Laporan

Tugas Besar 1 IF2230 Sistem Operasi

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

oleh

Aidil Rezjki Suljztan Syawaludin / 13517070

M. Khairul Makirin / 13517088

Edward Alexander Jaya / 13517115

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2019

BAB I
JAWABAN PERTANYAAN

1. [Bootloader] Apa perbedaan booting disk MBR dengan GPT? Cara boot yang mana yang digunakan dalam tugas ini?

MBR adalah singkatan dari *Master Boot Record* dan GPT adalah singkatan dari *GUID Partition Table*. Sebelum *harddrive* dapat digunakan, *harddrive* harus dapat dipartisi terlebih dahulu. MBR dan GPT menyimpan informasi *partitioning* (berisi kapan *partition* mulai) yang berbeda. Tujuan dari informasi ini adalah OS dapat mengenali sektor mana yang terdapat partisi tertentu dan partisi manakah yang dapat di-*boot*.

MBR menggunakan *boot sector* spesial yang terdapat pada awal *harddrive*. *Bootloader* terdapat dalam sektor ini, yang digunakan untuk me-*load bootloader* yang lebih besar dari partisi lain yang terdapat dalam drive. MBR hanya bekerja pada *harddrive* dengan penyimpanan maksimum sebesar 2 TB. MBR juga hanya kompatibel hanya sampai dengan empat partisi.

GPT dipakai untuk menggantikan MBR secara perlahan. Menggunakan UEFI yang digunakan untuk menggantikan BIOS dengan sesuatu yang lebih modern. Setiap partisi di dalam *harddrive* memiliki *identifier* unik. *Identifier* yang dimaksud adalah GUID. GUID adalah string acak yang panjang, dengan kondisi panjang dari string tersebut dapat menjamin string unik. Drive yang menggunakan GPT bisa berukuran lebih besar daripada ukuran dari MBR. Selain itu, GPT memperbolehkan partisi sangat banyak, tergantung dari sistem operasi yang digunakan.

Menurut definisi MBR dan GPT, Cara *boot* yang dipakai dalam tugas ini adalah MBR karena terdapat *bootloader* yang digunakan.
Sumber:
https://www.howtogeek.com/193669/whats-the-difference-between-gpt-and-mbr-when-partitioning-a-drive/

2. [Kernel] Apa itu *kernel panic* dan mengapa itu bisa terjadi?

Kernel panic adalah error yang terjadi pada komputer. Pada sistem operasi Windows, *kernel panic* lebih dikenal sebagai *Blue Screen Of Death* (BSOD). *Kernel panic* menampilkan pesan atau beberapa pesan pada komputer. Kadang informasi pada *kernel panic* berguna untuk teknisi yang akan mendiagnosis masalah pada komputer.

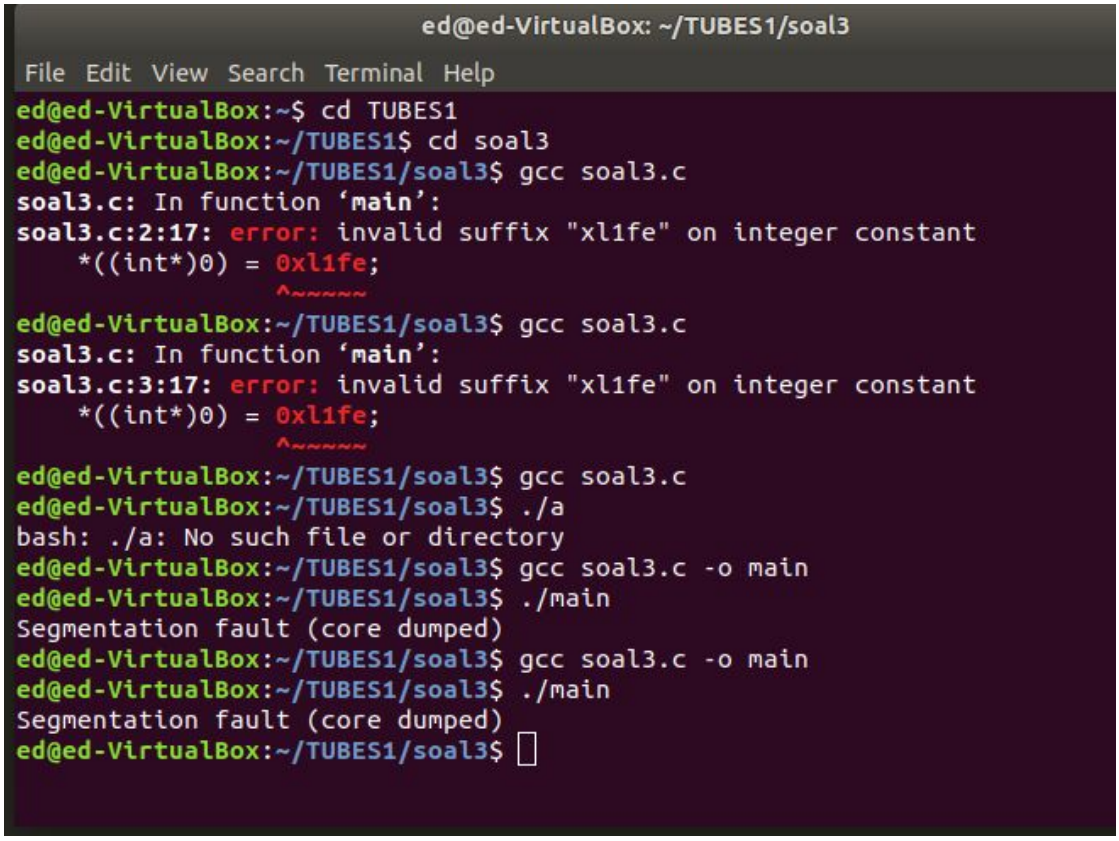
Kernel panic dapat terjadi karena percobaan tidak wajar oleh sistem operasi untuk mengakses atau menulis ke memori (RAM). Kadang bisa disebabkan oleh *bug* atau *malware*. Selain itu, masalah *hardware* dapat terjadi, yaitu kegagalan atau instalasi yang tidak wajar dari *chip* RAM, *data corruption*, *harddisk damage*, *chip* mikroprosesor yang cacat atau *driver* yang tidak kompatibel.

https://searchdatacenter.techtarget.com/definition/kernel-panic

3. [Write to Memory] Apa yg terjadi jika code snippet ini dijalankan di kernel mode dan di usermode?

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

Kode ini ditujukan untuk membuat suatu program *force crash*, kadang instruksi ini diperlukan untuk *debugging*. Namun pada kode di atas, terdapat compile error yang akan ditunjukkan dengan gambar di bawah ini:



Penulis mengganti 0xl1fe menjadi 0x1, lalu 0x1111. Nampaknya kedua perubahan ini menimbulkan *segmentation fault (core dumped)*.

Kode tersebut dapat menyebabkan *segmentation fault* karena (int *) 0 digunakan untuk membuat POINTER NULL. Isikan nilai / *value* pada POINTER NULL tersebut dan jadilah:

```
*((int*)0) = 0xl1fe;
```

POINTER NULL seharusnya tidak bisa diberi nilai, namun kode ini “memaksa”.

4. [Sector] Apa kelebihan dan kekurangan jika setiap sector pada suatu Hard Disk berukuran besar, misalkan 64MB?

Kelebihan dari ukuran *sector* suatu *hard disk* berukuran besar adalah dapat menyimpan data berukuran besar dengan lebih mudah dan mempermudah proses pengambilan data itu sendiri. Dengan semakin besar ukuran *sector*, maka pemanggilan data yang berukuran besar dapat diefisienkan. Misalkan ukuran *sector* 64MB, mengambil data berukuran 256MB memerlukan pemanggilan data pada berbagai *sector* sebanyak empat kali, sedangkan untuk ukuran *sector* 32MB memerlukan pemanggilan data pada berbagai *sector* sebanyak delapan kali. Dengan

semakin banyaknya pemanggilan data tersebut, maka semakin banyak juga *overhead* proses yang dilakukan, yang akan menyebabkan pengambilan data tidak efisien.

Kekurangan dari ukuran *sector* suatu *hard disk* berukuran besar adalah ketika menyimpan data dengan ukuran yang kecil (cukup jauh lebih kecil dari ukuran *sector*), maka akan ada banyak ruangan kosong (*space*) yang dipakai hanya untuk data yang berukuran kecil. Hal ini menyebabkan kemampuan penyimpanan menjadi tidak efisien.

Sumber: https://stackoverflow.com/questions/19473772/data-block-size-in-hdfs-why-64mb, https://stackoverflow.com/questions/12345804/difference-between-blocks-and-sectors

5. [Read/Write] Melakukan file I/O pada kernel cenderung dikatakan pelanggaran standard practice, mengapa?

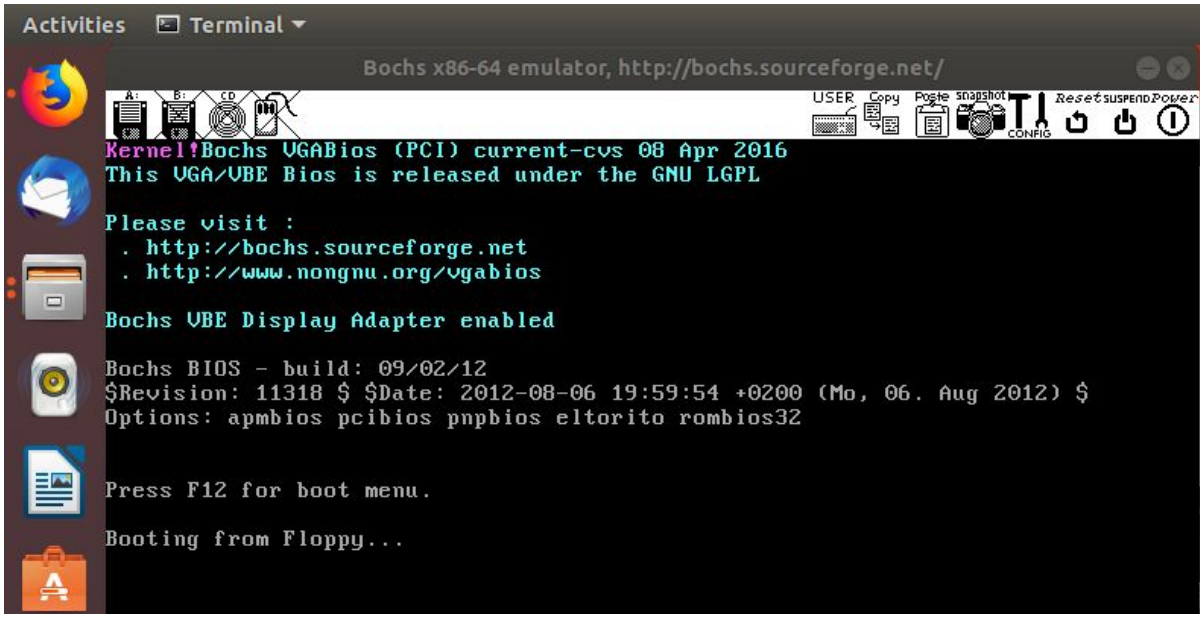
Dengan melakukan file I/O tepat di kernel, kemungkinan terjadinya buffer overflow sangat besar, sehingga dapat mengancam stabilitas dan keamanan sistem. Melakukan file I/O pada kernel juga berarti mengharuskan adanya interpreter pada kernel. Apabila terjadi sedikit saja kesalahan pada interpreter tersebut, sistem secara keseluruhan dapat mengalami gangguan. Selain itu, terdapat beberapa kekurangan dari file I/O langsung melalui kernel, seperti mengharuskan posisi file yang statik (tidak berubah). Apabila posisi file ingin diubah, maka kernel harus diubah.

Sumber: https://www.linuxjournal.com/article/8110

BAB II

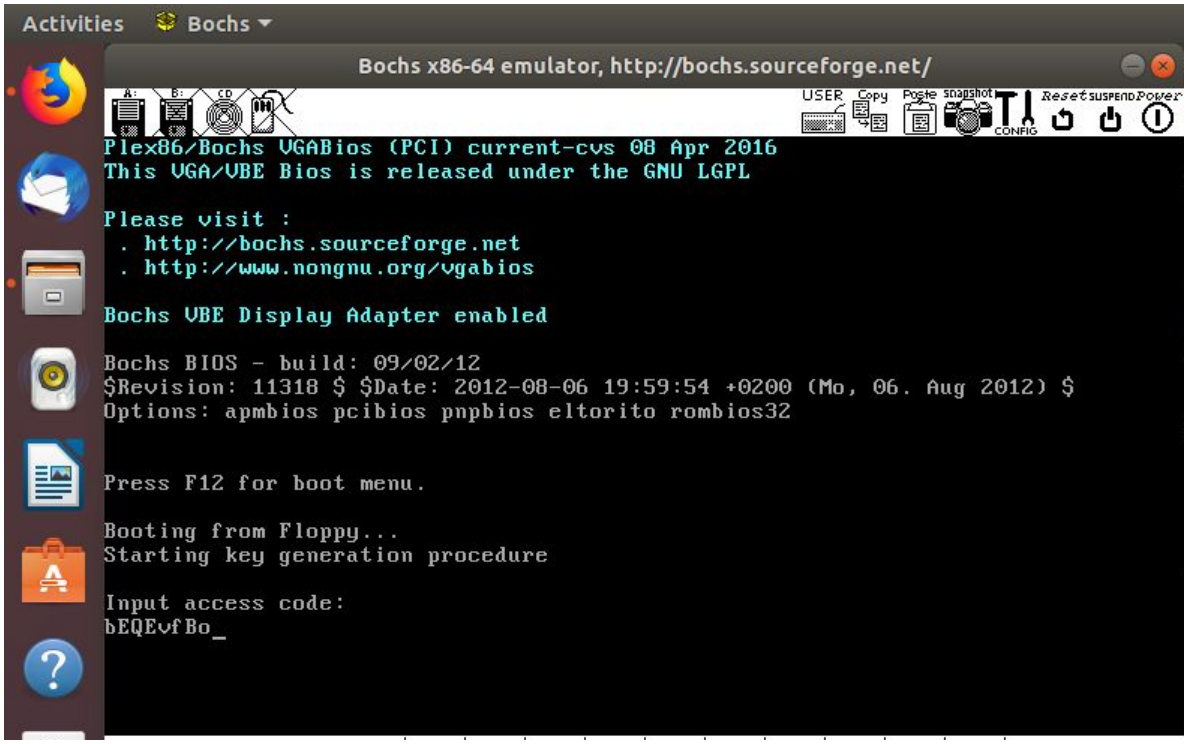
SCREENSHOT LANGKAH

1. **Langkah no 6:**

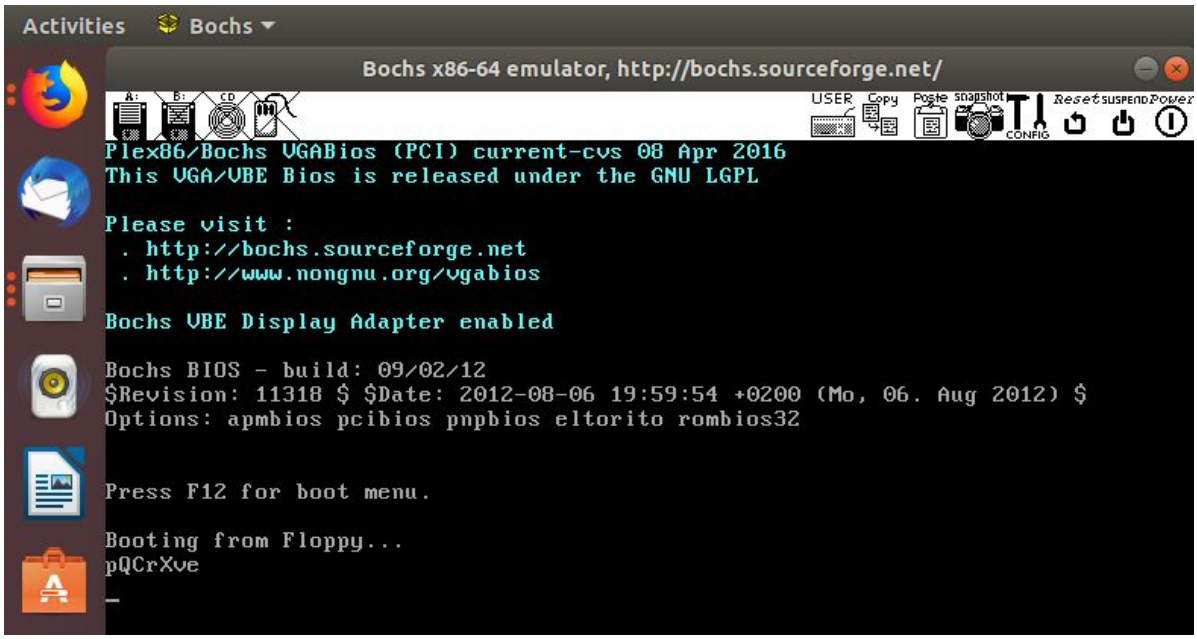


2. **Langkah no 8:**

(Load from keyproc)

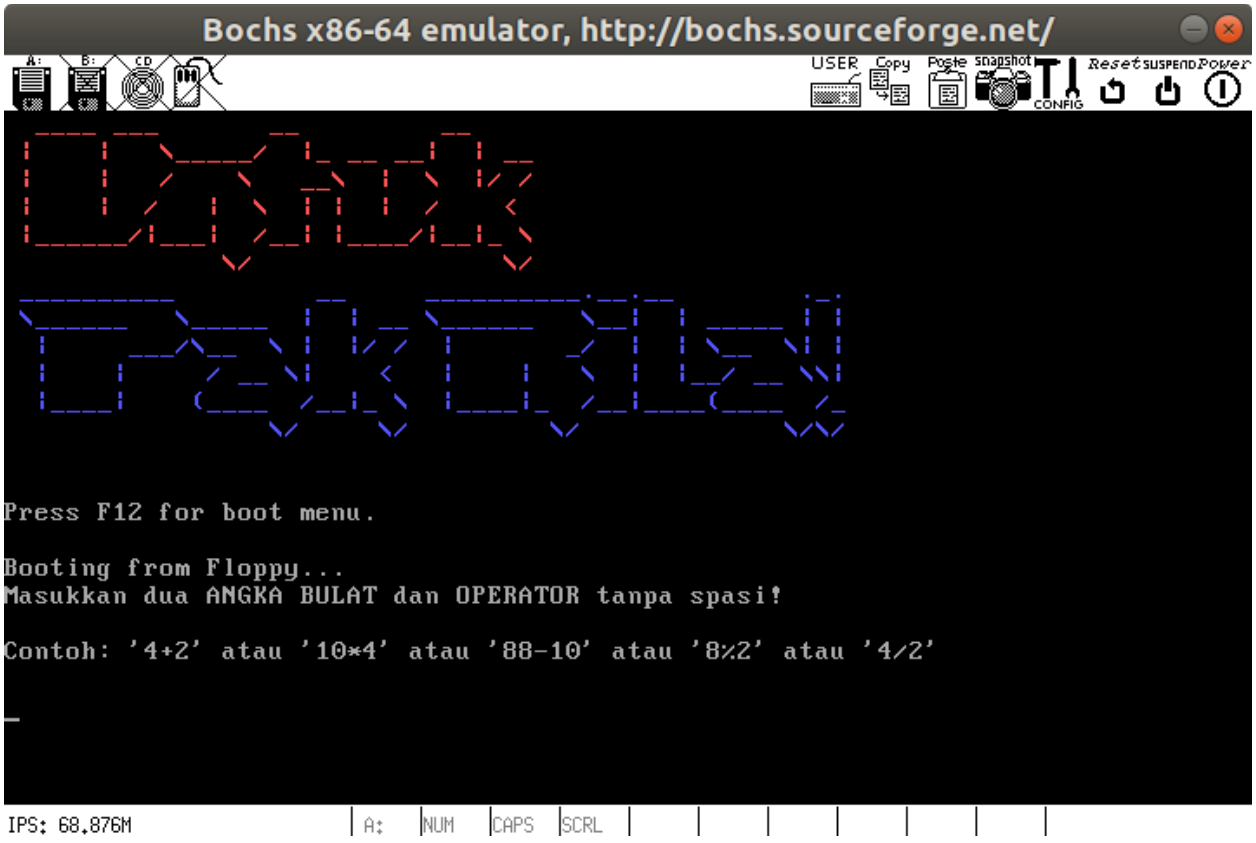


(Reading key.txt)

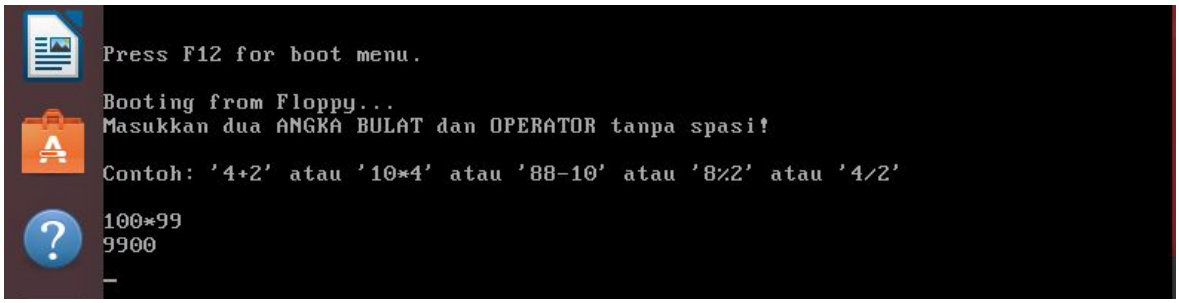


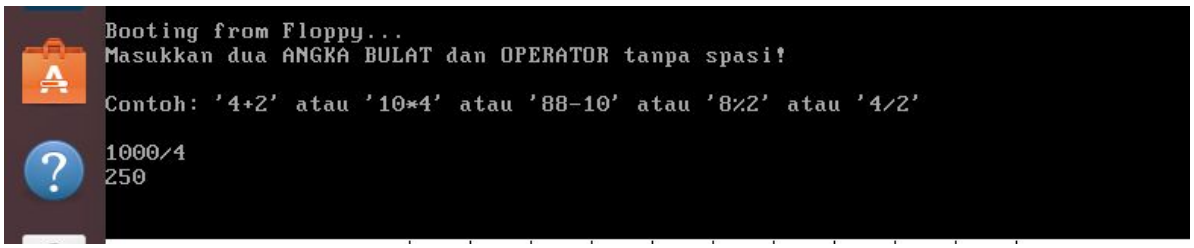
3. **Langkah no 9:**

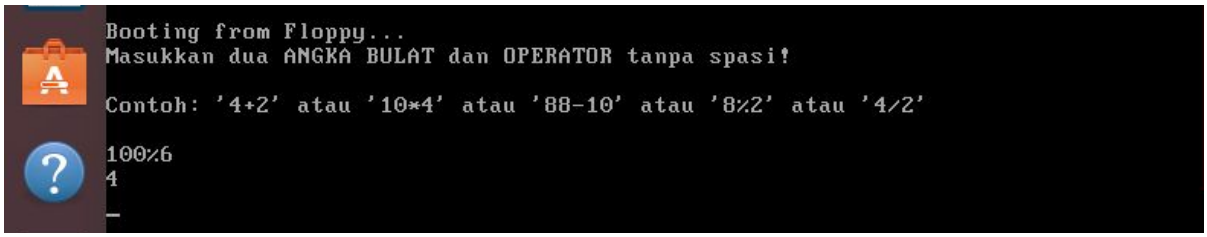
9a (logo):

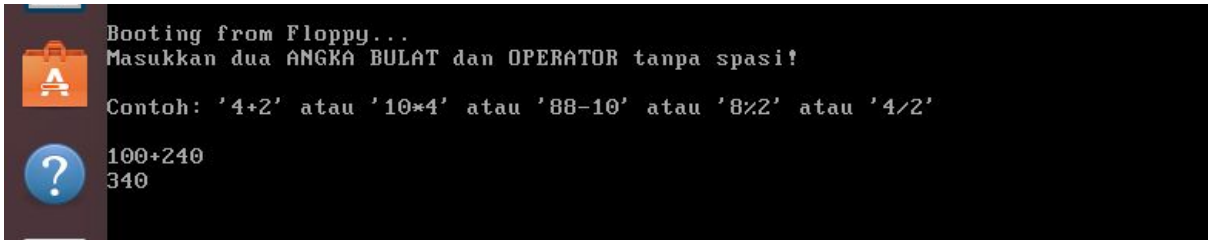


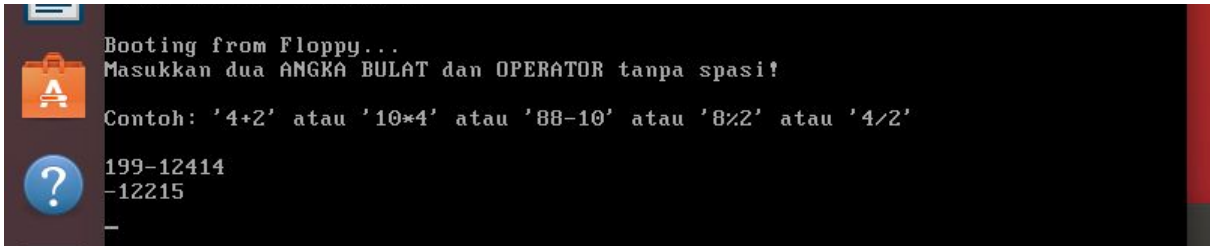
9b (kalkulator):











Perhatikan bahwa kernel.c memanggil handle interrupt pada file “program”.

BAB III

PEMBAGIAN TUGAS, KESULITAN, FEEDBACK

Pembagian Tugas:

NIM	Nama	Bagian kerja	Persentase kontribusi
13517070	Aidil Rezjki Suljztan Syawaludin	Mengerjakan dan mendebug kode readString, writeString, readFile, executeProgram hingga bochs meminta access code dan mengenerate key.txt Mengerjakan bonus 9a (logo). Laporan	33.33%
13517088	Muhammad Khairul Makirin	Membuat ulang (agar rapi) dan mendebug kode readString, writeString, readFile, executeProgram sehingga bisa membaca isi dari key.txt. Laporan	33.33%
13517115	Edward Alexander Jaya	Mendapat solusi untuk membaca isi dari key.txt, dengan mengubah kode main dan membuat file compileOSKey.sh. Mendebug kode readString, writeString, readFile, executeProgram. Mengerjakan bonus 9b (kalkulator). Laporan (BAB 1 no 1-3, BAB 2 semua kecuali 9a dan BAB 3).	33.33%

Kesulitan:

Sulit ketika mendebug kode, karena bukan gcc yang digunakan melainkan bcc. Alasan yang lain, karena kami menggunakan gedit untuk tugas ini (kecuali untuk no 9b menggunakan sublime

text) sehingga banyak kesalahan seperti tidak tahu line pada kode, deklarasi variabel bukan di atas eksekusi, memanggil fungsi yang tidak perlu di main(). Contohnya, tim penulis mendebug readString namun salahnya terletak pada fungsi yang tidak perlu di main(). Contoh lain, compiler mengeluarkan error pada line code ke-x. Namun gedit tidak menyediakan fitur untuk melihat line code, sehingga kami menggunakan c formatter di Internet dan melihat kode pada line yang disebut di compiler. Lalu, tim penulis bingung ketika program nampaknya sudah benar secara sintaks, namun terdapat satu baris kode yang menggunakan deklarasi variabel tidak di atas kode eksekusi. Tim penulis juga bingung ketika program sudah meminta *access code* namun tim penulis kesulitan membaca isi key.txt. Sekitar kurang lebih 40%-50% waktu untuk mengerjakan bagian wajib dihabiskan untuk mendebug readFile (yang tidak ada salah) sampai salah satu orang dari tim penulis berinisiatif untuk mengganti kode untuk membaca keyproc dengan key.txt dan membuat compileOSKey.sh.

Feedback:

Tugas Besar ini menarik untuk dikerjakan. Benar-benar mengeksplorasi dunia OS dan cukup menantang. Namun, a*ccess code* yang diminta adalah 8 karakter dan tim penulis diberikan *access code* 15 karakter. Mungkin jika tugas berikutnya menggunakan *access code*, panjang a*ccess code* yang diberikan disamakan dengan panjang a*ccess code* aslinya. Untuk soal 3 pada bagian pertanyaan, `0xl1fe;` tidak bisa di-*compile*. Mohon untuk lebih diperhatikan. Selain itu, spesifikasi sudah jelas dan tidak membuat bingung. Tambahan lagi, program dari asisten melakukan *infinite looping* sehingga cukup membuat bingung saat pengerjaan. Akan tetapi, tugasnya keren!