Spesifikasi Tugas Besar - Milestone 3
IF2230 - Sistem Operasi

“Finale”

Pembuatan Sistem Operasi Sederhana

Process, Multiprogramming

Dipersiapkan oleh:
Asisten Lab Sistem Terdistribusi

Didukung oleh:

Waktu Mulai:
Kamis, 12 April 2018 14.00.00 WIB

Waktu Akhir:
Rabu, 25 April 2018 12.59.59 WIB

I. Latar Belakang

Tahun 2238

Sudah lewat lagi 10 tahun sejak ditemukan file-file misterius dalam komputer-komputer tersebut. Para researcher komputer berhasil mengupgrade sistem operasi yang mereka gunakan sehingga dapat membaca file dalam direktori hirarkis di dalamnya. File-file tersebut ternyata mengandung kode khusus yang meruju ke alamat dua program lain yang tersembunyi di dalam file systemnya. Akan tetapi entah mengapa kedua program tersebut tidak dapat dijalankan dengan benar.

Beberapa hari yang lalu, terdapat seorang researcher menemukan breakthrough yaitu kedua program tersebut harus dijalankan secara **multiprogramming**. Selain itu, juga ditemukan bahwa hasil dari kedua program tersebut akan menghasilkan kode yang dapat digunakan untuk mengaktifkan sejenis mesin yang mampu mengembalikan bumi ke kondisi sebelum terjadinya perang.

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
3. **Milestone 3: Process, Multiprogramming**
    1. Mengubah kernel sistem operasi
        a. Memperbesar ukuran maksimum kernel
        b. Implementasi Process Control Block (PCB)
        c. Implementasi handleTimerInterrupt
        d. Implementasi syscall yieldControl
        e. Implementasi syscall sleep
        f. Implementasi syscall resumeProcess
        g. Implementasi syscall killProcess
        h. Implementasi syscall pauseProcess
        i. Mengubah implementasi syscall readFile
        j. Mengubah implementasi syscall executeProgram
        k. Mengubah implementasi syscall terminateProgram
        l. Mengubah implementasi syscall readString
    2. Mengubah shell sistem operasi
        a. Mengubah implementasi perintah menjalankan program
        b. Implementasi perintah menjalankan program secara paralel
        c. Implementasi perintah pause, resume, dan kill
    3. Membuat program-program utilitas
        a. Melakukan pemanggilan fungsi enableInterrupts untuk semua user program
        b. Membuat program ps
    4. Menjalankan program test
    5. **Bonus**:
        a. Implementasi syscall timedSleep
        b. Membuat program timer
        c. Lain-lain

III. Langkah Pengerjaan

1. Mengubah kernel sistem operasi

a. Memperbesar ukuran maksimum kernel

Untuk milestone ini, mungkin ukuran maksimum kernel anda sekarang (10 sektor) mungkin tidak cukup. Oleh karena itu, pada kit milestone 3 sudah tersedia **bootload.asm**, **map.img**, **files.img**, dan **sectors.img** yang telah disesuaikan untuk ukuran maksimum kernel yang baru (16 sektor). Salinlah file-file tersebut ke direktori sistem operasi kalian.

b. Implementasi Process Control Block (PCB)

Untuk menambahkan process dan multiprogramming pada sistem operasi anda, dibutuhkan struktur data yaitu Process Control Block (PCB). Sebuah PCB mengandung informasi penting mengenai suatu process seperti segmennya, alamat stack pointernya, dan scheduling statenya. Informasi-informasi tersebut digunakan oleh sistem operasi untuk melakukan scheduling dan context switching antar process.

Pada kit milestone 3 sudah tersedia **proc.c** dan **proc.h** yang mengimplementasikan PCB. Salinlah file-file tersebut ke direktori sistem operasi kalian.

proc.h mendefinisikan sebuah tipe struct C bernama PCB yang memiliki atribut sebagai berikut:

- **index**: indeks file program pada sector files
- **state**: scheduling state dari process yang dapat berupa:
    - DEFUNCT (0): process tidak berlaku lagi
    - RUNNING (1): process sedang berjalan
    - STARTING (2): process akan pertama kali berjalan
    - READY (3): process sedang dalam queue akan berjalan
    - PAUSED (4): process diberhentikan sementara
- **segment**: segmen memori dimana process berada (0x2000, 0x3000, 0x4000, hingga 0x9000)
- **stackPointer**: alamat stack pointer process (0xFF00 saat awal eksekusi program)
- **parentSegment**: segmen memori dari process parent (yang mengeksekusikan processnya)
- **next**: pointer ke PCB selanjutnya dalam queue ready yang berupa doubly-linked list
- **prev**: pointer ke PCB sebelumnya dalam queue ready yang berupa doubly-linked list

proc.h juga mendefinisikan beberapa fungsi yang melakukan beberapa operasi pada struktur data dan queue PCB:

- **initializeProcStructures**: menginisiasikan variabel-variabel global PCB
- **getFreeMemorySegment**: mendapatkan dan menandakan segmen memori yang tidak digunakan
- **releaseMemorySegment**: melepaskan segmen memori yang sebelumnya digunakan
- **getFreePCB**: mendapatkan dan menandakan PCB dari PCB pool yang tidak digunakan
- **releasePCB**: melepaskan PCB yang sebelumnya digunakan
- **addToReady**: memasukkan PCB ke akhir ready queue
- **removeFromReady**: mengambil PCB dari awal ready queue
- **getPCBOfSegment**: mendapatkan PCB dengan nilai segmen tertentu

Untuk memasukkan PCB ke dalam kernel anda, tambahkan kode berikut pada baris teratas kernel.c anda:

```
#define MAIN
#include "proc.h"
```

Hal di atas akan melakukan definisi tipe struct PCB dan beberapa variabel global penting seperti:

- **running**: pointer ke PCB process yang sedang berjalan
- **idleProc**: PCB process idle, saat tidak ada process lain sedang jalan
- **readyHead**: pointer ke awal dari queue ready process
- **readyTail**: pointer ke akhir dari queue ready process
- **pcbPool**: array PCB
- **memoryMap**: array yang menandakan segmen memori yang sedang digunakan

Selain itu, pada awal fungsi main kernel anda (sebelum makeInterrupt21 dipanggil), tambahkan kode berikut:

```
initializeProcStructures();
```

Hal di atas akan melakukan inisiasi variabel-variable global PCB penting yang didefinisikan sebelumnya.

Untuk melakukan kompilasi kernel anda sekarang, ada terlebih dahulu harus melakukan kompilasi **proc.c** dan melakukan linking kernel anda dengan object file hasil kompilasi tersebut. Sesuaikan compileOS.sh anda dengan perintah-perintah berikut:

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

c. Implementasi handleTimerInterrupt

Untuk melakukan scheduling dan context switching process, sistem operasi anda membutuhkan sebuah timer interrupt, yaitu interrupt yang secara teratur dipanggil otomatis. Hal ini diperlukan supaya scheduling dan context switching juga dilakukan secara teratur.

Pada kit milestone 3 sudah tersedia **kernel.asm** dan **lib.asm** yang mengimplementasikan timer interrupt yang dipanggil setiap 1/12 detik. Nomor interrupt yang digunakan adalah 0x08. Salinlah file-file tersebut ke direktori sistem operasi kalian.

kernel.asm sekarang membutuhkan implementasi dari handleTimerInterrupt, yang dipanggil setiap kali terjadi timer interrupt. handleTimerInterrupt memiliki parameter segment dan stackPointer yang masing-masing merupakan segmen memori dan alamat stack pointer dari process yang diinterrupt oleh timer interrupt. Hal yang dilakukan oleh handleTImerInterrupt adalah memasukkan process yang diinterrupt (yang bukan PAUSED) ke ready queue. Setelah itu, top dari ready queue dibuat menjadi program yang dijalankan.

Gunakan kode berikut untuk mengimplementasikan handleTimerInterrupt:

```
void handleTimerInterrupt(int segment, int stackPointer) {
  struct PCB *currPCB;
  struct PCB *nextPCB;

  setKernelDataSegment();
  currPCB = getPCBOfSegment(segment);
  currPCB->stackPointer = stackPointer;
  if (currPCB->state != PAUSED) {
    currPCB->state = READY;
    addToReady(currPCB);
  }

  do {
    nextPCB = removeFromReady();
  }
  while (nextPCB != NULL && (nextPCB->state == DEFUNCT ||
nextPCB->state == PAUSED));

  if (nextPCB != NULL) {
    nextPCB->state = RUNNING;
    segment = nextPCB->segment;
    stackPointer = nextPCB->stackPointer;
    running = nextPCB;
  }
  else {
```

```
    running = &idleProc;
  }
  restoreDataSegment();

  returnFromTimer(segment, stackPointer);
}
```

Perhatikan bahwa setiap kali kode kernel ingin mengakses variabel-variabel ataupun fungsi-fungsi dari proc.h, kode tersebut harus diawali dengan `setKernelDataSegment()` dan `restoreDataSegment()`. Hal ini dibutuhkan karena pada pemanggilan syscall atau interrupt, data segment yang digunakan adalah data segment program yang memanggil syscall tersebut. Akan tetapi, variabel-variabel global proc.h berada pada data segment kernel sendiri. Oleh karena itu setiap kali anda butuh mengakses variabel-variabel dan fungsi-fungsi tersebut, anda harus terlebih dahulu membuat register DS meruju ke data segment kernel dengan `setKernelDataSegment()`. Dan setelah selesai kembali ke data segment program dengan `restoreDataSegment()`. Jika ingin mengakses data dan pointer-pointer program user, harus dilakukan setelah `restoreDataSegment()` atau sebelum `setKernelDataSegment()`.

Selain itu, pada awal fungsi main kernel anda (setelah makeInterrupt21 dipanggil), tambahkan kode berikut:

```
makeTimerInterrupt();
```

Pemanggilan fungsi tersebut berfungsi untuk mempersiapkan interrupt vector dari timer interrupt (interrupt 0x08).

d. Implementasi syscall yieldControl

Buat syscall yieldControl yang mengembalikan kontrol dari sebuah process ke sistem operasi meskipun time-slice 1/12 detiknya belum berakhir. Hal ini dapat dengan mudah dilakukan dengan cara memanggil timerInterrupt 0x08 secara manual.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x30 dan memiliki implementasi sebagai berikut:

```
void yieldControl () {
  interrupt(0x08, 0, 0, 0, 0);
}
```

e. Implementasi syscall sleep

Buat syscall sleep yang membuat state dari process yang sedang berjalan menjadi PAUSED. Hal ini akan menyebabkan process tersebut tidak dimasukkan kembali ke ready cycle sampai diresume oleh process lain.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x31 dan memiliki implementasi sebagai berikut:

```
void sleep () {
  setKernelDataSegment();
  running->state = PAUSED;
  restoreDataSegment();
  yieldControl();
}
```

f. Implementasi syscall pauseProcess

Buat syscall pauseProcess yang membuat state dari process dengan segmen tertentu menjadi PAUSED. Hal ini akan menyebabkan process tersebut tidak dimasukkan kembali ke ready cycle sampai diresume oleh process lain. Syscall ini mengembalikan SUCCESS (0) jika ditemukan process yang dapat diresume dan NOT_FOUND (-1) jika tidak.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x32 dan memiliki implementasi sebagai berikut:

```
void pauseProcess (int segment, int *result) {
  struct PCB *pcb;
  int res;

  setKernelDataSegment();
  pcb = getPCBOfSegment(segment);
  if (pcb != NULL && pcb->state != PAUSED) {
      pcb->state = PAUSED;
      res = SUCCESS;
  }
  else {
      res = NOT_FOUND;
  }
  restoreDataSegment();

  *result = res;
}
```

g. Implementasi syscall resumeProcess

Buat syscall resumeProcess yang menjalankan kembali process yang sebelumnya dalam state PAUSED dan memasukkan process kembali ke ready queue. Syscall ini mengembalikan SUCCESS (0) jika ditemukan process yang dapat dipause dan NOT_FOUND (-1) jika tidak.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x33 dan memiliki implementasi sebagai berikut:

```
void resumeProcess (int segment, int *result) {
  struct PCB *pcb;
  int res;

  setKernelDataSegment();
  pcb = getPCBOfSegment(segment);
  if (pcb != NULL && pcb->state == PAUSED) {
     pcb->state = READY;
     addToReady(pcb);
     res = SUCCESS;
  }
  else {
     res = NOT_FOUND;
  }
  restoreDataSegment();

  *result = res;
}
```

h. Implementasi syscall killProcess

Buat syscall killProcess yang menghentikan sebuah process lain. Syscall ini mengembalikan SUCCESS (0) jika ditemukan process yang dapat dihentikan dan NOT_FOUND (-1) jika tidak.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x34 dan memiliki implementasi sebagai berikut:

```
void killProcess (int segment, int *result) {
  struct PCB *pcb;
  int res;

  setKernelDataSegment();
  pcb = getPCBOfSegment(segment);
  if (pcb != NULL) {
     releaseMemorySegment(pcb->segment);
     releasePCB(pcb);
```

```
      res = SUCCESS;
  }
  else {
      res = NOT_FOUND;
  }
  restoreDataSegment();

  *result = res;
}
```

i. Mengubah implementasi syscall readFile

Ubah syscall readFile sehingga parameter result tidak hanya mengembalikan SUCCESS (0) saat berhasil menemukan file, tetapi juga nilai indeks file yang ditemukan (0-31).

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

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

j. Mengubah implementasi syscall executeProgram

Ubah syscall executeProgram sehingga tidak menjalankan programnya secara langsung tetapi hanya mempersiapkan PCB dari process yang akan dijalankan dan menginisiasi program. Selain itu, executeProgram juga memanggil sleep sehingga process yang sedang berjalan berubah state menjadi PAUSED, dan mengassign process tersebut sebagai parent dari process yang baru sehingga dapat dijalankan kembali saat process yang baru berakhir.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x06 dan memiliki implementasi sebagai berikut:

```
void executeProgram (char *path, int *result, char parentIndex){
  struct PCB* pcb;
  int segment;
  int i, fileIndex;
  char buffer[MAX_SECTORS * SECTOR_SIZE];
  readFile(buffer, path, result, parentIndex);

  if (*result != NOT_FOUND) {
    setKernelDataSegment();
    segment = getFreeMemorySegment();
    restoreDataSegment();
```

```
    fileIndex = *result;
    if (segment != NO_FREE_SEGMENTS) {
      setKernelDataSegment();
      pcb = getFreePCB();
      pcb->index = fileIndex;
      pcb->state = STARTING;
      pcb->segment = segment;
      pcb->stackPointer = 0xFF00;
      pcb->parentSegment = running->segment;
      addToReady(pcb);
      restoreDataSegment();
      for (i = 0; i < SECTOR_SIZE * MAX_SECTORS; i++) {
        putInMemory(segment, i, buffer[i]);
      }
      initializeProgram(segment);
      sleep();
    }
    else {
      *result = INSUFFICIENT_SEGMENTS;
    }
  }
}
```

k. Mengubah implementasi syscall terminateProgram

Ubah syscall terminateProgram sehingga tidak hanya mengeksekusi shell (karena shell hanya PAUSED selama process berjalan), tetapi mengakhiri process yang sedang berjalan dan menjalankan kembali process parentnya (tidak selalu shell).

Syscall ini dipanggil melalui interrupt 0x21 AL=0x07 dan memiliki implementasi sebagai berikut:

```
void terminateProgram (int *result) {
  int parentSegment;

  setKernelDataSegment();
  parentSegment = running->parentSegment;
  releaseMemorySegment(running->segment);
  releasePCB(running);
  restoreDataSegment();
  if (parentSegment != NO_PARENT) {
     resumeProcess(parentSegment, result);
  }
  yieldControl();
}
```

l. Mengubah implementasi syscall readString

Ubah syscall readString sehingga memanggil terminateProgram() jika pengguna menginput Ctrl+C, dan memanggil sleep() dan menjalankan kembali shell (menggunakan resumeProcess) jika pengguna menginput Ctrl+Z. readString juga menerima parameter tambahan yaitu disableProcessControls yang mengnonaktifkan aksi Ctrl+C dan Ctrl+Z tersebut (digunakan untuk shell supaya shell tidak dihentikan).

Syscall ini dipanggil melalui interrupt 0x21 AL=0x01 dan memiliki signature sebagai berikut:

```
void readString(char *string, int disableProcessControls);
```

2. Mengubah shell sistem operasi

a. Mengubah implementasi input dan perintah menjalankan program

Sesuaikan perintah tersebut dengan perubahan-perubahan sebelumnya.

b. Implementasi perintah menjalankan program secara paralel

Buatlah perintah khusus yang menjalankan program secara paralel. Eksekusi pararel berarti saat executeProgram process sebelumnya tidak menjadi pause, dan saat terminateProgram process parentnya tidak dijalankan kembali karena tidak dipause sebelumnya. Anda dapat mengubah proc.c, proc.h, handleTimerInterrupt, dan syscall lainnya untuk melakukan ini jika diperlukan.

Perintah ini dipanggil dengan syntax yang hampir sama dengan menjalankan program biasa, tetapi di akhirnya adalah karakter ‘&’. Misal untuk menjalankan program ‘myprog’ dengan argumen ‘abc’ dan ‘def’ secara paralel digunakan perintah seperti berikut:

```
$ myprog abc def &
```

c. Implementasi perintah pause, resume, dan kill

Buatlah supaya shell anda dapat menerima perintah-perintah pause yang memanggil pauseProcess, resume yang memanggil resumeProcess, dan kill yang memanggil killProcess. Semua perintah tersebut menerima sebuah argumen yaitu pid (0, 1, 2, 3, 4, 5, 6, 7) yang dipetakan secara langsung ke segmen memori dimana process dapat berada (0x2000, 0x3000, 0x4000, 0x5000, 0x6000, 0x7000, 0x8000, 0x9000).

3. Membuat program utilitas

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

a. Melakukan pemanggilan fungsi enableInterrupts untuk semua user program

Semua user program sekarang harus memasukkan kode berikut di awal fungsi mainnya. [Update 2018-04-14]

```
enableInterrupts();
```

Hal ini supaya interrupt diaktifkan pada user program dan timerInterrupt dapat mengambil alih kontrol dari programnya.

b. Membuat program ps

Program ps menampilkan daftar process yang sedang berjalan beserta informasi mengenai process tersebut seperti PID-nya dan statusnya. Detil implementasi diserahkan kepada anda.

4. Menjalankan program pengujian

Detil pengerjaan program pengujian akan diberitahukan sekitar 1 minggu sebelum deadline lewat milis.

5. Bonus

a. Implementasi syscall timedSleep

Buat sebuah syscall timedSleep dimana seperti syscall sleep sebelumnya, tetapi process akan diresume secara otomatis setelah beberapa tick (1/12 detik). Anda dapat mengubah proc.c, proc.h, handleTimerInterrupt, dan syscall lainnya untuk melakukan ini jika diperlukan.

Syscall ini dipanggil melalui interrupt 0x21 AL=0x40 dan memiliki implementasi sebagai berikut:

```
void timedSleep(int ticks);
```

b. Membuat program timer

Program timer seperti program echo sebelumnya, tetapi menuliskan teks ke layar setelah beberapa tick yang merupakan argumen kedua dari programnya. Gunakan timedSleep untuk melakukan hal tersebut.

c. Lain-lain

Anda diperbolehkan untuk mengimplementasikan fitur-fitur lain yang berhubungan dengan Process dan Multiprogramming pada sistem operasi kalian. Kreativitas anda akan dihargai dengan nilai bonus.

IV. Pengumpulan dan Deliverables

1. Buat sebuah file **zip** dengan nama **IF2230-TugasBesar-Milestone3-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. proc.c
        iv. proc.h
        v. compileOS.sh
        vi. Source code program-program utilitas
    b. Folder **doc**, berisi file laporan dengan nama file **IF2230-TugasBesar-Milestone3-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.