Tugas Besar
IF2230 - Sistem Operasi
Milestone 01 of ??
"The Awakening"
Pembuatan Sistem Operasi Sederhana
Booting, Kernel, File Program, System Call, Eksekusi Program

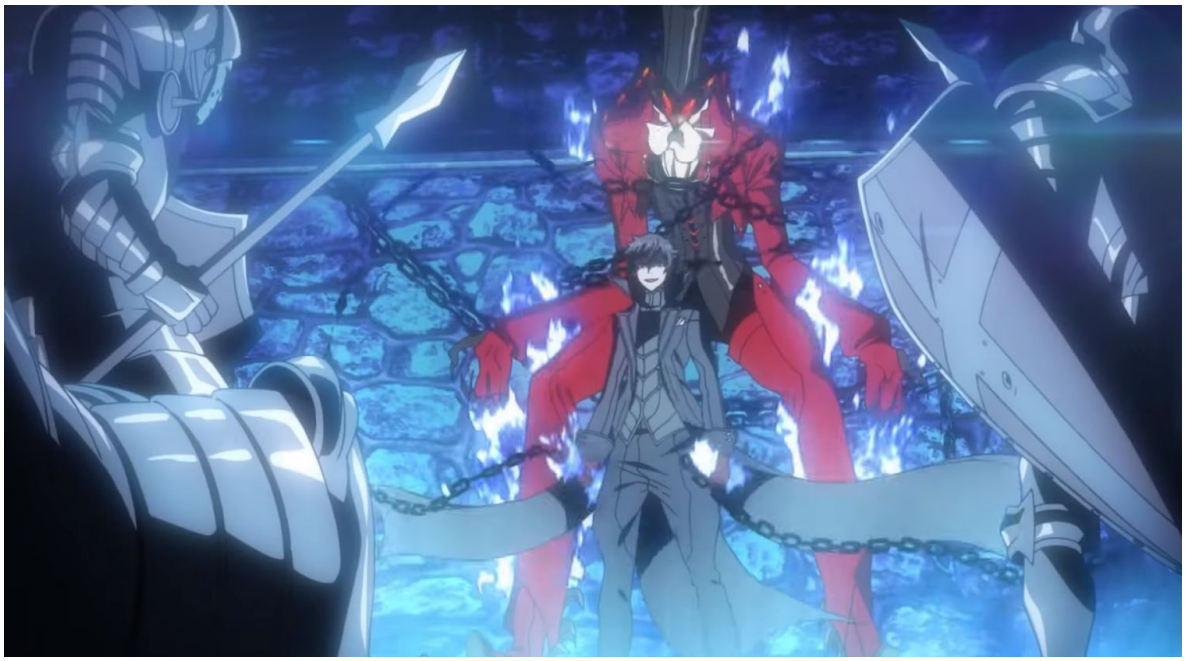
Dipersiapkan oleh :
Asisten Lab Sistem Terdistribusi

Didukung Oleh :

Waktu Mulai :
Rabu, 5 Februari 2020, 20.20 WIB

Waktu Akhir :
Jumat, 14 Februari 2020, 20.20 WIB

I. Latar Belakang



Hari ini adalah hari pertamamu bersekolah setelah pindah dari sekolahmu yang lama. Hujan di pagi hari memaksamu untuk berteduh dengan seorang siswa lain yang tidak kamu kenal hingga hujan berhenti. Kalian berdua langsung berlari ke sekolah, tanpa menyadari bahwa layar ponselmu tiba-tiba menyala dan menjalankan sebuah aplikasi dengan sendirinya. Setibanya di sana, kamu mendapati bahwa gedung yang berdiri di depanmu bukan lagi sebuah sekolah melainkan sebuah istana besar.

Kalian memberanikan diri kalian untuk memasuki istana tersebut, namun yang menanti di dalam adalah sepasukan penjaga berbaju zirah yang langsung sadar atas keberadaan kalian, lalu mulai mengejar kalian sambil berteriak, “Penyusup!!”. Kalian berdua pun berusaha kabur dari mereka, tetapi tepat saat kalian berbalik badan, kamu melihat bahwa kalian telah dikepung oleh para penjaga dari segala sisi. Tiba-tiba kedua lenganmu ditahan oleh dua orang penjaga dari kedua sisi. Kamu mencoba untuk membebaskan diri tetapi genggaman mereka terlalu kuat.

Pada saat itu, kamu mendengar sebuah suara di pikiranmu.

“Apakah kamu akan diam saja?”
“Hanya kematian yang akan menunggumu jika diam saja.”

Kamu pun berpikir, “Tidak, aku tidak akan membiarkannya berakhir seperti ini!” Seketika itu juga kamu merasakan rasa sakit yang luar biasa. Seluruh tubuhmu rasanya seperti sedang terbakar

oleh api. Tetapi, di saat yang sama kamu merasa sebuah kekuatan mulai memasuki dirimu. Kekuatan yang belum pernah kamu rasakan sebelumnya. Kamu kembali mendengar suara yang sama berkata,

“Baiklah, tunjukkan kemampuanmu padaku!”

Kamu mendapati bahwa pakaianmu telah berubah, dan terdapat “sesuatu” di belakangmu. Dengan kekuatan baru ini, kamu dapat mengalahkan para penjaga. Gunakan kekuatan baru mu untuk menemukan kelemahan para penjaga, kalahkan mereka, dan kaburlah dari istana mengerikan tersebut!.

II. Deskripsi Tugas

Pada praktikum ini, kalian akan membuat sebuah sistem menakjubkan bernama sistem operasi. Sistem operasi yang akan kalian buat akan berjalan pada sistem 16-bit yang nanti akan disimulasikan dengan *emulator* bochs. Tugas ini akan dibagi menjadi beberapa *milestone* dan inilah yang akan dikerjakan di *milestone* pertama

- Menyiapkan *disk image*
- Melakukan instalasi *bootloader*
- Menyiapkan *filesystem* sederhana
- Implementasi kernel sederhana
- Menjalankan sistem operasi
- Melakukan implementasi beberapa *syscall* dengan fungsi
    - Membaca dan menulis ke layar
    - Membaca dan menulis ke *disk*
    - Menjalankan program
    - Eksekusi program

III. Langkah Pengerjaan

Semua instruksi berikut ini akan mengasumsikan Anda menjalankan Linux. Untuk program-program yang sudah harus terinstal pada sistem operasi Anda adalah sebagai berikut:

- Netwide assembler (https://www.nasm.us/) untuk kompilasi program assembly
- Bruce C Compiler (https://linux.die.net/man/1/bcc) untuk kompilasi C 16 bit (GCC sudah tidak mendukung 16 bit yang murni)
- ld86 (https://linux.die.net/man/1/ld86) untuk melakukan *linking object code*
- Bochs (http://bochs.sourceforge.net/) sebagai emulator untuk menjalankan sistem operasi

Pada Ubuntu 18.04 berikut adalah perintah yang dapat digunakan untuk menginstal program-program di atas

```
sudo apt install nasm bcc bin86 bochs bochs-x
```

3.1. Persiapan *disk image*

Sebelum memulai, Anda membutuhkan sebuah *image* yang akan digunakan untuk menyimpan sistem operasi Anda. Untuk tugas ini akan dibuat sebuah *image* disket berukuran 1.44 megabyte. Untuk membuatnya bisa digunakan *command line utility* dd

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

3.2. *Bootloader*

Saat komputer pertama kali melakukan *booting*, BIOS pada komputer akan mencari program untuk dijalankan. Program yang pertama kali dijalankan ini biasa disebut *bootloader* dan tugasnya adalah melakukan *booting* ke sistem operasi.

Untuk tugas ini, file *bootload.asm* sudah disediakan dan tinggal dikompilasi dengan menggunakan nasm

```
nasm bootloader.asm -o bootloader
```

Kemudian bootloader dapat dimasukkan ke dalam *disk image* dengan perintah

```
dd if=bootloader of=system.img bs=512 count=1 conv=notrunc
```

3.3. Persiapan *File System*

Filesystem dari sistem operasi anda terdiri atas 2 sektor khusus yang telah dialokasikan sejak awal, yaitu sektor map pada sektor 1 dan sektor dir pada sektor 2.

Sektor map terdiri atas 512 byte dimana masing-masing byte menandakan apakah sektor terisi atau belum terisi pada filesystem. Anda dapat lihat dengan perintah hexedit map.img bahwa byte ke-0 hingga ke-12 bernilai 0xFF yang berarti bahwa sektor tersebut terisi.

Salin map.img ke system.img pada sektor ke-1 dengan perintah berikut:

```
dd if=map.img of=system.img bs=512 count=1 seek=1 conv=notrunc
```

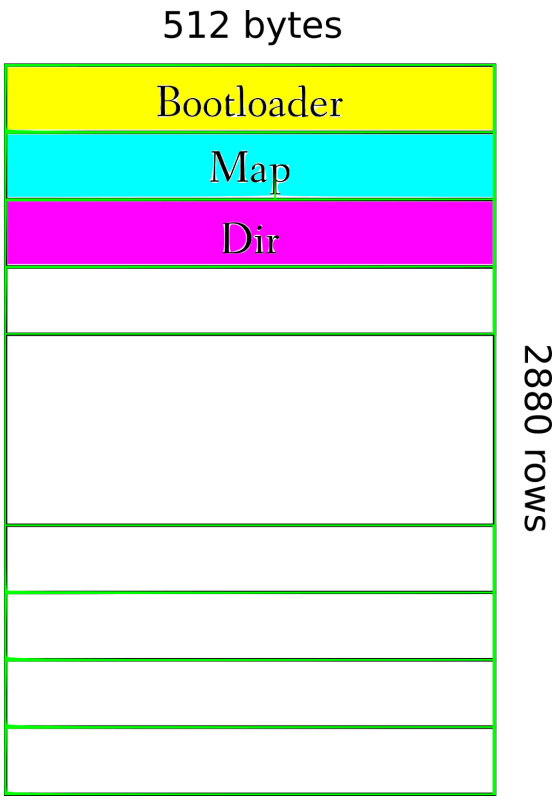
Sektor dir terdiri atas 16 baris berukuran 32 byte dimana 12 byte pertama merupakan filename dari sebuah file dan 20 byte terakhir merupakan sektor-sektor file tersebut. Dapat dilihat bahwa filesystem ini memiliki beberapa kelemahan, yakni:

- Filesystem hanya dapat menyimpan data 16 file.
- Setiap file hanya dapat memiliki filename sepanjang 12 byte.
- Setiap file hanya dapat memiliki 20 sektor.
- Sektor yang dapat dimiliki hanya sektor 0-255 karena 1 byte hanya dapat mengandung nilai-nilai tersebut.

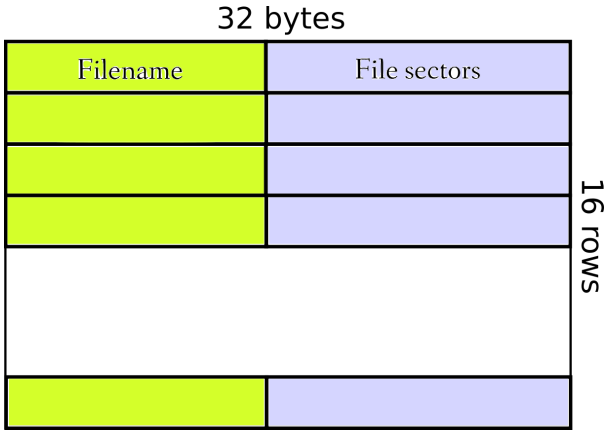
Anda juga dapat melihat dengan perintah dir.img dimana 12 byte baris pertama mengandung nilai `4B 45 52 4E 45 4C 00 00 00 00 00 00`
Jika diterjemahkan via tabel ASCII maka baris itu berisi tulisan “kernel” (karena kernel merupakan *file* pertama yang dimasukkan ke *disk*)

Salin dir.img ke system.img pada sektor ke-2 dengan perintah berikut:

```
dd if=dir.img of=system.img bs=512 count=1 seek=2 conv=notrunc
```



System.img Illustration



Dir Illustration

3.4. Pembuatan Kernel

Seperti sudah dijelaskan sebelumnya kode *bootloader* adalah kode yang pertama kali dijalankan oleh BIOS. Namun kode tersebut sangatlah kecil sehingga tidak mungkin memuat seluruh sistem operasi. Oleh karena itu, terdapat sebuah kernel terpisah yang nantinya akan dijalankan oleh *bootloader*.

Desain kernel untuk tugas besar ini dibuat sesederhana mungkin agar dapat lebih mudah dimengerti. Jika dilihat, terdapat file kernel.asm yang merupakan kode *assembly* dari kernel. Tugas anda adalah membuat file kernel.c yang merupakan bagian C dari kernel. Beberapa fungsi sudah disediakan oleh kode kernel.asm untuk digunakan pada file kernel.c. Jika dilihat pada bagian atas kernel.asm terdapat beberapa baris seperti berikut

```
global _putInMemory
```

Artinya, file kernel.asm mengimplementasi fungsi putInMemory yang nantinya bisa Anda gunakan. Daftar fungsi-fungsi tersebut beserta *signature*-nya adalah

- `void putInMemory (int segment, int address, char character)`
  Fungsi ini menulis sebuah karakter pada *segment* memori dengan *offset* tertentu

- `int interrupt (int number, int AX, int BX, int CX, int DX)`
  Fungsi ini memanggil sebuah *interrupt* dengan nomor tertentu dan juga meneruskan parameter AX, BX, CX, DX berukuran 16-bit ke *interrupt* tersebut

- `void makeInterrupt21()`
  Fungsi ini mempersiapkan tabel *interrupt vector* untuk memanggil kode Anda jika *interrupt 0x21* terpanggil

- `void handleInterrupt21 (int AX, int BX, int CX, int DX)`

Fungsi ini dipanggil saat terjadi *interrupt* nomor 0x21. Implementasi *interrupt* 0x21 pada kernel dilakukan di sini

- `void launchProgram(int segment)`
  Fungsi ini mengeksekusikan sebuah program pada segment memori tertentu

```
int main () {
  putInMemory(0xB000, 0x8000, 'H');
  putInMemory(0xB000, 0x8001, 0xD);
  putInMemory(0xB000, 0x8002, 'a');
  putInMemory(0xB000, 0x8003, 0xD);
  putInMemory(0xB000, 0x8004, 'i');
  putInMemory(0xB000, 0x8005, 0xD);

  while (1);
}

void handleInterrupt21 (int AX, int BX, int CX, int DX) {}
```

Kode di atas akan menghasilkan tulisan “Hai” di pojok kiri atas layar. Perhatikan bahwa:

- 0xB000 merupakan segment video memory
- 0x8000 adalah offset memori yang mendefinisikan karakter pada posisi (x=0, y=0), dimana (x=0, y=0) adalah posisi kiri atas layar.
- 0x8001 adalah offset memori yang mendefinisikan warna dari karakter pada posisi (x=0, y=0).
- Rumus umum untuk posisi alamat memori karakter adalah 0x8000 + (80 * y + x) * 2
- Rumus umum untuk posisi alamat memori karakter adalah 0x8001 + (80 * y + x) * 2
- while (1) digunakan agar kernel tidak reboot
- void handleInterrupt21 (int AX, int BX, int CX, int DX) harus ada karena dideklarasikan sebagai extern di kernel.asm (seperti di mesin karakter Alstrukdat). Fungsi ini akan diisi nanti.alamat

Setelah itu *kernel* sudah dapat di-*compile* dengan perintah berikut

```
bcc -ansi -c -o kernel.o kernel.c
nasm -f as86 kernel.asm -o kernel_asm.o
ld86 -o kernel -d kernel.o kernel_asm.o
dd if=kernel of=system.img bs=512 conv=notrunc seek=3
```

Arti dari perintah di atas:

- bcc akan mengkompilasi kernel.c menjadi *object code* kernel.o
- nasm akan mengkompilasi kernel.asm menjadi *object code* kernel_asm.o

- Kedua *object code* tersebut bisa dianggap potongan dari kode utuh kernel yang akan digabungkan dalam proses bernama *linking*. Perintah ld86 akan digunakan untuk tugas ini
- dd akan digunakan kembali untuk memasukkan kernel ke *disk image* di sektor 3

Untuk mempermudah sangat disarankan **membuat script** untuk melakukan kompilasi sistem operasi Anda (mulai dari tahap 1).

Sedikit Teori Penting dan Menarik

Sebuah *processor* x86 mempunyai beberapa mode, yang paling awal ada yaitu *real mode* yang paling simpel. Karakteristik dari mode ini adalah ukuran alamat memori sepanjang 20-bit (artinya hanya sekitar 1 Mebibyte data yang dapat dialamatkan). Selain itu pada mode ini tidak terdapat konsep proteksi memori sehingga jika tidak diatur dengan baik **program dapat saling menggunakan blok memori yang sama**(memori yang digunakan program melebihi tempat yang dialokasikan sehingga memakai blok memori program lain) sehingga menghasilkan *error random*. Untuk lebih detail bisa melihat referensi pada https://wiki.osdev.org/Real_Mode#Information

Selain mode *real* terdapat juga mode *protected* yang digunakan oleh semua sistem operasi modern. Namun mode ini jauh lebih kompleks dibanding mode *real* karena pada mode ini sistem operasi harus mengatur banyak hal sendiri seperti *global descriptor table* untuk mengatur segmen memori.

Seperti prosesor, tampilan grafis juga memiliki beberapa mode, salah satunya adalah mode teks. Pada mode ini (yang biasa disebut juga mode VGA 3) diperbolehkan penulisan langsung ke memori *text buffer* yang terletak pada alamat 0xB8000 ke atas. Mode ini menyediakan ukuran layar 80x25 karakter dengan dukungan untuk warna. Penulisan data ke alamat memori *text buffer* akan langsung ditampilkan di layar. Untuk lebih detail bisa melihat referensi pada https://wiki.osdev.org/Text_UI

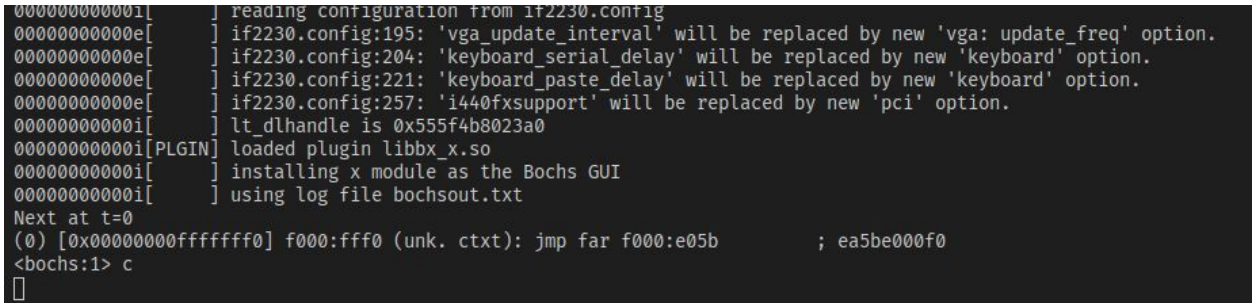
Jika dilihat pada potongan kode di atas, fungsi putInMemory menerima *segment* dan *offset* yang akan dikalkulasikan dengan rumus (segment << 4) + offset. Hal ini diperlukan untuk mengatasi ukuran register yang hanya 16 bit sedangkan alamat memori berukuran 20 bit. Untuk lebih detail bisa melihat referensi pada https://wiki.osdev.org/Real_Mode#Memory_Addressing

3.5. Menjalankan Sistem Operasi

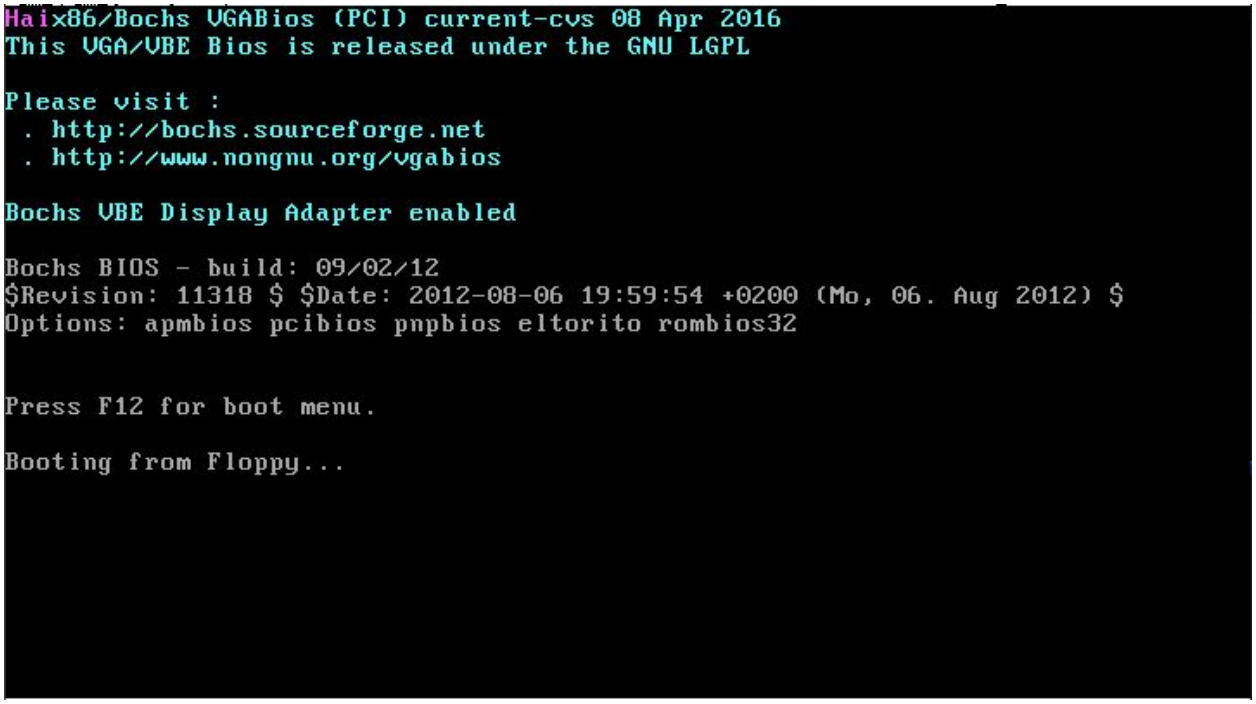
Untuk menjalankan sistem operasi yang Anda buat, akan digunakan emulator bochs. Untuk menjalankannya dibutuhkan sebuah file konfigurasi yang sudah diberikan dengan nama if2230.config

```
bochs -f if2230.config
```

Setelah itu akan muncul sebuah *prompt*. Ketik ‘c’ (artinya *continue*) dan enter pada *prompt* tersebut.



Berikutnya akan muncul sebuah *window* baru yang akan langsung menjalankan sistem operasi Anda (bisa dilihat dari tulisan “Hai” di pojok kiri atas)



3.6. Implementasi *interrupt* 0x21

Seperti pada semua sistem operasi, sistem operasi yang Anda buat juga harus mempunyai *syscall*. Pada sistem operasi ini *syscall* diimplementasikan dengan menggunakan *interrupt* 0x21.

```
/* Ini deklarasi fungsi */
void handleInterrupt21 (int AX, int BX, int CX, int DX);
void printString(char *string);
```

```
void readString(char *string);
void readSector(char *buffer, int sector);
void writeSector(char *buffer, int sector);
void readFile(char *buffer, char *filename, int *success);
void clear(char *buffer, int length); //Fungsi untuk mengisi buffer dengan 0
void writeFile(char *buffer, char *filename, int *sectors);
void executeProgram(char *filename, int segment, int *success);

int main() {
  makeInterrupt21();
  while (1);
}

void handleInterrupt21 (int AX, int BX, int CX, int DX){
  switch (AX) {
    case 0x0:
      printString(BX);
      break;
    case 0x1:
      readString(BX);
      break;
    case 0x2:
      readSector(BX, CX);
      break;
    case 0x3:
      writeSector(BX, CX);
      break;
    case 0x4:
      readFile(BX, CX, DX);
      break;
    case 0x5:
      writeFile(BX, CX, DX);
      break;
    case 0x6:
      executeProgram(BX, CX, DX);
      break;
    default:
      printString("Invalid interrupt");
  }
}
```

Tugas Anda adalah melengkapi implementasi tiap fungsi yang ada.

3.6.1. Implementasi printString dan readString

Kedua fungsi ini sangatlah sederhana dan mudah diimplementasikan. Dalam implementasinya hanya perlu menggunakan fasilitas yang disediakan oleh BIOS yang dapat dipanggil via *interrupt*. *Interrupt* yang dapat digunakan adalah *interrupt* 0x10 (tulis) dan 0x16 (baca). Untuk penggunaannya dapat melihat referensi di bawah

Sedikit Teori Penting dan Menarik [1]

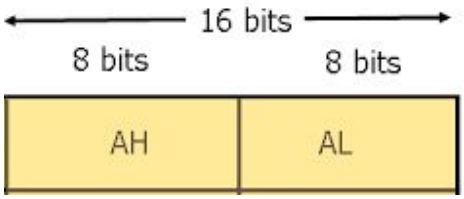
BIOS interrupt calls adalah fasilitas yang diberikan sistem operasi untuk memanggil Basic Input/Output System (BIOS) software pada komputer. BIOS interrupt calls dapat melakukan kontrol langsung pada hardware atau fungsi I/O yang diminta oleh program atau mengembalikan informasi terkait sistem.
Pada arsitektur x86, pemanggilan interrupt dilakukan dengan fungsi berikut.

```
interrupt(int number, int AX, int BX, int CX, int DX);
```

`int number` di sini mengacu pada nomor fungsi interrupt yang akan dipanggil misal interrupt 13. Berikut beberapa services yang dapat dilakukan melalui interrupt 13.

Tabel Interrupt vector 13h (Low Level Disk Services) [2]

AH	Description
00h	Reset Disk Drives
01h	Check Drive Status
02h	Read Sectors
03h	Write Sectors



Pada OS yang kalian buat, sebuah integer akan memiliki besar 16 bit. Sebuah integer 16 bit sendiri dapat dibagi menjadi dua bagian 8 bit yaitu AH dan AL. Untuk pemanggilan interrupt 13, parameter kedua yaitu `int AX` akan terbagi menjadi dua yaitu AH dan AL. Pada contoh kali ini, dibutuhkan kasus pemanggilan read sector yang akan digunakan untuk membaca 1 buah sector. Pada dokumentasi int 13 (dapat ditemukan pada reference [2]), untuk memanggil read sector, nilai dari AH = 02h, sedangkan nilai dari AL = jumlah sektor yang akan dibaca dalam hexadecimal. Sehingga parameter AX akan berisi nilai 0x0201 yang menunjukkan gabungan dari AH=02h dan AL=01h. Secara umum, penggunaan parameter AX, BX, CX, dan DX akan disesuaikan dengan kebutuhan fungsi interrupt yang akan kalian panggil.

3.6.2. Implementasi readSector dan writeSector

Implementasikan void readSector(char *buffer, int sector) dengan kode berikut:

```
interrupt(0x13, 0x201, buffer, div(sector, 36) * 0x100 + mod(sector, 18) +
1, mod(div(sector, 18), 2) * 0x100);
```

Keterangan:
Pada kode di atas, terdapat fungsi mod dan fungsi div. Fungsi ini harus kalian implementasikan sendiri.
Interrupt 0x13 dapat digunakan untuk berbagai macam hal, namun pada tugas ini hanya akan kita gunakan untuk membaca dan menulis sector. Interrupt tersebut memiliki beberapa argumen:

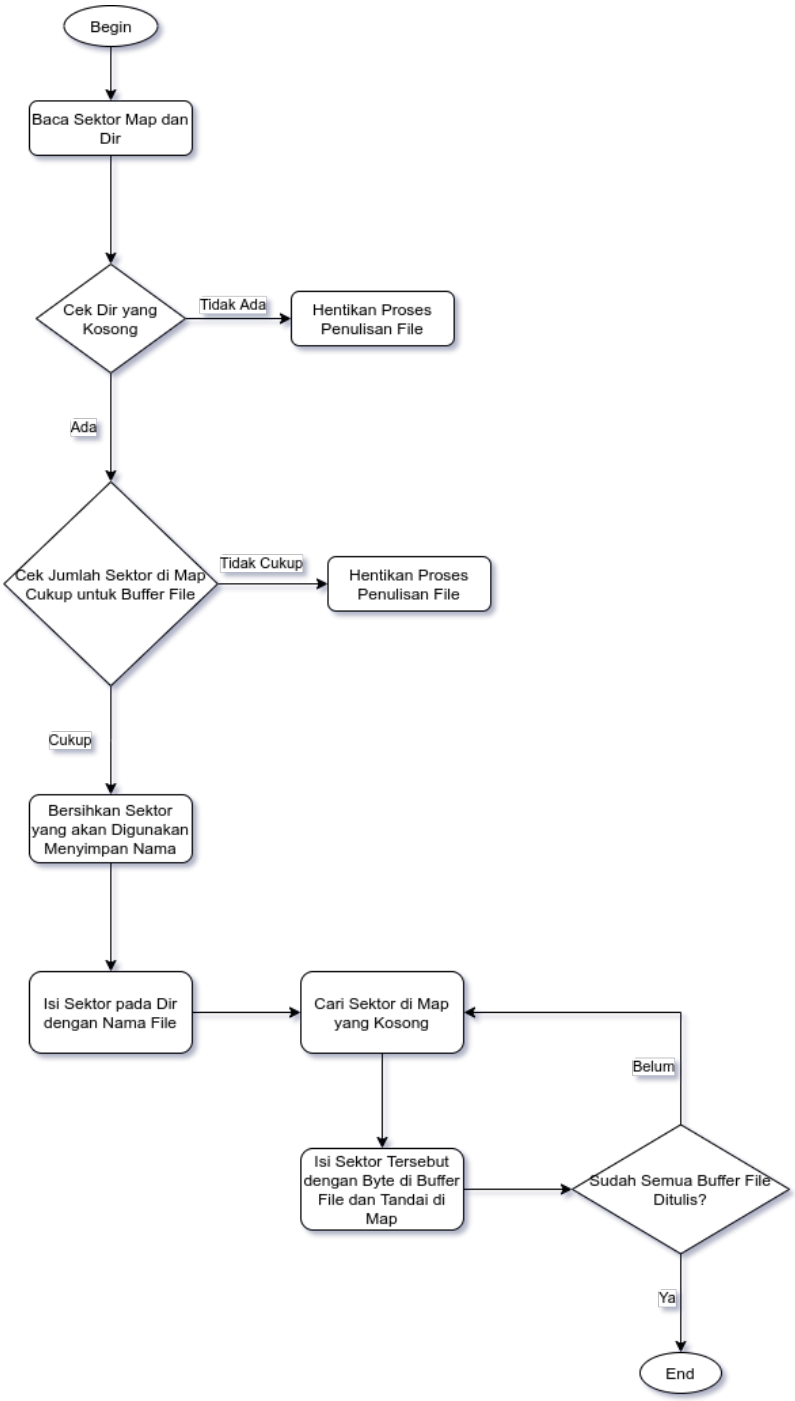
1. Argumen kedua terbagi menjadi dua bagian, pada kasus di atas 0x02 yang merupakan fungsi yang akan digunakan (pada kasus ini fungsi untuk membaca sector) dan 0x01 yang merupakan`jumlah sektor yang akan dibaca.
2. Argumen ketiga adalah lokasi dimana isi sektor akan disalin.
3. Argumen keempat adalah *cylinder* dan *sector. Sector* yang dimaksud adalah nomor *sector* pada satu *track*. Karena pada floppy, 1 *track* memiliki 18 *sector*, maka pada kode di atas, *sector* diisi dengan mod(sector, 18) + 1. *Cylinder* yang dimaksud adalah nomor *cylinder* pada satu *track.* Karena 1 *cylinder* memiliki 2 *head* yang masing-masing ada 18 *sector* dengan total 36 *sector* per *cylinder*, pada kode di atas *cylinder* diisi dengan
4. Argumen kelima adalah *head* dan *drive*. Pada kode di atas, *head* nya adalah mod(div(sector, 18), 2) karena *head* hanya dapat bernilai 0 atau 1 dengan masing-masing memiliki 18 *sector*. dan drive nya adalah 0 (drive A:).

Lalu implementasikan void writeSector(char *buffer, int sector) yang mirip seperti readSector(char *buffer, int sector), tetapi fungsi yang digunakan adalah 0x03 (Argumen kedua interrupt menjadi 0x301). Untuk informasi lebih lanjut mengenai interrupt 0x13, bisa dilihat di [2]. Untuk informasi lebih lanjut mengenai floppy yang digunakan pada tugas ini, bisa dilihat di [15]

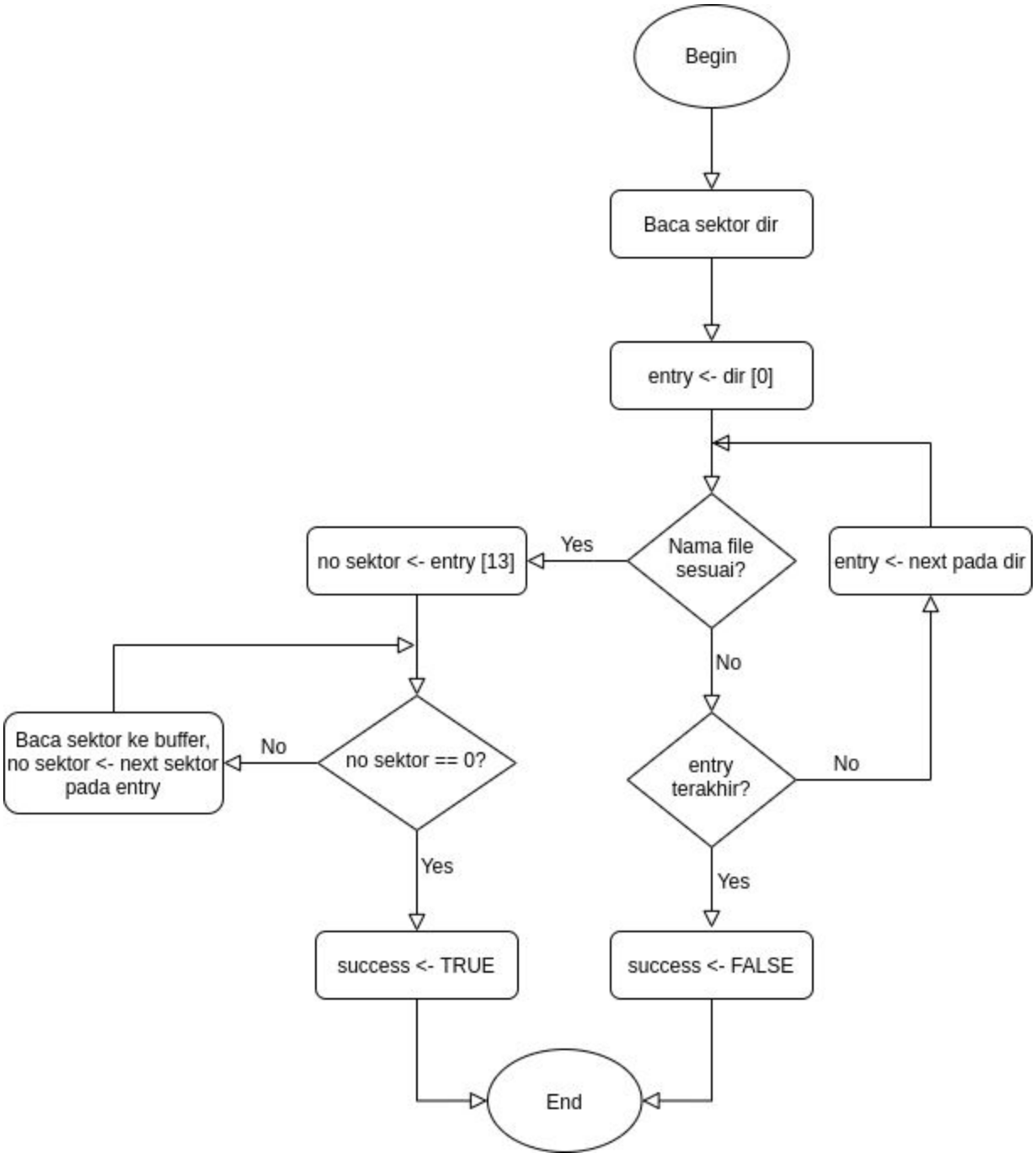
3.6.3. Implementasi readFile dan writeFile

Untuk implementasi readFile dan writeFile perhatikan diagram alir berikut

3.6.3.1 Flowchart Write File



3.6.3.2 Flowchart Read File



3.6.4. Implementasi executeProgram

Fungsi `void executeProgram(char *filename, int segment, int *success)` akan menjalankan tahap-tahap berikut

- Mengalokasikan sebuah *buffer* (ukuran lihat bagian *file system*)
- Buka file dengan fungsi *readFile*

- Jika sukses, gunakan *loop* untuk menyalin tiap byte kode dengan fungsi `putInMemory(segment, i, buffer[i])`
- Panggil fungsi `launchProgram(segment)` (fungsi ini telah didefinisikan di kernel.asm) untuk menjalankan program di segment yang telah terisi

3.7. Pengujian

Untuk keperluan penilaian, akan diberikan sebuah program bernama milestone1. Program ini harus Anda jalankan dengan memanggil fungsi executeProgram. Untuk memasukkan program ke *disk*, gunakan program loadFile

```
gcc loadFile.c -o loadFile
./loadFile milestone1
```

Program tersebut akan memanggil fungsi *writeFile* dan akan menulis key.txt di *disk* sistem operasi. Tambahkan sedikit *logic* di program *main* kernel Anda untuk memilih antara menjalankan program atau membaca file key.txt

3.7. Bonus

Bonus untuk milestone ini (diurutkan berdasarkan kesulitan dan skor)

- Membuat boot logo
- Mengerti cara kerja *interrupt* di sistem operasi Anda
- Membuat program sederhana yang terpisah dan bekerja dengan menggunakan *syscall* yang disediakan Sistem Operasi Anda. Gunakan fungsi *interrupt* pada program yang Anda buat. Untuk menggunakan fungsi *interrupt,* compile lib.asm dengan NASM lalu *link* dengan ld86
- Mengerti cara kerja kode di kernel.asm

IV. Pengumpulan dan Deliverables

1. Untuk tugas ini Anda diwajibkan menggunakan *version control system* **git** dengan menggunakan sebuah *repository* ***private*** di github (gunakan surel *student* agar gratis)

2. Setiap kelompok akan diberikan *repository* dari asisten (yang nanti akan diberitahukan pembagiannya)

3. Walaupun *commit* tidak dinilai, namun diharapkan melakukan *commit* yang wajar (tidak semua kode satu *commit*)

4. File yang harus terdapat pada *repository* adalah file-file *source code* dan *script* (jika ada) sedemikian rupa sehingga jika diunduh dari github dapat dijalankan. Dihimbau untuk tidak memasukkan *binary* atau *image* hasil kompilasi ke *repository*

5. Isilkan nama kelompok dan anggotanya pada link berikut s.id/daftar-kelompok-os, paling lambat tanggal 5 Februari 2020.

6. **Mulai** Rabu, 5 Februari 2020, 20.20 WIB waktu server.
   Deadline Jumat, 14 Februari 2020, 20.20 WIB waktu server.
   Setelah lewat waktu *deadline*, perubahan kode akan dikenakan pengurangan nilai

7. Teknis pengumpulan akan diberitahukan sekitar 48 jam sebelum *deadline* pengumpulan

8. Kami akan menindaklanjuti segala bentuk kecurangan yang terstruktur, masif, dan/atau sistematis

9. Diharapkan untuk mengerjakan sendiri terlebih dahulu sebelum mencari sumber inspirasi lain (Google, maupun teman anda yang sudah bisa). Percayalah jika menemukan sendiri jawabannya akan merasa bangga dan senang.

10. Dilarang melakukan kecurangan lain yang merugikan peserta mata kuliah IF2230.

11. Jika ada pertanyaan atau masalah pengerjaan harap segera mengirimkan surel ke milis mata kuliah IF2230 Sistem Operasi.

V. Tips

1. Bcc tidak menyediakan *check* sebanyak gcc sehingga ada kemungkinan kode yang Anda buat berhasil *compile* tapi *error*. Untuk mengecek bisa mengcompile dahulu dengan gcc dan melihat apakah *error*
2. Untuk melihat isi dari *disk* bisa digunakan utilitas hexedit
3. Walaupun kerapihan tidak dinilai langsung, kode yang rapi akan sangat membantu saat *debugging*
4. Fungsi-fungsi dari *stdc* yang biasa Anda gunakan seperti mod, div, strlen, dan lainnya tidak tersedia di sini. Anda harus membuatnya sendiri, terutama mod dan div yang akan sangat berguna

VI. Referensi Tambahan

6.1. Interrupt

[1] https://en.wikipedia.org/wiki/BIOS_interrupt_call
[2] http://www.oldlinux.org/Linux.old/docs/interrupts/int-html/int-13.htm

6.2. Referensi Perintah

6.2.1 NASM

```
nasm -f <format output> <file input> -o <file output>
```

Dalam tugas ini format output yang digunakan adalah as86 (sebuah format *object code* sederhana

6.2.2 ld86

```
ld86 -o <output> -d <object code untuk dilink>
```

Parameter -d menyatakan bahwa hasil *output* tidak memiliki *header* (digunakan pada sistem operasi yang lebih kompleks)

6.2.3 BCC

```
bcc -ansi -c -o <output> <input>
```

Parameter -ansi menyatakan versi C yang digunakan (ansi c) dan -c menyatakan untuk meng-*compile* ke *object code*

6.3. Bacaan Menarik

[3] https://www.eeeguide.com/8086-interrupt/
[4] https://www.ques10.com/p/10876/explain-the-interrupt-structure-of-8086-processo-1/
[5] https://wiki.osdev.org/Text_mode
[6] https://wiki.osdev.org/Text_Mode_Cursor
[7] https://wiki.osdev.org/Text_UI
[8] https://wiki.osdev.org/Real_Mode
[9] https://users.cs.fiu.edu/~downeyt/cop3402/absolute.html
[10] https://en.wikipedia.org/wiki/Intel_8086#Registers_and_instructions
[11] https://www.includehelp.com/embedded-system/types-of-registers-in-the-8086-microprocessor.aspx
[12] https://wiki.osdev.org/VGA_Hardware
[13] https://cs.nyu.edu/courses/fall03/V22.0201-001/combining.html
[14] https://en.wikibooks.org/wiki/X86_Assembly/NASM_Syntax
[15] https://wiki.osdev.org/Floppy_Disk_Controller