Spesifikasi Tugas Besar - Milestone 1
IF2230 - Sistem Operasi

"The Journey Continues"

Pembuatan Sistem Operasi Sederhana
Booting, Kernel, File Program, System Call, Eksekusi Program

Dipersiapkan oleh :
Asisten Lab Sistem Terdistribusi

Didukung Oleh :

Waktu Mulai :
Senin, 25 Februari 2019, 18.00.00 WIB

Waktu Akhir :
Senin, 4 Maret 2019, 12.59.59 WIB

I. Latar Belakang

Selamat, kalian telah menyelesaikan tugas pertama kalian sebagai *prerequisite* dari petualangan yang akan kalian lakukan kali ini! Kalian pun mulai menjelajahi *The Abyss* dengan bantuan alat sonar berbasis Linux yang telah kalian buat.

50m... 100m... 150m... tak terasa kalian sudah menjelajah cukup dalam. Kalian merasa kelelahan dan akhirnya memutuskan untuk rehat sejenak. Kalian pun merasa kehausan dan ketika kalian merogoh ransel... Sial! Setelah kalian ingat-ingat, ternyata kalian lupa membawa bekal air mineral ketika melewati Indomerat yang ada di pinggir jalan ketika kalian menuju mulut dari *The Abyss*. Lebih parahnya lagi, kini kalian berada di suatu titik yang mana tidak ada sama sekali sumber air yang mengalir. Beruntung, kalian menemukan sebuah kios yang menjual air mineral, tetapi sayangnya kalian tidak dapat menyampaikan apa yang kalian inginkan kepada penjual kios karena mereka bukan manusia dan mereka tidak berbicara menggunakan bahasa kalian. Karena kalian adalah petualang ulung, kalian tidak lupa membawa translator. Akan tetapi, translator tersebut berjalan di atas OS 16-bit yang tentunya sudah tidak zaman lagi.

Sebagai petualang yang telah belajar bagaimana sistem operasi bekerja, kalian harus membuat sistem operasi menggunakan bahasa kuno ‘C’ yang mampu menjalankan translator tersebut dan mendapatkan hasil translasi yang dapat kalian gunakan untuk berkomunikasi dengan penjaga kios. Kalian memiliki waktu satu minggu untuk bertahan hidup dan membuat OS tersebut.

II. Deskripsi Tugas

Dalam tugas besar ini, Anda akan membuat sebuah sistem operasi 16-bit sederhana. Sistem operasi Anda akan dijalankan di atas bochs, sebuah emulator yang sering digunakan untuk pembuatan dan debugging sistem operasi sederhana.

Tugas besar ini bersifat inkremental dan terbagi atas tiga milestone dengan rincian sebagai berikut:

1. **Milestone 1: Booting, Kernel, File System, System Call, Eksekusi Program**
    1. Menyiapkan image floppy disk
    2. Melakukan instalasi bootloader
    3. Menyiapkan file system
    4. Membuat kernel awal
    5. Melakukan kompilasi kernel
    6. Menjalankan sistem operasi
    7. Melakukan implementasi interrupt 0x21
        i. Menyiapkan interrupt 0x21
        ii. Implementasi fungsi syscall printString, readString
        iii. Implementasi fungsi syscall readSector, writeSector
        iv. Implementasi fungsi syscall readFile, writeFile
        v. Implementasi fungsi syscall executeProgram
    8. Mengeksekusi sebuah program
    9. **Bonus**
        i. Membuat logo OS yang ditampilkan setelah booting
        ii. Membuat program kalkulator sederhana (memiliki operasi tambah, kurang, kali, bagi, modulo integer)
2. **Milestone 2: File System dengan Direktori, Shell**
3. **Milestone 3: Process, Multiprogramming**

III. Langkah Pengerjaan

1. Persiapan image floppy disk

Di archive kit tugas, terdapat sebuah file floppya.img. File tersebut adalah sebuah image floppy disk 1.44 megabyte yang akan dibaca oleh bochs emulator untuk menjalankan OS Anda.

Pertama supaya keadaan disk image Anda bersih, Anda harus menghapus (mengisi dengan byte bernilai 0) seluruh isi disk image tersebut dengan perintah berikut:

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

2. Instalasi bootloader

Saat komputer melakukan booting, program bootstrap pada BIOS akan menjalankan sektor ke-0 pada disk yang diisi dengan program bootloader. Bootloader kemudian akan menjalankan kernel sistem operasi.

File bootload.asm sudah tersedia di archive kit tugas Anda. Hal yang harus Anda lakukan adalah melakukan assembly file asm tersebut dengan program nasm (gunakan perintah **`sudo apt-get install nasm`** jika belum terinstall pada sistem Anda).

```
nasm bootload.asm
```

Program nasm akan menghasilkan sebuah executable bernama bootload. Program bootloader tersebut pada dasarnya membaca data pada sektor 3-12 floppy disk Anda dimana kernel Anda akan berada (sektor 1 dan 2 untuk file system), menyalinkannya ke address 0x10000, lompat ke address 0x10000, dan mengeksekusi program pada address tersebut, yaitu kernel yang barusan disalin.

Agar program bootloader jalan, Anda harus menyalin file bootload tersebut ke sektor ke-0 file floppya.img dengan perintah berikut:

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

3. Persiapan file system

File system 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 file system. Anda dapat lihat dengan perintah hexedit map.img bahwa byte ke-0 hingga ke-0 hingga ke-12 bernilai 0xFF yang berarti bahwa sektor tersebut terisi.

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

```
dd if=map.img of=floppya.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 file sistem ini memiliki beberapa kelemahan, yakni:

- file system 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 hexedit dir.img dimana 12 byte baris pertama mengandung nilai **4B 45 52 4E 45 4C 00 00 00 00 00 00**

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

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

4. Pembuatan kernel awal

Bootloader pada bagian sebelumnya sudah masuk ke disk, akan tetapi kernel sistem operasi yang akan dijalankan oleh bootloader masih belum ada. Pertama, Anda harus membuat terlebih dahulu source code kernel.c.

Anda bisa melihat pada kernel.asm telah terdapat beberapa fungsi yang diimplementasi pada bagian assembly kernel sehingga dapat dipakai pada bagian C dari kernel, yaitu:

- **`void putInMemory (int segment, int address, char character)`**
  Fungsi ini menulis sebuah karakter pada segment memori dengan offset address 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 interrupt vector untuk interrupt nomor 0x21. Fungsi ini harus dipanggil sebelum interrupt nomor 0x21 dapat ditangani oleh kernel.

- **`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.

Untuk sementara Anda cukup membuat kernel yang sederhana saja untuk mengetes apakah sistem operasi Anda dapat dijalankan. Fungsi yang digunakan hanya fungsi putInMemory untuk menulis karakter ke video memory. Berikut source code-nya:

```
int main () {
   putInMemory(0xB000, 0x8000, 'K');
   putInMemory(0xB000, 0x8001, 0xD);
   putInMemory(0xB000, 0x8002, 'e');
   putInMemory(0xB000, 0x8003, 0xD);
   putInMemory(0xB000, 0x8004, 'r');
   putInMemory(0xB000, 0x8005, 0xD);
   putInMemory(0xB000, 0x8006, 'n');
   putInMemory(0xB000, 0x8007, 0xD);
   putInMemory(0xB000, 0x8008, 'e');
   putInMemory(0xB000, 0x8009, 0xD);
```

```
   putInMemory(0xB000, 0x800A, 'l');
   putInMemory(0xB000, 0x800B, 0xD);
   putInMemory(0xB000, 0x800C, '!');
   putInMemory(0xB000, 0x800D, 0xD);
   while (1);
}

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

Kode di atas akan mencetak tulisan '**Kernel!**' dengan warna magenta di posisi kiri atas layar. Perhatikan bahwa:

- **0xB000** merupakan segment video memory
- **0x8000** adalah alamat memori yang mendefinisikan karakter pada posisi (x=0, y=0), dimana (x=0, y=0) adalah posisi kiri atas layar.
- **0x8001** adalah alamat 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 linkage external di kernel.asm. Fungsi ini akan diisi nanti.

5. Kompilasi kernel

Sekarang Anda akan melakukan kompilasi kernel. Karena sistem operasi yang kalian akan buat merupakan sistem operasi 16-bit, maka untuk kompilasi akan digunakan bcc, sebuah compiler C 16-bit. Selain kompilasi kernel, Anda juga harus melakukan assembly kernel.asm dengan as86, dan linking kernel bagian assembly dan bagian C dengan ld86 (gunakan perintah **`sudo apt-get install bin86`** jika belum terinstall pada sistem Anda). Setelah itu, hasil linking yaitu program jadi kernel disalin ke floppy disk ke sektor ke-3 seperti pada bagian-bagian sebelumnya. Perintah-perintah yang digunakan adalah sebagai berikut:

```
bcc -ansi -c -o kernel.o kernel.c
as86 kernel.asm -o kernel_asm.o
ld86 -o kernel -d kernel.o kernel_asm.o
```

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

Anda harus melakukan langkah 1, 2, 3, dan 5 setiap kali Anda ingin mengubah kode dan melakukan kompilasi ulang kernel. Oleh karena itu, Anda disarankan untuk membuat sebuah script bash compileOS.sh yang berisi perintah-perintah pada setiap langkah tersebut.

6. Menjalankan sistem operasi

Untuk menjalankan sistem operasi Anda pada emulator bochs (gunakan perintah **`sudo apt-get install bochs bochs-x bochs-wx`** jika belum terinstall pada sistem Anda), diperlukan sebuah file konfigurasi opsys.bxrc. File konfigurasi tersebut akan mengatur driver, memory, dan file yang dijalankan bochs, dan juga menginstruksikan bochs untuk melakukan 6 booting menggunakan file disk image floppya.img. Gunakan perintah berikut untuk menjalankan sistem operasi Anda:

```
bochs -f opsys.bxrc
```

Ketik ‘c’ dan klik enter jika layar hitam. Jika pengerjaan Anda benar, maka pada posisi kiri atas layar terdapat tulisan “**Kernel!**” berwarna magenta.

7. Melakukan implementasi interrupt 0x21

Seperti pada tugas kecil sebelumnya, sistem operasi yang Anda buat juga memiliki *system call*. *System call* pada sistem operasi Anda dipanggil melalui mekanisme interrupt, tepatnya interrupt dengan nomor 0x21.

a. Menyiapkan interrupt 0x21

Sebelum interrupt 0x21 dapat ditangani dan dipanggil, terlebih dahulu interrupt vectornya harus dipersiapkan terlebih dahulu. Anda juga harus menyiapkan fungsi handleInterrupt21. Parameter AX akan digunakan sebagai penentu syscall yang dipanggil. Selain itu, beberapa fungsi dan konstanta juga harus dideklarasikan terlebih dahulu di awal source code. Untuk melakukan hal-hal tersebut, ganti kode kernel Anda sebelumnya dengan kode berikut:

```
#define MAX_BYTE 256
#define SECTOR_SIZE 512
```

```
#define MAX_FILES 16
#define MAX_FILENAME 12
#define MAX_SECTORS 20
#define DIR_ENTRY_LENGTH 32
#define MAP_SECTOR 1
#define DIR_SECTOR 2
#define TRUE 1
#define FALSE 0
#define INSUFFICIENT_SECTORS 0
#define NOT_FOUND -1
#define INSUFFICIENT_DIR_ENTRIES -1
#define EMPTY 0x00
#define USED 0xFF

void handleInterrupt21 (int AX, int BX, int CX, int DX);
void printString(char *string);
void readString(char *string);
int mod(int a, int b);
int div(int a, int b);
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);
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");
    }
}
```

b. Implementasi fungsi syscall printString, readString

Implementasikan **void printString(char *string)** dimana **string** adalah sebuah string NUL-terminated yang ingin dicetak. Berbeda dengan sebelumnya dimana karakter 8 dicetak langsung ke video memory, sekarang Anda akan menggunakan interrupt nomor 0x10 yang menyimpan posisi karakter terakhir dan memajukan karakter ketika dicetak secara otomatis. Parameter AX yang diterimanya adalah 0xE00 + c dimana c adalah karakter yang ingin dicetak. Berikut contoh kode penggunaan interrupt tersebut:

```
interrupt(0x10, 0xE00 + c, 0, 0, 0);
```

Setelah itu, implementasikan **void readString(char *string)** dimana **string** adalah sebuah string NUL-terminated hasil pembacaan input keyboard pengguna. Anda dapat menggunakan interrupt nomor 0x16 yang mengembalikan karakter yang diinput keyboard pengguna. Interrupt ini tidak menerima parameter. Berikut contoh kode penggunaan interrupt tersebut:

```
char c = interrupt(0x16, 0, 0, 0, 0);
```

Terdapat kasus khusus dimana c adalah ‘\r’ (carriage return) dan ‘\b’ (backspace) dimana ‘\r’ menAndakan akhir dari pembacaan input dan ‘\b’ menAndakan akan dilakukan backspace.

Hint:

- **interrupt(0x10, 0xE00 + '\b', 0, 0, 0)** dapat digunakan untuk memundurkan kursor satu karakter.
- **interrupt(0x10, 0xE00 + '\0', 0, 0, 0)** dapat digunakan untuk menghapus karakter pada posisi kursor.
- **interrupt(0x10, 0xE00 + '\r', 0, 0, 0)** dapat digunakan untuk memindahkan kursor ke awal baris (carriage return).
- **interrupt(0x10, 0xE00 + '\n', 0, 0, 0)** dapat digunakan untuk memindahkan kursor ke baris selanjutnya (newline).

c. Implementasi fungsi syscall readSector, writeSector

Sebelumnya, implementasikan fungsi bagi dan modulo terlebih dahulu. bcc tidak menyediakan operasi bagi dan modulo sehingga harus dibuat manual.

```
int mod(int a, int b) {
   while(a >= b) {
      a = a - b;
   }
   return a;
}

int div(int a, int b) {
   int q = 0;
   while(q*b <= a) {
      q = q+1;
   }
   return q-1;
}
```

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);
```

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

```
interrupt(0x13, 0x301, buffer, div(sector, 36) * 0x100 +
mod(sector, 18) + 1, mod(div(sector, 18), 2) * 0x100);
```

d. Implementasi fungsi syscall readFile, writeFile

Implementasikan **void readFile(char *buffer, char *filename, int *success)** dimana buffer adalah array char yang akan diisi dengan file yang dibaca, filename adalah nama file yang dibaca, dan success adalah return value TRUE jika file berhasil dibaca, FALSE jika tidak.

Pertama-tama, gunakan **readSector(dir, DIR_SECTOR)** untuk membaca sektor dir ke sebuah array char sebesar SECTOR_SIZE (512). Setelah pemanggilan fungsi ini, array dir akan berisi sektor dir. Lakukan traversal setiap entry pada dir (setiap DIR_ENTRY_LENGTH (32) byte) dan cek MAX_FILENAME (12) byte pertama pada entry tersebut apakah sama dengan filename. Jika tidak ada entri yang sama dengan filename, set success menjadi FALSE dan return. Jika sama, maka lakukan traversal setiap byte pada MAX_SECTORS (20) byte terakhir dari entry tersebut. Byte-byte tersebut merupakan nomor sektor dari file tersebut. Gunakan **readSector(buffer + i * SECTOR_SIZE, sector)** yang mana **i** adalah urutan sektor (dari 0 hingga MAX_SECTORS - 1) dan **sector** adalah nomor sektor. Traversal dilakukan hingga menemukan byte bernilai 0 atau **i** mencapai MAX_SECTORS. Set success menjadi TRUE dan return.

Sebelum implementasi writeFile, implementasikan terlebih dahulu fungsi pembantu clear yang berfungsi mengosongkan sebuah array char dengan kode berikut:

```
void clear(char *buffer, int length) {
   int i;
   for(i = 0; i < length; ++i) {
      buffer[i] = EMPTY;
   }
```

```
}
```

Implementasikan **void writeFile(char *buffer, char *filename, int *sectors)** dengan kode berikut:

```
char map[SECTOR_SIZE];
char dir[SECTOR_SIZE];
char sectorBuffer[SECTOR_SIZE];
int dirIndex;

readSector(map, MAP_SECTOR);
readSector(dir, DIR_SECTOR);

for (dirIndex = 0; dirIndex < MAX_FILES; ++dirIndex) {
   if (dir[dirIndex * DIR_ENTRY_LENGTH] == '\0') {
      break;
   }
}

if (dirIndex < MAX_FILES) {
   int i, j, sectorCount;
   for (i = 0, sectorCount = 0; i < MAX_BYTE && sectorCount < *sectors; ++i) {
      if (map[i] == EMPTY) {
         ++sectorCount;
      }
   }

   if (sectorCount < *sectors) {
      *sectors = INSUFFICIENT_SECTORS;
      return;
   } else {
      clear(dir + dirIndex * DIR_ENTRY_LENGTH, DIR_ENTRY_LENGTH);
      for (i = 0; i < MAX_FILENAME; ++i) {
         if (filename[i] != '\0') {
            dir[dirIndex * DIR_ENTRY_LENGTH + i] = filename[i];
         } else {
            break;
         }
      }
```

```
      for (i = 0, sectorCount = 0; i < MAX_BYTE && sectorCount < *sectors; ++i) {
         if (map[i] == EMPTY) {
            map[i] = USED;
            dir[dirIndex * DIR_ENTRY_LENGTH + MAX_FILENAME + sectorCount] = i;
            clear(sectorBuffer, SECTOR_SIZE);
            for (j = 0; j < SECTOR_SIZE; ++j) {
               sectorBuffer[j] = buffer[sectorCount * SECTOR_SIZE + j];
            }
            writeSector(sectorBuffer, i);
            ++sectorCount;
         }
      }
   }
} else {
   *sectors = INSUFFICIENT_DIR_ENTRIES;
   return;
}

writeSector(map, MAP_SECTOR);
writeSector(dir, DIR_SECTOR);
```

e. Implementasi fungsi syscall executeProgram

Implementasikan void **executeProgram(char *filename, int segment, int *success)** dimana filename adalah string nama file, segment adalah nomor segment memory program akan berada (0x2000), dan success adalah return value.

Pertama, alokasikan sebuah array char buffer sebesar MAX_SECTORS * SECTOR_SIZE. Kemudian, buka file dengan fungsi readFile dengan parameter yang sesuai. Set nilai success fungsi executeProgram sama dengan nilai success readFile. Jika success sama dengan TRUE, lakukan traversal setiap byte pada array buffer, untuk setiap byte gunakan **putInMemory(segment, i, buffer[i])** untuk menyalin buffer ke segment memori yang dispesifikasikan. Setelah traversal selesai, gunakan **launchProgram(segment)** untuk menjalankan program pada segment yang sudah diisi sebelumnya.

8. Mengeksekusi sebuah program

Hal terakhir yang akan kalian lakukan adalah menjalankan sebuah program yang sudah tersedia di archive zip Anda bernama keyproc. Pertama, gunakan program loadFile (kompilasi terlebih dahulu dengan **`gcc loadFile.c -o loadFile`**) untuk memasukkan file ke floppya.img.

Perintah yang digunakan adalah sebagai berikut:

```
./loadFile keyproc
```

(Anda dapat memasukkan baris ini ke file compileOS.sh Anda).

Kemudian di fungsi main kernel Anda, gunakan interrupt yang Anda sudah implementasikan sebelumnya untuk mengeksekusi program dengan filename "keyproc". Program akan meminta access code Anda dan menulis sebuah file bernama key.txt. Gunakan interrupt readFile untuk membaca isi file tersebut dan mendapatkan key. Key nanti akan disubmit saat pengumpulan tugas.

9. Bonus

Kerjakan bonus berikut ini hanya jika tugas utama sudah diselesaikan.

a. Membuat logo OS yang ditampilkan setelah booting

Gunakan fungsi putInMemory untuk membuat logo OS Anda yang ditampilkan setelah booting. Buatlah sekreatif mungkin. **`interrupt(0x16, 0, 0, 0, 0)`** dapat digunakan untuk membuat fitur seperti '*Press any key to continue*'.

b. Membuat program kalkulator sederhana

Kalkulator minimal memiliki operasi tambah, kurang, kali, bagi, dan modulo integer. Untuk melakukan kompilasi program gunakan perintah berikut:

```
as86 lib.asm -o lib_asm.o
bcc -ansi -c -o program.o program.c
ld86 -o program -d program.o lib_asm.o
./loadFile program
```

(Anda dapat memasukkan baris ini ke file compileOS.sh Anda).

lib.asm berfungsi agar program Anda dapat menggunakan fungsi **`int interrupt (int number, int AX, int BX, int CX, int DX)`**.

IV. Pengumpulan dan Deliverables

1. Buat sebuah file zip dengan nama **IF2230-TugasBesar-Milestone1-KXX-KelompokYY.zip** dengan XX adalah nomor kelas dan YY adalah nomor kelompok (2 digit). File zip ini terdiri dari 2 folder:
    a. Folder **src**, berisi file:
        i. Kernel.c
        ii. compileOS.bash (jika ada)
        iii. Source code untuk bonus jika mengerjakan
    b. Folder **doc**, berisi file laporan dengan nama file **IF2230-TugasBesar-Milestone1-KXX-KelompokYY-Laporan.pdf** berisi:
        i. Cover yang mencakup minimal NIM dan nama setiap anggota kelompok.
        ii. Jawaban dari pertanyaan berikut:
            1. [Bootloader] Apa perbedaan booting disk MBR dengan GPT? Cara boot yang mana yang digunakan dalam tugas ini?
            2. [Kernel] Apa itu Kernel Panic dan mengapa bisa terjadi?
            3. [Write to Memory] Apa yg terjadi jika code snippet ini dijalankan di kernel mode dan di usermode?

```
int main (void) {
   *((int*)0) = 0xl1fe;
}
```

            4. [Sector] Apa kelebihan dan kekurangan jika setiap sector pada suatu Hard Disk berukuran besar, misalkan 64MB?
            5. [Read/Write] Melakukan file I/O pada kernel cenderung dikatakan pelanggaran standard practice, mengapa?
        iii. *Screenshot* hasil dari langkah 6 dan 8. Jika mengerjakan bonus, tambahkan *screenshot* langkah 9.
        iv. Pembagian tugas dengan dalam bentuk tabel dengan header seperti berikut:

NIM	Nama	Bagian kerja	Persentase kontribusi

        v. Kesulitan saat mengerjakan (jika ada) dan feedback mengenai tugas ini.

2. Teknis pengumpulan akan diberitahukan sekitar 48 jam sebelum deadline pengumpulan. Deadline terdapat pada halaman cover pada tugas ini.