Spesifikasi Tugas Besar - Milestone 2
IF2230 - Sistem Operasi

“Step One to Two”

Pembuatan Sistem Operasi Sederhana

Hierarchical File System, Shell, Utility Programs

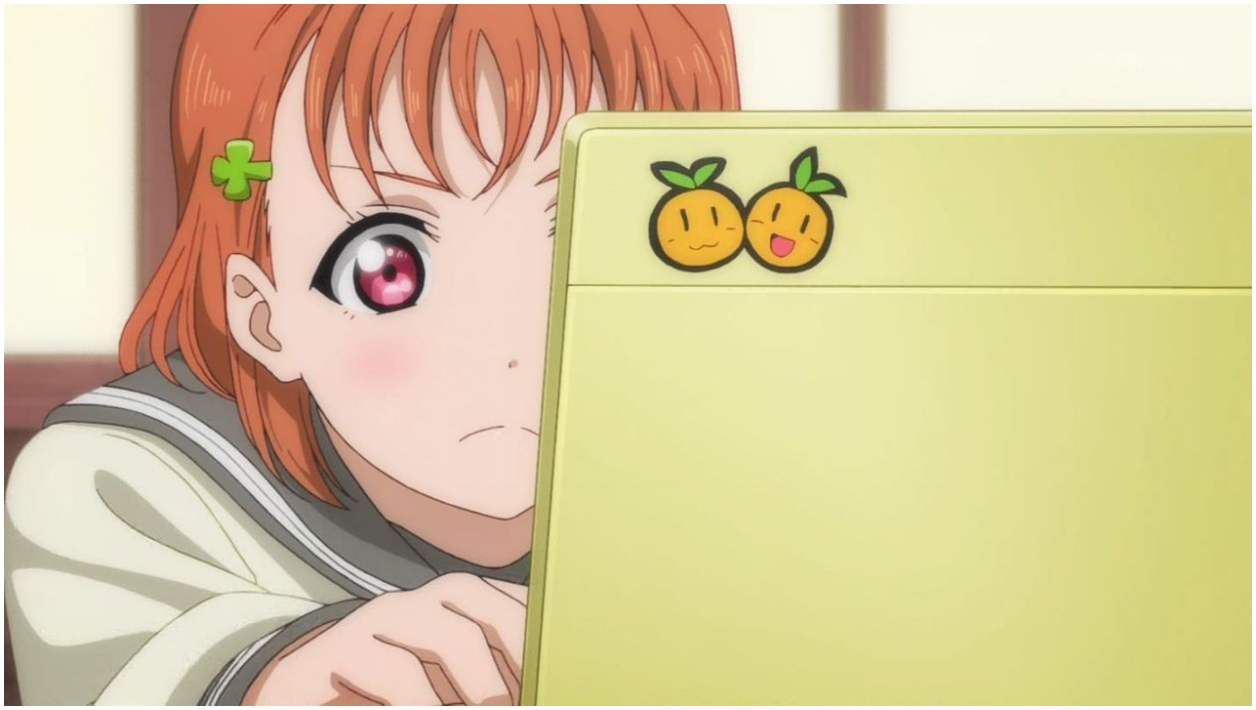
Dipersiapkan oleh:
Asisten Lab Sistem Terdistribusi

Didukung oleh:

Waktu Mulai:
Kamis, 29 Maret 2018 20.08.00 WIB

Waktu Akhir:
Rabu, 11 Maret 2018 20.07.59 WIB

I. Latar Belakang



Tahun 2228.

Telah lewat 10 tahun sejak peristiwa rusaknya mesin-mesin produksi rations yang merupakan satu-satunya sumber makanan manusia yang tersisa di bumi. Akan tetapi, berkat keahlian dari para arkeolog komputer saat itu, mesin-mesin tersebut berhasil dijalankan kembali menggunakan kunci akivasi yang ditemukan oleh mereka.

Ketika seorang researcher sedang menganalisa data kejadian tersebut, ia menemukan sebuah file janggal pada floppy disk komputer-komputer yang digunakan. Akan tetapi sistem operasi yang dikembangkan saat kejadian itu tidak dapat mengaksesnya karena file tersebut tersembunyi dalam berlapis-lapis direktori. Sebuah upgrade untuk sistem operasinya, yang mengubah file systemnya menjadi file system hirarkis, dibutuhkan untuk memecahkan misteri ini.

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, Program Execution
2. **Milestone 2: Hierarchical File System, Shell, Utility Programs**
    1. Mengubah kernel sistem operasi
        a. Mengubah struktur file system menjadi file system hirarkis
        b. Mengubah fungsi handleInterrupt21
        c. Mengubah implementasi syscall readFile
        d. Mengubah implementasi syscall writeFile
        e. Mengubah implementasi syscall executeProgram
        f. Implementasi syscall terminateProgram
        g. Implementasi syscall makeDirectory
        h. Implementasi syscall deleteFile
        i. Implementasi syscall deleteDirectory
        j. Implementasi syscall putArgs, getArgc, getArgv
    2. Membuat shell sistem operasi
        a. Implementasi perintah cd
        b. Implementasi perintah menjalankan program
    3. Membuat program-program utilitas
        a. Membuat program utilitas echo
        b. Membuat program utilitas mkdir
        c. Membuat program utilitas ls
        d. Membuat program utilitas rm
        e. Membuat program utilitas cat
    4. Menjalankan program test
    5. **Bonus**:
        a. Membuat program utilitas mv
        b. Membuat program utilitas cp
        c. Lain-lain
3. Milestone 3: Process, Multiprogramming

III. Langkah Pengerjaan

1. Mengubah kernel sistem operasi

a. Mengubah struktur file system menjadi file system hirarkis

Berbeda dengan seperti struktur file system anda sebelumnya dimana setiap file berada pada satu direktori root saja, struktur file system yang akan anda implementasikan pada milestone ini bersifat hirarkis. Untuk file system yang baru ini, direktori root selain berisi file-file berisi direktori lain juga, dimana direktori lain tersebut juga dapat berisi file-file dan direktori lain.

Sebelumnya, struktur dari file system anda adalah seperti berikut:

- **map** (di sektor 0x1)
- **dir** (di sektor 0x2)
- **kernel** (di sektor 0x3-0xC)

Struktur dari sektor **dir**:

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Nama file 0 (max 12 karakter)												Sektor file 0			
01	Sektor file 0 (20 sektor = max 10.24 kB per file)															
02	Nama file 1 (max 12 karakter)												Sektor file 1			
03	Sektor file 1 (20 sektor = max 10.24 kB per file)															
...																
1E	Nama file 15 (max 12 karakter)												Sektor file 15			
1F	Sektor file 15 (20 sektor = max 10.24 kB per file)															

Setelah diubah, struktur dari file system anda akan seperti berikut:

- **map** dipindahkan ke sektor 0x100. Hal ini dilakukan supaya sektor-sektor utilitas tidak memakan tempat sektor-sektor yang dapat digunakan oleh file system (0x00-0xFF).
- **dir** dihilangkan.
- Sektor baru **dirs** dibuat di sektor 0x101. Sektor ini berfungsi untuk menyatakan sebuah direktori pada file system. Setiap entry memiliki indeks direktori parentnya (0x00-0x1F, 0xFF jika root) dan nama direktorinya sendiri maksimum sepanjang 15 karakter.

- Sektor baru **files** dibuat di sektor 0x102. Sektor ini berfungsi untuk menyatakan sebuah file pada file system. Setiap entry memiliki indeks direktori parentnya (0x00-0x1F, 0xFF jika root) dan nama filenya sendiri maksimum sepanjang 15 karakter.
- Sektor baru **sectors** dibuat di sektor 0x103. Sektor ini berfungsi untuk menyatakan sector-sector setiap file.
- **kernel** (di sektor 0x1-0xA)

Struktur dari sektor **dirs**:

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	p	Nama direktori 0 (max 15 karakter)														
01	p	Nama direktori 1 (max 15 karakter)														
...																
1F	p	Nama direktori 31 (max 15 karakter)														

p : index direktori parent (0x00-0x1F) dari file tersebut, 0xFF jika parent adalah root

Struktur dari sektor **files**:

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	p	Nama file 0 (max 15 karakter)														
01	p	Nama file 1 (max 15 karakter)														
...																
1F	p	Nama file 31 (max 15 karakter)														

p : index direktori parent (0x00-0x1F) dari direktori tersebut, 0xFF jika parent adalah root

Struktur dari sektor **sectors**:

0x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	Sector file 0 (16 sektor = max 8.192 kB per file)															
01	Sector file 1 (16 sektor = max 8.192 kB per file)															
...																
1F	Sector file 31 (16 sektor = max 8.192 kB per file)															

Pada kit milestone 2 sudah tersedia **bootload.asm**, **loadFile.c**, **map.img**, **files.img**, dan **sectors.img** yang telah disesuaikan untuk struktur file system yang baru. Salinlah file-file tersebut ke direktori sistem operasi kalian.

Selain itu, ubahlah bagian penulisan sektor awal pada compileOS.sh kalian sehingga menjadi seperti berikut:

```
dd if=/dev/zero of=floppya.img bs=512 count=2880
dd if=bootload of=floppya.img bs=512 count=1 conv=notrunc
dd if=map.img of=floppya.img bs=512 count=1 seek=256 conv=notrunc
dd if=files.img of=floppya.img bs=512 count=1 seek=258 conv=notrunc
dd if=sectors.img of=floppya.img bs=512 count=1 seek=259
conv=notrunc
```

b. Mengubah fungsi handleInterrupt21

Karena keperluan beberapa syscall setelah ini, parameter AX dari fungsi handleInterrupt21 akan dipisah menjadi AL dan AH.

```
void handleInterrupt21 (int AX, int BX, int CX, int DX) {
  char AL, AH;
  AL = (char) (AX);
  AH = (char) (AX >> 8);

  switch (AL) {
    case 0x00:
      printString(BX);
      break;
    case 0x01:
      readString(BX);
      break;
    case 0x02:
      readSector(BX, CX);
      break;
    case 0x03:
      writeSector(BX, CX);
      break;
    case 0x04:
      readFile(BX, CX, DX, AH);
      break;
    case 0x05:
      writeFile(BX, CX, DX, AH);
      break;
    case 0x06:
```

```
      executeProgram(BX, CX, DX, AH);
      break;
    case 0x07:
      terminateProgram(BX);
      break;
    case 0x08:
      makeDirectory(BX, CX, AH);
      break;
    case 0x09:
      deleteFile(BX, CX, AH);
      break;
    case 0x0A:
      deleteDirectory(BX, CX, AH);
      break;
    case 0x20:
      putArgs(BX, CX);
      break;
    case 0x21:
      getCurdir(BX);
      break;
    case 0x22:
      getArgc(BX);
      break;
    case 0X23:
      getArgv(BX, CX);
      break;
    default:
      printString("Invalid interrupt");
  }
}
```

Untuk pemanggilan interrupt sekarang menggunakan kode seperti berikut:

```
interrupt(0x21, (AH << 8) | AL, BX, CX, DX);
```

c. Mengubah implementasi syscall readFile

Ubah syscall readFile sehingga menerima path relatif dari file yang akan dibaca seperti “abc/def/g”, bukan nama filenya saja.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x04 dan memiliki function signature sebagai berikut:

```
void readFile(char *buffer, char *path, int *result, char
parentIndex)
```

Untuk membaca file “g” dengan path relatif “abc/def/g” dapat dilakukan langkah-langkah berikut:

1. Cari indeks direktori pada sektor **dirs** yang bernama “abc” dan indeks parentnya adalah adalah parentIndex (0xFF jika root). Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
2. Cari indeks direktori pada sektor **dirs** yang bernama “def” dan indeks parentnya adalah indeks direktori “abc” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
3. Cari indeks file pada sektor **files** yang bernama “g” dan indeks parentnya adalah indeks direktori “def” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
4. Baca daftar sektor file pada entry file dengan indeks tersebut pada sektor **sectors**.
5. Masukkan sektor-sektor tersebut ke buffer.
6. Kembalikan success (0) pada parameter result.

d. Mengubah implementasi syscall writeFile

Ubah syscall writeFile sehingga menerima path relatif dari file yang akan ditulis seperti “abc/def/g”, bukan nama filenya saja.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x05 dan memiliki function signature sebagai berikut:

```
void writeFile(char *buffer, char *path, int *sectors, char
parentIndex)
```

Untuk menulis file “g” dengan path relatif “abc/def/g” dapat dilakukan langkah-langkah berikut:

1. Cek apakah jumlah sektor kosong cukup pada sektor **map**. Jika tidak ada maka syscall akan gagal dan mengembalikan error INSUFFICIENT_SECTORS (0) pada parameter sectors.
2. Cek apakah masih tersisa entry kosong pada sektor **files** (ciri-ciri entry kosong adalah byte pertama dari nama filenya adalah karakter NUL ‘\0’). Jika tidak ada maka syscall akan gagal dan mengembalikan error INSUFFICIENT_ENTRIES (-3) pada parameter sectors.
3. Cari indeks direktori pada sektor **dirs** yang bernama “abc” dan indeks parentnya adalah adalah parentIndex (0xFF jika root). Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter sectors.
4. Cari indeks direktori pada sektor **dirs** yang bernama “def” dan indeks parentnya adalah indeks direktori “abc” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter sectors.

5. Cari indeks file pada sektor **files** yang bernama “g” dan indeks parentnya adalah indeks direktori “def” yang didapatkan sebelumnya. Jika ada maka syscall akan gagal dan mengembalikan error ALREADY_EXISTS (-2) pada parameter sectors.
6. Pada entry kosong pertama pada sektor **files** tulis indeks dari direktori “def” yang didapatkan sebelumnya pada byte indeks parent dan “g” pada byte-byte nama filenya.
7. Cari sektor kosong pertama pada sektor **map** dan mark byte sektor tersebut menjadi terisi (0xFF).
8. Tulis data dari buffer ke sektor dengan nomor tersebut.
9. Lakukan 6 dan 7 untuk setiap sektor dari file.

e. Mengubah implementasi syscall executeProgram

Ubah syscall executeProgram sehingga menerima path relatif dari program yang akan diexecute. Anda tidak perlu mengubah banyak karena perubahan utama sudah ditangani oleh readFile.

Syscall ini dipanggil melalui interrupt 0x21 AX=0x06 dan memiliki function signature sebagai berikut:

```
void executeProgram(char *path, int segment, int *result, char
parentIndex)
```

f. Implementasi syscall terminateProgram

[Update 2018-04-07]
Buat syscall executeProgram yang mengembalikan eksekusi program ke shell sehingga seakan-akan mengakhiri eksekusi program sebelumnya. Syscall ini dipanggil setiap akhir program. Berikut kode implementasinya:

```
void terminateProgram (int *result) {
  char shell[6];
  shell[0] = 's';
  shell[1] = 'h';
  shell[2] = 'e';
  shell[3] = 'l';
  shell[4] = 'l';
  shell[5] = '\0';
  executeProgram(shell, 0x2000, result, 0xFF);
}
```

Syscall ini dipanggil melalui interrupt 0x21 AL=0x07.

g. Implementasi syscall makeDirectory

Buatlah syscall makeDirectory yang menerima path relatif dari direktori yang akan dibuat seperti “abc/def/ghi”.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x08 dan memiliki function signature sebagai berikut:

```
void makeDirectory(char *path, int *result, char parentIndex)
```

Untuk membuat direktori “ghi” dengan path relatif “abc/def/ghi” dapat dilakukan langkah-langkah berikut:

1. Cek apakah masih tersisa entry kosong pada sektor **dirs** (ciri-ciri entry kosong adalah byte pertama dari nama direktorinya adalah karakter NUL ‘\0’). Jika tidak ada maka syscall akan gagal dan mengembalikan error INSUFFICIENT_ENTRIES (-3) pada parameter result.
2. Cari indeks direktori pada sektor **dirs** yang bernama “abc” dan indeks parentnya adalah adalah parentIndex (0xFF jika root). Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
3. Cari indeks direktori pada sektor **dirs** yang bernama “def” dan indeks parentnya adalah indeks direktori “abc” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
4. Cari indeks file pada sektor **files** yang bernama “ghi” dan indeks parentnya adalah indeks direktori “def” yang didapatkan sebelumnya. Jika ada maka syscall akan gagal dan mengembalikan error ALREADY_EXISTS (-2) pada parameter result.
5. Pada entry kosong pertama pada sektor **dirs,** tulis indeks dari direktori “def” yang didapatkan sebelumnya pada byte indeks parent dan “ghi” pada byte-byte nama direktorinya.
6. Kembalikan success (0) pada parameter result.

h. Implementasi syscall deleteFile

Buat syscall deleteFile yang menerima path relatif dari file yang akan dihapus seperti “/abc/def/g”.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x09 dan memiliki function signature sebagai berikut:

```
void deleteFile(char *path, int *result, char parentIndex)
```

Untuk menghapus file “g” dengan path relatif “abc/def/g” dapat dilakukan langkah-langkah berikut:

1. Cari indeks direktori pada sektor **dirs** yang bernama “abc” dan indeks parentnya adalah parentIndex (0xFF jika root). Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
2. Cari indeks direktori pada sektor **dirs** yang bernama “def” dan indeks parentnya adalah indeks direktori “abc” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
3. Cari indeks file pada sektor **files** yang bernama “g” dan indeks parentnya adalah indeks direktori “def” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
4. Pada entry file dengan indeks tersebut pada sektor **files**, ubah byte pertama nama filenya menjadi karakter NUL ‘\0’.
5. Baca daftar sektor file pada entry file dengan indeks tersebut pada sektor **sectors**.
6. Mark byte sektor-sektor tersebut menjadi kosong (0x00) pada sektor **map**.
7. Kembalikan success (0) pada parameter result.

i. Implementasi syscall deleteDirectory

Buat syscall deleteDirectory yang menerima path relatif dari direktori yang akan dihapus seperti “abc/def/ghi”. Syscall ini juga akan menghapus secara rekursif isi dari direktori tersebut, termasuk isi dari direktori lain di dalam direktori tersebut.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x0A dan memiliki function signature sebagai berikut:

```
void deleteDirectory(char *path, int *success, char parentIndex)
```

Untuk menghapus direktori “ghi” dengan path relatif “abc/def/ghi” dapat dilakukan langkah-langkah berikut:

1. Cari indeks direktori pada sektor **dirs** yang bernama “abc” dan indeks parentnya adalah adalah parentIndex (0xFF jika root). Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
2. Cari indeks direktori pada sektor **dirs** yang bernama “def” dan indeks parentnya adalah indeks direktori “abc” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
3. Cari indeks direktori pada sektor **dirs** yang bernama “ghi” dan indeks parentnya adalah indeks direktori “def” yang didapatkan sebelumnya. Jika tidak ada maka syscall akan gagal dan mengembalikan error NOT_FOUND (-1) pada parameter result.
4. Pada entry direktori dengan indeks tersebut pada sektor **dirs**, ubah byte pertama nama direktorinya menjadi karakter NUL ‘\0’.
5. Cari indeks dari semua file yang indeks parentnya adalah indeks direktori “def” pada sektor **files.** Hapus file-file tersebut menggunakan langkah 4-6 syscall deleteFile. Anda dapat

memisahkan langkah-langkah tersebut menjadi fungsi deleteFile terpisah yang menerima indeks file langsung.

6. Cari indeks dari semua direktori yang indeks parentnya adalah indeks direktori “def” pada sektor **dirs**. Hapus direktori-direktori tersebut menggunakan langkah 4-6 syscall deleteDirectory (termasuk langkah ini). Anda dapat memisahkan langkah-langkah tersebut menjadi fungsi deleteDirectory terpisah yang menerima indeks direktori langsung.
7. Kembalikan success (0) pada parameter result.

j. Implementasi syscall putArgs, getArgc, getArgv

Syscall putArgs (interrupt 0x21 AL=0x20) digunakan untuk menyimpan current directory, jumlah parameter program, dan isi parameter program sebelum program dieksekusi.
Syscall getCurdir (interrupt 0x21 AL=0x21) digunakan untuk mendapatkan indeks current directory.
Syscall getArgc (interrupt 0x21 AL=0x22) digunakan untuk membaca jumlah parameter program.
Syscall getArgc (interrupt 0x22 AL=0x23) digunakan untuk membaca isi parameter ke-n program.
Gunakan kode di bawah untuk implementasinya:

```
#define ARGS_SECTOR 512

void putArgs (char curdir, char argc, char **argv) {
  char args[SECTOR_SIZE];
  int i, j, p;
  clear(args, SECTOR_SIZE);

  args[0] = curdir;
  args[1] = argc;
  i = 0;
  j = 0;
  for (p = 1; p < ARGS_SECTOR && i < argc; ++p) {
    args[p] = argv[i][j];
    if (argv[i][j] == '\0') {
      ++i;
      j = 0;
    }
    else {
     ++j;
    }
  }

  writeSector(args, ARGS_SECTOR);
```

```
}

void getCurdir (char *curdir) {
  char args[SECTOR_SIZE];
  readSector(args, ARGS_SECTOR);
  *curdir = args[0];
}


void getArgc (char *argc) {
  char args[SECTOR_SIZE];
  readSector(args, ARGS_SECTOR);
  *argc = args[1];
}

void getArgv (char index, char *argv) {
  char args[SECTOR_SIZE];
  int i, j, p;
  readSector(args, ARGS_SECTOR);

  i = 0;
  j = 0;
  for (p = 1; p < ARGS_SECTOR; ++p) {
    if (i == index) {
      argv[j] = args[p];
      ++j;
    }
    if (args[p] == '\0') {
      if (i == index) {
        break;
      }
      else {
        ++i;
      }
    }
  }
}
```

2. Membuat shell sistem operasi

a. Persiapan program shell

Anda sekarang diminta untuk membuat program shell untuk sistem operasi anda. Buatlah sebuah file source code C bernama **shell.c**. Pada fungsi main kernel.c, panggil

```
interrupt(0x21, 0xFF << 8 | 0x6, "shell", 0x2000, &success);
```

untuk menjalankan shell setiap kali kernel dijalankan.

Program shell akan menampilkan $ pada layar dan menunggu input dari penguna seperti di bawah:

```
$
```

b. Implementasi perintah cd

Program shell juga menyimpan nilai indeks dari current directory sebagai sebuah variable berukuran byte. Current directory dapat dibuah menggunakan command cd, dengan parameter pertamanya adalah path relatif ke direktori baru yang diinginkan pengguna.

c. Implementasi perintah menjalankan program

Hal yang harus ditangani oleh shell adalah perintah dari user untuk menjalankan program. Misal, jika pengguna memasukkan hal seperti di bawah pada shell:

```
$ myprog
```

Gunakan syscall executeProgram untuk menjalankan program dengan nama “myprog” pada root directory.

Untuk menjalankan program pada current directory, pengguna menggunakan perintah seperti di bawah:

```
$ ./myprog
```

Syntax ‘./’ menandakan shell untuk mencari program pada current directory.

Pengguna juga dapat memasukkan argumen-argumen (dipisah dengan spasi) untuk program, misal:

```
$ ./myprog abc 123
```

Sebelum executeProgram dijalankan, gunakan terlebih dahulu setArgs untuk menyimpan direktori sekarang (curdir), jumlah argumen (argc), dan nilai-nilai argumen (argv). Contoh penggunaan dapat dilihat di bawah:

```
char curdir;
char argc;
char *argv[2];

curdir = 0xFF; //root
argc = 2;
argv[0] = "abc";
argv[1] = "123";
interrupt(0x21, 0x20, curdir, argc, argv);
```

Untuk melakukan get argumen program, gunakan syscall getCurdir, getArgc dan getArgv. Contoh penggunaannya dapat dilihat sebagai berikut:

```
int main() {
  int i;
  char curdir;
  char argc;
  char argv[4][16];

  interrupt(0x21, 0x21, &curdir, 0, 0);
  interrupt(0x21, 0x22, &argc, 0, 0);
  for (i = 0; i < argc; ++i) {
    interrupt(0x21, 0x23, i, argv[i], 0);
  }
}
```

3. Membuat program utilitas

Perhatian: setiap program utilitas harus berada di root directory supaya dapat diakses pengguna shell di direktori manapun.

a. Membuat program echo

Program echo mencetak parameter pertama yang diberikan user ke layar.

b. Membuat program mkdir

Program mkdir membuat sebuah direktori baru pada suatu direktori. Pengguna memberikan satu argumen yaitu nama direktori yang ingin dibuat.

c. Membuat program ls

Program ls mendaftarkan nama-nama semua file dan direktori yang berada pada current directory.

d. Membuat program rm

Program rm menghapus suatu file atau direktori. Pengguna memberikan satu argumen yaitu path relatif file atau direktori yang ingin dihapus.

e. Membuat program cat

Program cat mencetak isi suatu file ke layar. Pengguna memberikan satu argumen yaitu nama file yang ingin dicetak ke layar. Jika pengguna memberikan argumen kedua '-w' maka program cat akan berubah ke write mode, dimana akan dibuat file baru dengan isi input user. Detail implementasi dibebaskan.

4. Menjalankan program pengujian

[Update 2018-04-07]
Untuk menjalankan program pengujian, diperlukan aspek-aspek berikut dari sistem operasi anda sudah berfungsi dengan benar sesuai spesifikasi tugas:

- readFile (interrupt 0x21 AL=0x04)
- writeFile (interrupt 0x21 AL=0x05)
- executeProgram (interrupt 0x21 AL=0x06)
- terminateProgram (interrupt 0x21 AL=0x07)
- makeDirectory (interrupt 0x21 AL=0x08)
- Shell sistem operasi
- Program utilitas ls
- Program utilitas cat

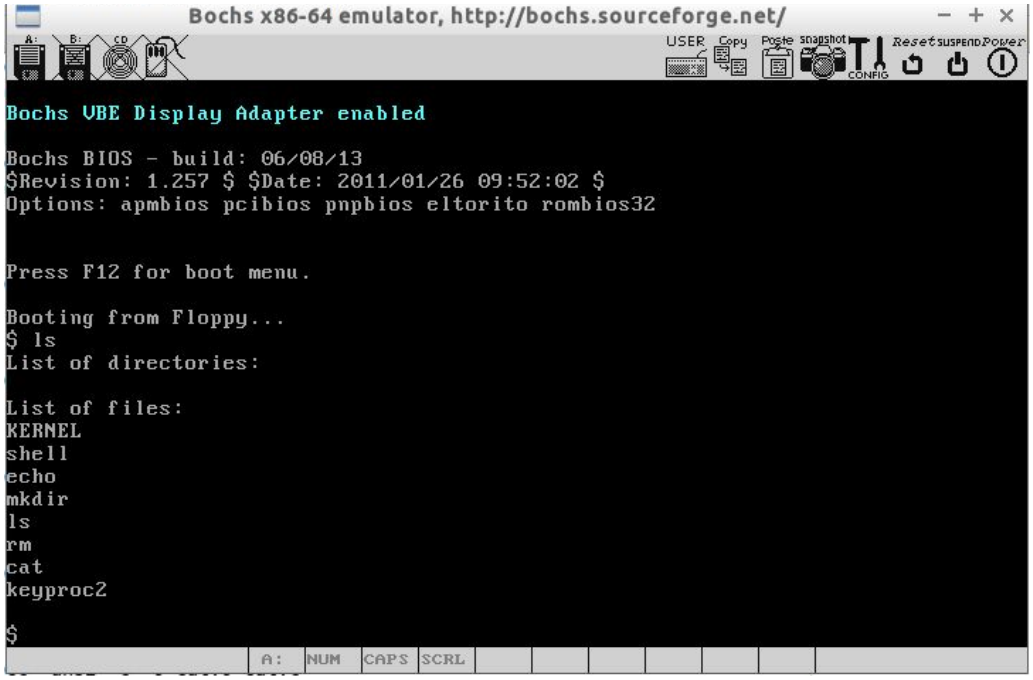
Langkah-langkah untuk menjalankan program pengujian adalah sebagai berikut:

1. Download archive zip yang berisi program pengujian yang telah disebar di milis.

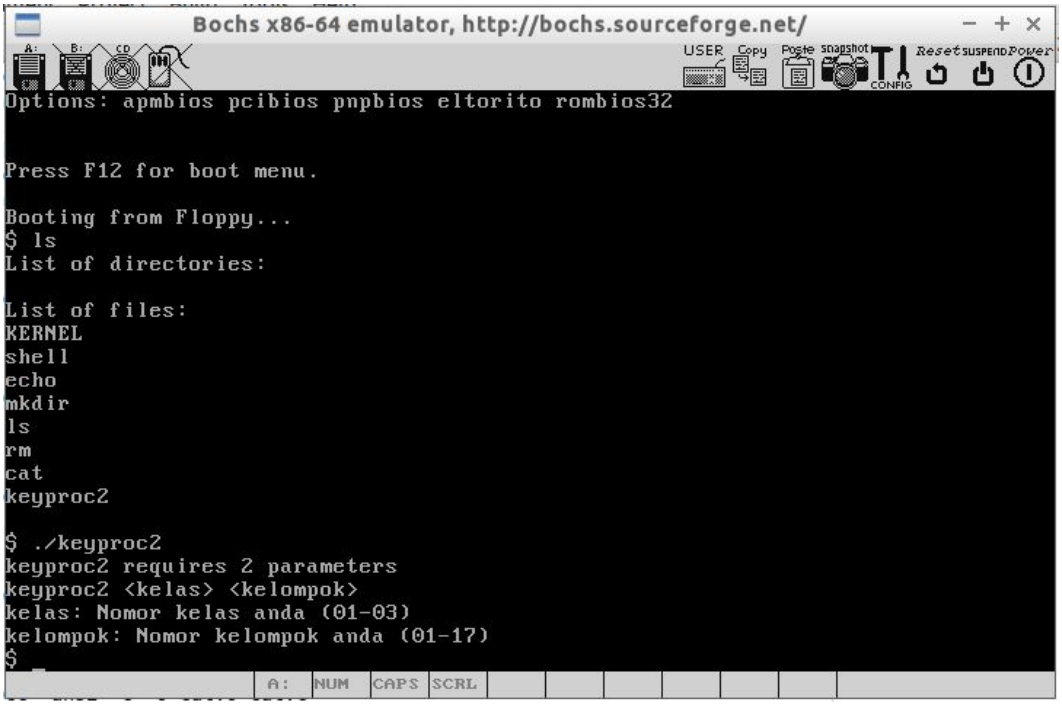
2. Masukkan file program "keyproc2" yang di dalam archive tersebut ke root directory sistem operasi anda dengan perintah. (Lebih baik perintah ini juga dimasukkan pada compileOS.sh anda).

```
./loadFile keyproc2
```

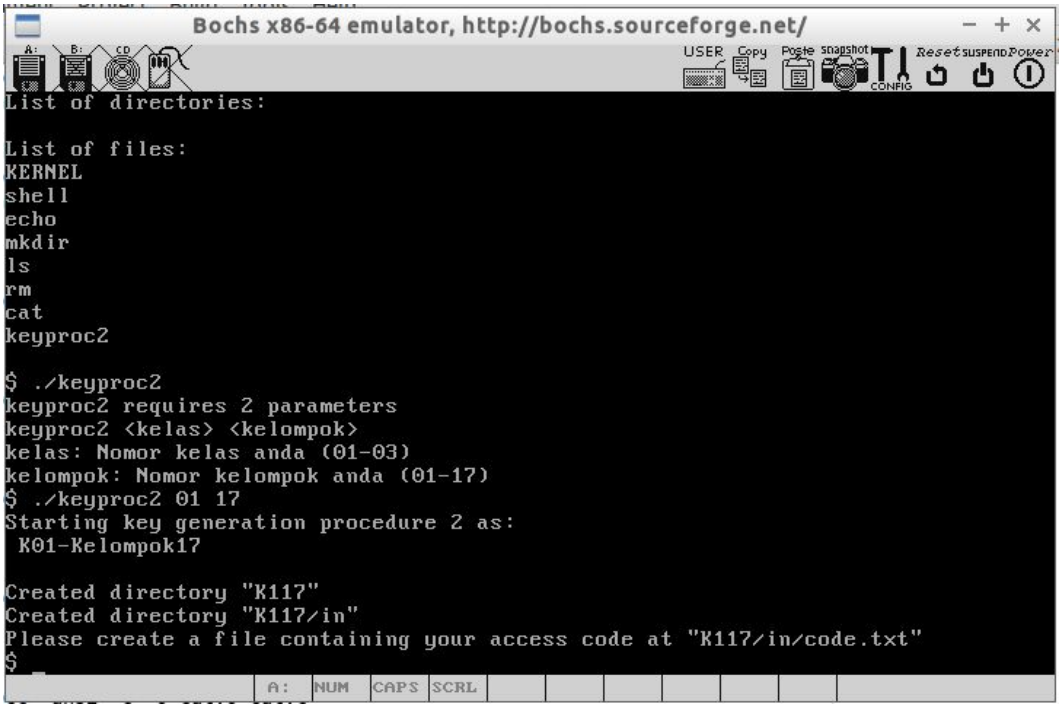
3. Pastikan bahwa program "keyproc2" sudah ada di root directory sistem operasi anda dengan mengeksekusikan program "ls" yang anda sudah buat pada shell anda.



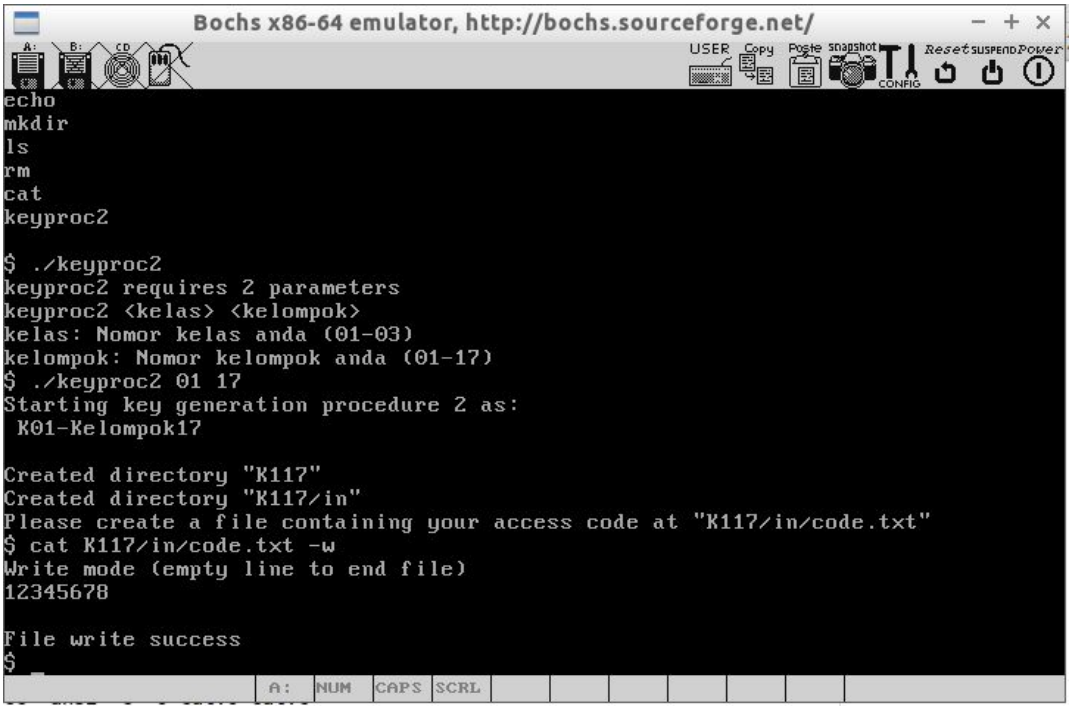
4. Jalankan program “keyproc2”.



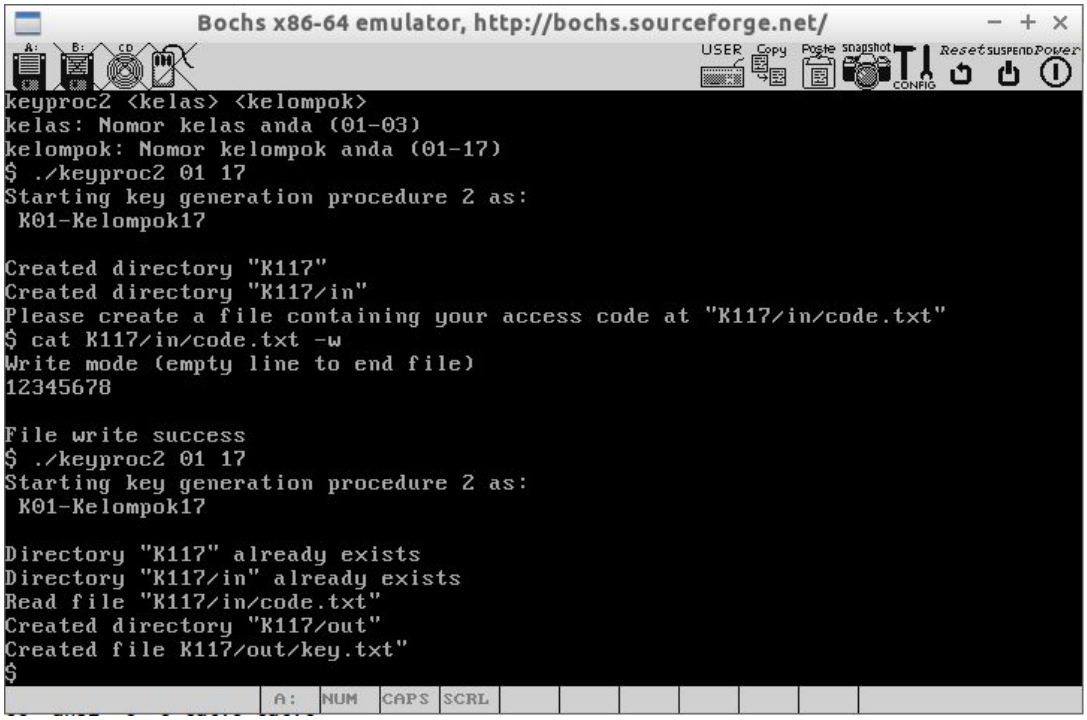
Program “keyproc2” membutuhkan 2 parameter, yaitu kelas dan kelompok anda. Jalankan kembali program “keyproc2” dengan parameter-parameter tersebut.



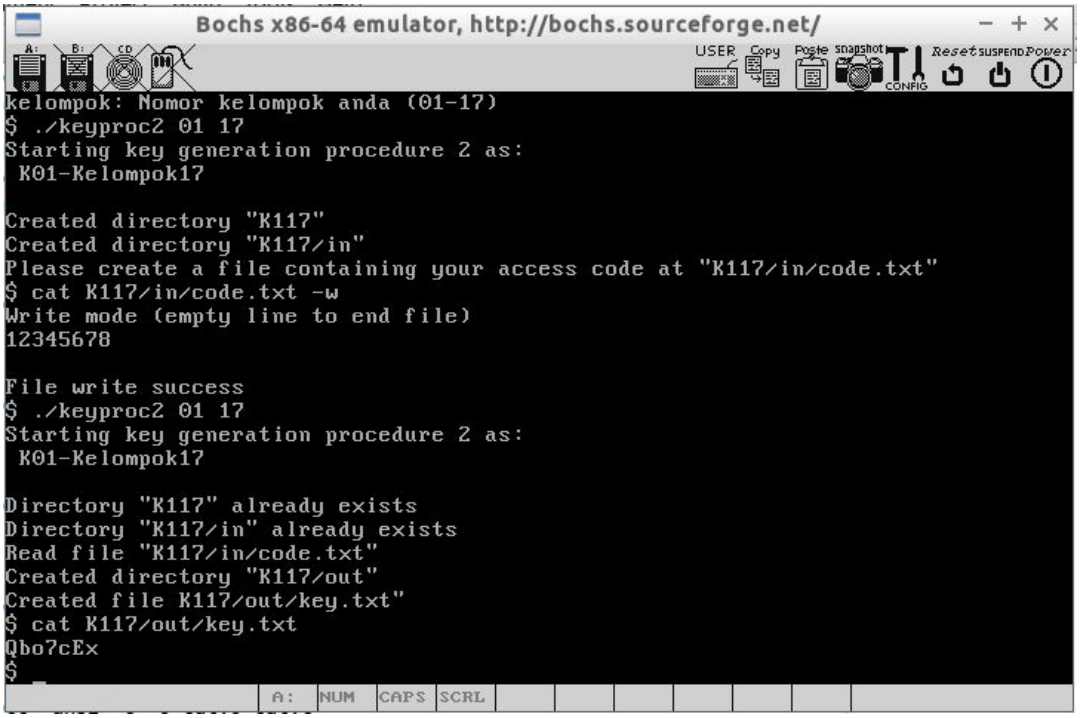
5. Lakukan apa yang ditampilkan pada layar oleh program dengan menggunakan program “cat” yang anda sudah buat dengan parameter pertama path yang dispesifikasikan dan parameter kedua flag “-w” untuk masuk ke write mode. Isi file tersebut dengan access code kelompok anda.



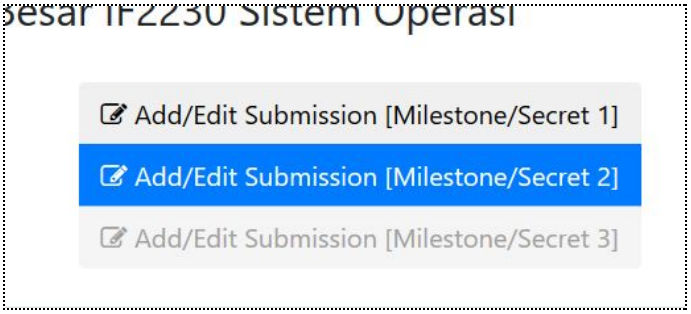
6. Jalankan kembali program “keyproc” dengan parameter sebelumnya.

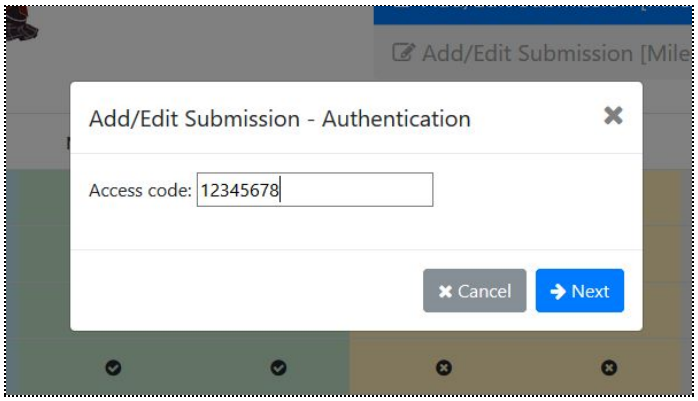


7. Gunakan program “cat” kembali untuk membaca file hasil pada path yang dispesifikasikan.



8. Masukkan secret key yang anda temukan pada
   http://liplop.herokuapp.com/if2230/pengumpulan/





5. Bonus

a. Membuat program mv

Program mv memindahkan file atau direktori argumen pertama menjadi file atau direktori argumen kedua. Fungsionalitas seperti mv pada Linux.

b. Membuat program cp

Program mv menyalin file atau direktori argumen pertama ke file atau direktori argumen kedua. Fungsionalitas seperti cp pada Linux.

c. Lain-lain

Anda diperbolehkan untuk mengimplementasikan fitur-fitur lain pada sistem operasi kalian. Kreativitas anda akan dihargai dengan nilai bonus.

IV. Pengumpulan dan Deliverables

1. Buat sebuah file **zip** dengan nama **IF2230-TugasBesar-Milestone2-KXX-KelompokYY.zip** dengan XX adalah nomor kelas dan YY adalah nomor kelompok. File zip ini terdiri dari 2 folder:
    a. Folder **src**, berisi file:
        i. kernel.c
        ii. shell.c
        iii. compileOS.sh
        iv. Source code program-program utilitas
    b. Folder **doc**, berisi file laporan dengan nama file **IF2230-TugasBesar-Milestone2-KXX-KelompokYY-Laporan.pdf** berisi:
        i. Cover yang mencakup minimal NIM dan nama setiap anggota kelompok.
        ii. Langkah-langkah pengerjaan dan screenshot seperlunya.
        iii. Pembagian tugas dengan dalam bentuk tabel dengan header seperti berikut:

NIM	Nama	Apa yang dikerjakan	Persentase kontribusi

        iv. 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.