiapkan *stack* dan *register* ke *address* `0x10000`.
2. Membaca 10 sector dari sector 3-12 (sector 1 dan 2 akan dipakai untuk keperluan lain) dan dipindahkan ke address `0x10000`.
3. Jump pada memory `0x10000` dan mengeksekusi program pada memory tersebut.

Memory `0x10000` akan diisi oleh program kernel yang akan kalian buat. Sebelum menginstall bootloader, kalian harus meng-*assemble* terlebih dahulu bootloader.asm untuk menghasilkan sebuah *machine code*.

```
$ nasm bootload.asm
```

Untuk mengecek output file bootload yang dihasilkan, kalian dapat mengecek menggunakan hex editor. File bootload akan mengandung magic number **55 AA** yang menandakan bahwa itu adalah boot sector.

Disk Images

Untuk menginstall bootloader ke disk image, kita akan menggunakan tools bernama dd. Pertama hal yang perlu dilakukan adalah mengisi disk dengan angka 0.

```
$ dd if=/dev/zero of=floppya.img bs=512 count=2880
```

Command ini akan melakukan copy 2880 sector dengan size 512 bytes dari input file **/dev/zero** ke output file floppya.img. File input tersebut adalah special linux device yang berisi oleh angka 0 saja. Hasil dari command ini adalah floppya.img yang berisi 0.

Selanjutnya kita bisa menggunakan tools yang sama untuk menginstall bootloader:

```
$ dd if=bootload of=floppya.img bs=512 count=1 conv=notrunc seek=0
```

Parameter seek untuk menandakan sector mana hasil copy akan diletakkan di output file, conv=notrunc berarti output file tidak akan ditruncate, dan count=1 berarti dd hanya akan meng-copy file sebanyak 1 sector.

Sekarang file floppya.img sudah berisi bootloader, namun belum ada apapun selain bootloader pada file tersebut. Selanjutnya kalian akan mengisi memory `0x10000` dengan program kernel yang akan mencetak "Hello World".

Hello World Kernel

Kernel yang akan kalian buat memakai bahasa C, disimpan dengan nama **kernel.c**. Untuk melakukan print, kalian tidak bisa menggunakan library standar seperti `<stdio.h>`. Hal ini karena library seperti itu akan memanggil system call yang dimiliki operating system. Untuk OS yang baru kalian buat tentu saja belum berisi system call sama sekali.

Karena tidak bisa menggunakan library, jika ingin menuliskan sesuatu kalian harus menulis langsung ke video memory. Video memory terletak pada *address* `0xB8000` untuk text mode. Dalam text mode, suatu screen terdiri dari 25 line @ 80 karakter. Setiap karakter memakan 2 byte dalam video memory; 1 byte untuk ASCII code karakter itu sendiri dan 1 byte untuk warna.

Dengan demikian, cara untuk mencetak karakter 'A' dengan warna putih dan background hitam di baris ketiga adalah:

1. Hitung *address* relatif terhadap video memory : 80 * (3 baris -1) = 160
2. Kalikan 2 bytes/char : 160*2 = 320
3. Convert ke hex = 0x140
4. Hitung *address* absolut : `0xB8000`+`0x140` = `0xB8140`
5. Tulis `0x41` (ASCII code 'A') ke *address* `0xB8140`
6. Tulis `0x0F` (warna putih) ke *address* `0xB8141`

16-bit C compiler yang akan digunakan tidak bisa menulis langsung ke memory. Karena itu kami berikan kernel.asm dengan fungsi putInMemory, yang akan kalian panggil di kernel.c.

Fungsi `putInMemory` memiliki prototipe:

```
void putInMemory(int segment, int offset, char b);
```

Segment : most significant hex digit * 0x1000
Offset : Least 4 signicant hex digit
b : ASCII code dari karakter yang ingin ditulis

Untuk menulis karakter 'A' seperti di atas, maka pada **kernel.c** cukup memanggil fungsi **putInMemory** sebagai berikut:

putInMemory(0xB000, 0x8140, 'A')

Compile Kernel

Selain dalam UEFI boot, OS selalu memulai booting dalam mode 16-bit. Karena itu kernel yang kita buat harus dalam 16-bit *machine code*. Untuk itu kita harus menggunakan bcc sebagai compiler kode yang telah dibuat. Kelemahan dari bcc adalah masih memakai syntax C lama.

Modern OS akan melakukan mode *switching* dari 16-bit ke 32/64-bit setelah booting selesai. Namun, karena OS yang anda buat adalah 16-bit OS, maka anda tidak perlu memikirkan hal tersebut.

Lakukan kompilasi kernel:

```
$ bcc -ansi -c -o kernel.o kernel.c
```

Flag -c berarti tidak menggunakan library C yang bergantung pada system call. Flag -ansi menyuruh compiler untuk memakai standar ANSI syntax. Agar program kernel dapat memakai fungsi dari kernel.asm, maka anda perlu melakukan linking sebagai berikut:

- Assemble kernel.asm dengan command
  $ as86 -o kernel_asm.o kernel.asm
- Lakukan linking
  $ ld86 -o kernel -d kernel.o kernel_asm.o

Hasil akhirnya adalah program bernama kernel yang siap ditaruh di file **floppya.img**. Gunakan dd untuk mengcopy kernel ke sector 3:

```
$ dd if=kernel of=floppya.img bs=512 conv=notrunc seek=3
```

Sekarang jika bochs dijalankan, kernel akan dieksekusi dan melakukan print ke layar.

Shell Script

Melakukan kompilasi kernel memerlukan banyak command, karena itu kalian dapat menggunakan shell script untuk menjalankan command-command tersebut secara otomatis.

Buat file bernama **compileOS.bash** lalu isi dengan kode berikut di line pertama.

```
#!/bin/bash
```

Hal ini ditujukan agar *bash* dapat mengenali file tersebut sebagai *shell script*. Setelah itu, masukkan command-command kompilasi kernel ke *script*.

Setelah itu, untuk dapat menjalankan shell script, harus dilakukan perintah chmod terlebih dahulu:

```
$ chmod +x compileOS.bash
```

Terakhir, setiap akan melakukan kompilasi kalian cukup menjalankan **./compileOS.bash.**

printString

Implementasi fungsi **printString** dengan memanfaatkan fungsi **putInMemory**. Akan digunakan satu byte sebagai byte warna. Empat MSB (*most significant bit*) pada byte warna akan digunakan untuk warna background, sementara empat LSB (*least significant bit*) digunakan untuk warna text. Empat bit tersebut akan merujuk terhadap daftar kode warna sebagai berikut:

0	0000	black	8	1000	dark gray
1	0001	blue	9	1001	light blue
2	0010	green	A	1010	light green
3	0011	cyan	B	1011	light cyan
4	0100	red	C	1100	light red
5	0101	magenta	D	1101	light magenta
6	0110	brown	E	1110	yellow
7	0111	light gray	F	1111	white

Sebagai contoh: jika ingin membuat warna biru dengan *background* hitam maka kodenya adalah 0000 0001 = 0x01

V. Bonus

Setelah menulis karakter dan warna, kalian dapat membuat logo OS kalian sendiri dengan kombinasi karakter ASCII. Desain logo dibebaskan. Jadi saat bochs dinyalakan, OS akan langsung menampilkan boot logo.

VI. Pengumpulan dan Deliverables

1. Buatlah sebuah **zip/rar** dengan nama **TB_M1_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**, berisi file **kernel.c** dan **compileOS.bash**
   - Folder **doc**, berisi file laporan

2. Laporan yang berisi
   - Cover
   - Jawaban dari pertanyaan berikut :
     a) Apa perbedaan cara booting disk MBR dengan GPT? Apa cara boot yang dipakai dalam tugas ini?
     b) Apakah mungkin menginstall OS ini ke floppy disk atau disk lain? Jelaskan.
     c) Apa perbedaan booting secara real mode dan protected mode? Mode apa yang dipakai dalam tugas ini?
     d) Mengapa BIOS sekarang masih ada yang dieksekusi secara 16-bit?
     e) Mengapa magic number yang digunakan oleh bootloader adalah **55 AA**? Apa akibatnya jika magic number tersebut diubah nilainya?
   - Kesulitan saat mengerjakan (jika ada)
   - *Screenshot* hasil eksekusi program Hello World dan Bonus (jika dikerjakan)
   - 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 28 Februari 2017 pukul 20:18:00 waktu server. Teknis pengumpulan akan diberitahukan maksimal 48 jam sebelum *deadline* pengumpulan. **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