

Tugas Besar - Milestone 3
IF2230 - Sistem Operasi
“Grand Finale”
Pembuatan Sistem Operasi Sederhana
Process, Multiprocessing



Dipersiapkan oleh:

Muhammad Rafi Zhafran	13517005
Abiyyu Avicena Ismunandar	13517083
Jeremy Arden Hartono	13517101

Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Bandung 40132

LANGKAH Pengerjaan

1. Mengubah kernel sistem operasi

a. Memperbesar ukuran maksimum kernel

Untuk milestone ini, alokasi ukuran maksimum kernel sekarang (10 sektor) mungkin tidak cukup. Oleh karena itu, kami menggunakan `bootload.asm`, `map.img`, `files.img`, dan `sectors.img` yang telah disesuaikan untuk ukuran maksimum kernel yang baru (16 sektor) yang didapat dari kit milestone 3.

b. Implementasi Process Control Block (PCB)

Untuk menambahkan process dan multiprogramming pada sistem operasi kalian, dibutuhkan struktur data yaitu Process Control Block (PCB). Sebuah PCB mengandung informasi penting mengenai suatu process seperti segmennya, alamat stack pointer-nya, dan scheduling state-nya. Informasi-informasi tersebut digunakan oleh sistem operasi untuk melakukan scheduling dan context switching antar process. Maka dari itu kami menggunakan `proc.c` dan `proc.h` yang mengimplementasikan PCB yang didapat dari kit milestone 3.

c. Implementasi `handleTimerInterrupt`

Untuk melakukan scheduling dan context switching process, sistem operasi kalian membutuhkan sebuah timer interrupt, yaitu interrupt yang secara teratur dipanggil otomatis. Hal ini diperlukan supaya scheduling dan context switching juga dilakukan secara teratur. Maka dari itu kami menggunakan `kernel.asm` dan `lib.asm` yang mengimplementasikan timer interrupt yang dipanggil setiap 1/12 detik yang didapat dari kit milestone 3.

d. Implementasi syscall `yieldControl`

Kami membuat 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.

e. Implementasi syscall `sleep`

Kami membuat 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 di-resume oleh process lain.

f. Implementasi syscall `pauseProcess`

Kami membuat 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 di-resume oleh process lain. Syscall ini mengembalikan SUCCESS (0) jika ditemukan process yang dapat diresume dan NOT_FOUND (-1) jika tidak.

g. Implementasi syscall `resumeProcess`

Kami membuat 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 di-pause dan NOT_FOUND (-1) jika tidak.

h. Implementasi syscall `killProcess`

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

i. Implementasi syscall readFile

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

j. Implementasi syscall executeProgram

Kami mengubah 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 meng-assign process tersebut sebagai parent dari process yang baru sehingga dapat dijalankan kembali saat process yang baru berakhir.

Syscall normalnya mengembalikan indeks file pada parameter result, tetapi jika file tidak ada mengembalikan NOT_FOUND (-1), dan jika jumlah segment yang tersedia tidak cukup mengembalikan INSUFFICIENT_SEGMENTS (-2).

k. Implementasi syscall terminateProgram

Kami mengubah syscall terminateProgram sehingga tidak hanya mengeksekusi shell (karena shell hanya PAUSED selama process berjalan), tetapi mengakhiri process yang sedang berjalan dan menjalankan kembali process parent-nya (tidak selalu shell).

l. Implementasi syscall readString

Kami mengubah 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 menonaktifkan aksi Ctrl+C dan Ctrl+Z tersebut (digunakan terutama untuk shell supaya shell tidak dihentikan).

2. Mengubah shell sistem operasi

a. Mengubah implementasi input dan perintah menjalankan program

Kami menyesuaikan perintah dalam shell agar kompatibel dengan perubahan di atas.

b. Implementasi perintah menjalankan program secara paralel

Kami membuat perintah khusus yang menjalankan program secara paralel. Eksekusi paralel berarti saat executeProgram process sebelumnya tidak menjadi pause, dan saat terminateProgram process parent-nya tidak dijalankan kembali karena tidak di-pause sebelumnya. Kami 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

Kami membuat supaya shell kami dapat menerima 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

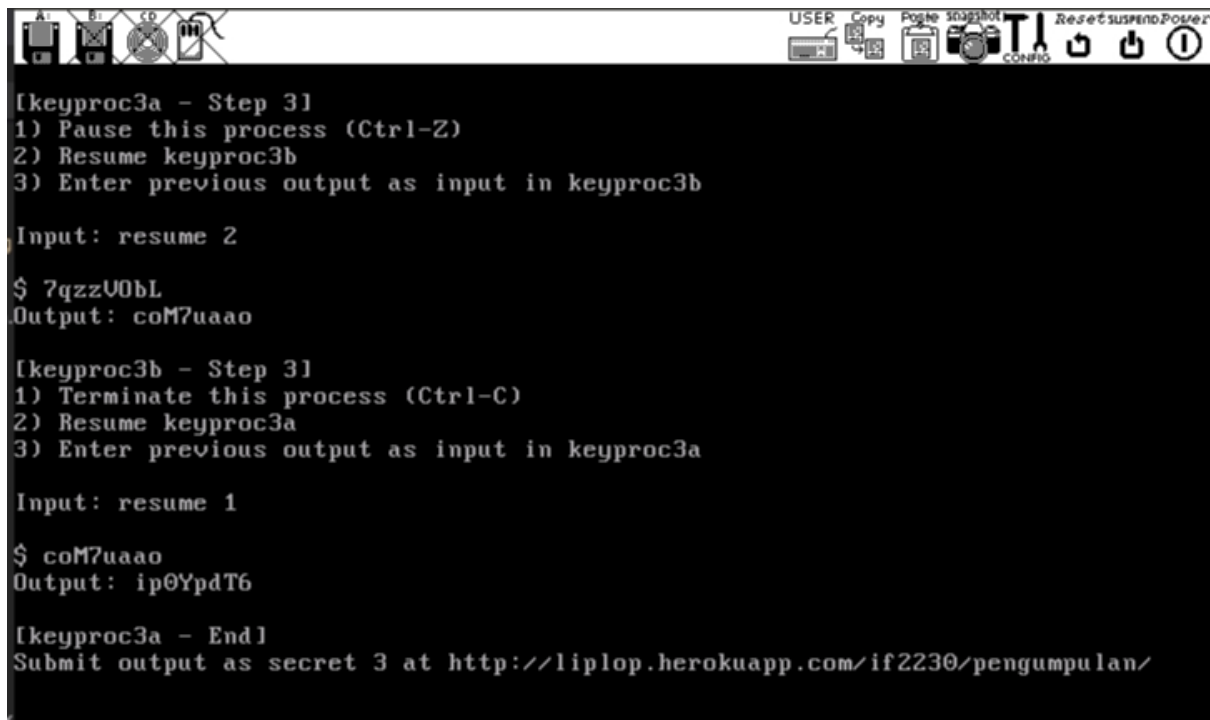
- a. Melakukan pemanggilan fungsi `enableInterrupts` untuk semua user program
Semua user program harus memasukkan kode berikut di awal fungsi mainnya.

```
enableInterrupts();
```

Hal ini dilakukan agar interrupt diaktifkan pada saat eksekusi 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 status-nya. Detil implementasi diserahkan kepada kalian.



```
[keyproc3a - Step 3]
1) Pause this process (Ctrl-Z)
2) Resume keyproc3b
3) Enter previous output as input in keyproc3b

Input: resume 2

$ 7qzzU0bL
Output: coM7uaao

[keyproc3b - Step 3]
1) Terminate this process (Ctrl-C)
2) Resume keyproc3a
3) Enter previous output as input in keyproc3a

Input: resume 1

$ coM7uaao
Output: ip0YpdT6

[keyproc3a - End]
Submit output as secret 3 at http://liplop.herokuapp.com/if2230/pengumpulan/
```

PEMBAGIAN TUGAS

NIM	Nama	Apa yang dikerjakan	Presentasi Kontribusi
13517005	Muhammad Rafi Zhafran	Shell dan Laporan	33%
13517083	Abiyyu Avicena Ismunandar	Kernel.c	33%
13517101	Jeremy Arden Hartono	Kernel.c	33%

KESULITAN Pengerjaan

Dalam mengerjakan tugas ini kami merasa sangat kesulitan karena tidak sesuai materi yang diajarkan di kelas dan yang ditugaskan, kurangnya bahan bacaan yang terpercaya di internet, dan kesulitan dalam memahami syscall – syscall dalam kernel.c. Selain itu ada beberapa algoritma yang kami buat yang menurut kami sudah benar, namun crash saat dijalankan.

Saran kedepannya adalah tolong cantumkan bahan bacaan yang terpercaya yang dianggap mampu membantu kami dalam mengerjakan tugas yang diberikan dan sesuaikan tingkat kesulitan tugas dengan materi yang disampaikan di kelas.