

**Tugas Besar  
IF2230 - Sistem Operasi  
Milestone 1 of ??**

**"*HolOS*"**

**Pembuatan Sistem Operasi x86  
Booting, Kernel, 32 bit Protected Mode**

Dipersiapkan oleh :  
Asisten Lab Sistem Terdistribusi

Didukung Oleh :

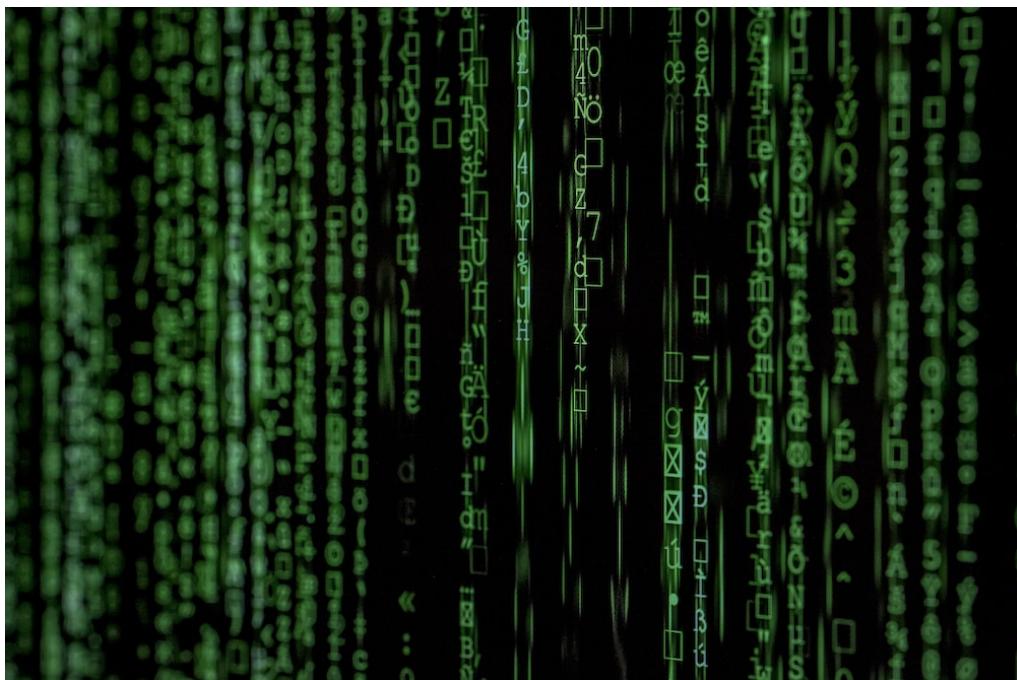


**Waktu Mulai :**  
Jum'at, 10 Februari 2023, 18.00 WIB

**Waktu Akhir :**  
Kamis, 2 Maret 2023, 23.59 WIB

## I. Latar Belakang

"D-dimana aku...?" Kamu terbangun di antah berantah. "WOOOOEEEEAAAA AKU DIMANAAA?!?!?!" Kamu mendengar suara gema kamu di sekitar tempat itu. Kamu mengucek-ngucek mata kamu agar bisa fokus melihat apa-apa yang ada di sekitarmu. Kamu menyimpulkan ternyata kamu berada di tengah-tengah lautan... bit? yang berantakan dan tidak terlihat satupun hal yang familiar pada dunia itu.



Kalau asisten bilang di sini ada kode rahasia, bakal percaya ga ya?

"Siapa saja, tolong jawab akuuuuu?!?!?!" kamu berteriak namun tidak ada jawaban. Kamu berusaha mengingat apa yang terjadi hingga kamu dapat berada pada situasi yang absurd ini. "Uncover Corps... Yadu... ruang bawah tanah... PA UNA! Hanya dia yang bisa membantuku saat ini! Bahkan hingga saat terakhir dia selalu membuatku kesulitan dengan ~~praktikum yang semakin sulit setiap tahunnya~~ jurusnya yang sangat sulit untuk dipatahkan.

Kamu melihat ke sana kemari ~~tanpa tertawa~~ tapi kamu tidak melihat gadis itu. "Hah? Aku ditinggal? Pa Una?!?!" Kamu menyadari kalau Pa Una tidak bersama kamu saat ini. Kamu kembali mencoba mengingat apa-apa yang terlewat, dan teringat akan seseorang.

"Benar, orang itu! Ke mana orang itu membawaku sekarang?" sedikit demi sedikit kamu berhasil menyusun kembali ingatanmu yang acak-acakan akibat jurus teleportasi milik neng yang rambutnya putih panjang, sampai sekarang mungkin kamu masih belum tahu

namanya. Berjalan di dalam kebingungan tanpa tahu apa yang harus kamu perbuat membuatmu semakin gila. "Apa yang harus kulakukan??? Bagaimana caraku keluar dari sini sekarang?!?!" KAPAN AKU BISA BERTEMU PEKOLAAAAAAAAAAAAAAA????????!?!?!?!?!?" pikiranmu semakin gila semakin lama kamu berada di lautan bit ini hingga akhirnya kamu melihat sebuah lubang dan cahaya. Berlari ke arah lubang dengan ekspektasi tinggi akhirnya kamu dapat melihat seseorang dari kejauhan.

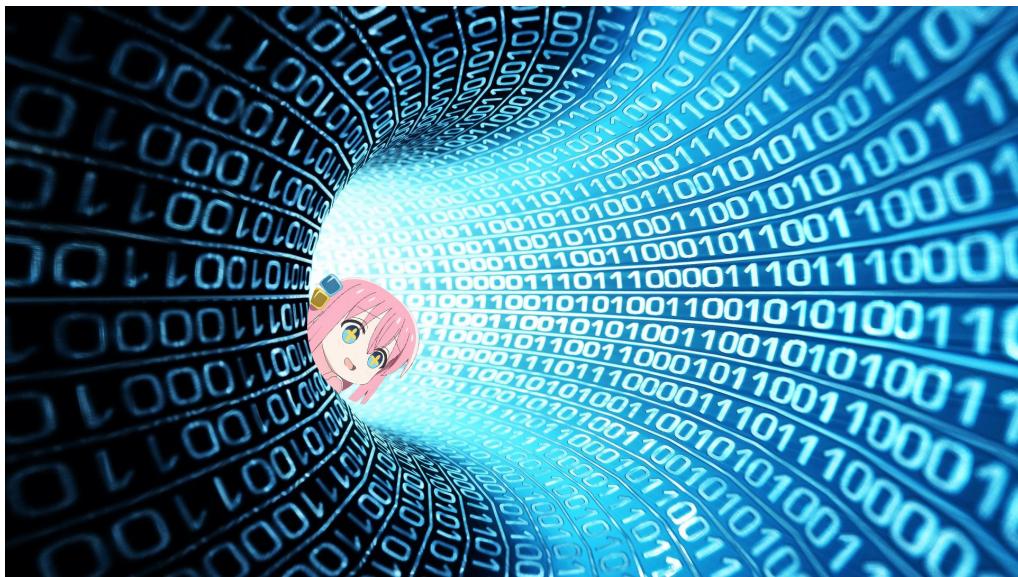
"AAAAAAAFJAPSDO;IFJAW;ELKVNZ.XM,CVNKJHADLFHAWUOERHQNBL1K2H3LJKWHIUDYFO0A92KJBNVMNZXBhttps://www.youtube.com/watch?v=dQw4w9WgXcQ;SHLJKAHFATWETY5OIQ2U4HLJKDBNVZXBCVLHIjkhdfkjashKJHLKUHFDSUHjh" kamu mendengar orang yang setengah nge-*glitch* tersebut berbicara. Kamu yang senang karena akhirnya dapat bertemu orang lain langsung menghampirinya. "Siapa namamu? Kenapa kamu ada di sini, apakah kamu juga dilempar oleh neng itu ke dimensi ini?".

"Makhluk pink tersebut sangat kaget melihatmu dan langsung lari menjauh sambil mengintip dari belokan. "D-d-d-d-docchi desu. A-a-a-ku sudah lama berada d-d-d-di sini. T-t-t-t-teman satu b-b-b-and ku memaksaku u-u-untuk b-b-b-ekerja dan aku d-d-d-dilempar ke sini untuk m-m-m-m-membangun menyusun semua perintah komputer untuk membangun s-s-s-sistem operasi desu A-a-a-aku tidak bisa pulang karena sistem t-t-t-teleportasi rusak desu. N-n-n-namun s-s-s-setiap aku menyusun satu hal, b-b-b-bagian lain r-r-r-r-usak desu," makhluk pink tersebut menjawab sambil malu-malu (~~pengalaman reat taun lalu no fek no root~~).

"Kamu terkejut, karena kamu belum pernah menyentuh pemrograman yang sangat primitif seperti ini dan takut dengan apa yang mungkin akan terjadi. Meskipun demikian, demi menyelamatkan diri ~~dan bertemu osihmu~~ kembali ke bumi, kamu pun memberanikan diri "Tidak apa Docchi-san! Mungkin ini terlihat sulit dan bahkan banyak hal yang tidak kita mengerti namun jika kita melangkah sedikit demi sedikit aku percaya kita pasti dapat membuatnya!" Kamu menghibur Docchi-san yang sudah turun resolusinya ke 144p karena stres besar dan kekurangan tidur.

"B-b-b-benarkah? Mari kita lakukan!" Docchi kembali bersemangat karena akhirnya dia mempunyai teman untuk mengerjakan hal ini bersama-sama. Kamu pun tidak mempunyai pilihan selain menerima nasibmu ~~yang akan ngoding sampai matahari terbit~~ yang tidak akan bisa keluar selain menyelesaikan ini bersama Docchi.

~~Oh iya, kamu bakal tahu dari mana matahari akan terbit? kan terjebak di lubang antah berantah h3h3...~~



:D

## II. Deskripsi Tugas

Tugas ini akan membuat sebuah program mistis yang umumnya tidak diketahui orang awam bernama sistem operasi. Sistem operasi yang akan dibuat akan berjalan pada arsitektur x86 32 bit yang nanti akan dijalankan dengan *emulator* QEMU. Tugas ini akan dibagi menjadi beberapa *milestone* dan berikut adalah hal-hal yang akan dikerjakan di *milestone* pertama

- Menyiapkan alat & repository
- Pembuatan build script
- Menjalankan sistem operasi
- Membuat output dengan text
- Memasuki Protected Mode

## III. Langkah Pengerjaan

### 3.0. Outline Pengerjaan

Sebelum memulai berikut adalah garis besar dan checklist pengerjaan milestone pertama. Bagian 3.1 hingga 3.6 merupakan tahap **get one's feet wet** yang berfokus pada persiapan alat-alat dan *setup dev environment*

- **3.1. Menyiapkan alat dan repository**
  - Instalasi tools
  - Github Classroom & Repository
- **3.2. Booting Sequence & GRUB**
  - Membuat menu.lst
- **3.3. Kernel Sederhana**
  - Kernel sederhana dengan C
  - Linker script untuk lokasi memory & alignment
  - Kompilasi kernel
- **3.4. Pembuatan Disk Image**
  - Penyusunan folder & pembuatan disk image
- **3.5. Build Script**
  - Otomatisasi kompilasi kernel
  - Otomatisasi pembuatan folder disk image
  - Otomatisasi pembuatan disk image
- **3.6. Menjalankan Sistem Operasi**
  - Menjalankan OS dengan QEMU

Bagian 3.7 merupakan awal dan pemanasan dari OS development

- **3.7. Kernel Development & Simple Text Framebuffer**
  - Memahami "T&C" OS development
  - framebuffer\_write()
  - framebuffer\_set\_cursor()
  - framebuffer\_clear()

Untuk dua bagian terakhir, 3.8 dan 3.9, akan memasuki 32-bit protected mode

- **3.8. Pembuatan Global Descriptor Table (GDT)**
  - Definisi **struct** SegmentDescriptor
  - Definisi global\_descriptor\_table
  - Definisi \_gdt\_gdtr
- **3.9. Memasuki Protected Mode**
  - 1 baris asm untuk membaca GDT
  - 1 baris asm untuk menyalakan flag Protected Mode
  - 2 baris asm untuk mengupdate segment register
  - Berhasil mengeksekusi enter\_protected\_mode()

### 3.1. Menyiapkan alat dan repository

Semua instruksi berikut ini akan mengasumsikan dijalankan instruksi Linux. Untuk program-program yang sudah harus terinstal pada sistem operasi Anda adalah sebagai berikut

- **Netwide assembler** (<https://www.nasm.us/>)  
*Compiler assembly utama, untuk kode yang membutuhkan instruksi khusus tertentu*
- **GNU C Compiler** (<https://man7.org/linux/man-pages/man1/gcc.1.html>)  
*Compiler C utama untuk sistem operasi*
- **GNU Linker** (<https://linux.die.net/man/1/ld>)  
*Linker object code hasil kompilasi*
- **QEMU - System i386** (<https://www.qemu.org/docs/master/system/target-i386.html>)  
*Emulator utama untuk menjalankan sistem operasi*
- **GNU Make** (<https://www.gnu.org/software/make/>)  
*Build tools untuk sistem operasi*
- **genisoimage** (<https://linux.die.net/man/1/genisoimage>)  
*Tool untuk pembuatan image sistem operasi*

**Sangat dianjurkan untuk memakai sistem operasi Ubuntu 20.04.** Pada Ubuntu 20.04 berikut adalah perintah yang dapat digunakan untuk menginstal program-program di atas.

**Tugas besar ini diuji oleh asisten pada arsitektur x86-64 dan Windows 10 & 11.** Sebagian besar tools seharusnya dapat dijalankan tanpa masalah pada arsitektur-arsitektur yang ada pada *consumer pc*. Jika mengalami kendala dengan *tools*, dapat dicoba untuk mencari solusi terlebih dahulu menggunakan *search engine* dan dapat menggunakan sheet QnA jika masih mengalami kendala.

```
sudo apt update  
sudo apt install gcc nasm make qemu-system-x86 genisoimage
```

Direkomendasikan untuk menggunakan **Visual Studio Code** karena akan memudahkan *debugging* dan lebih mudah dalam menjalankan OS. Template akan menyediakan konfigurasi *debugging* dan *build tasks* untuk Visual Studio Code. Untuk selain Visual Studio Code, konfigurasi masing-masing IDE dapat dieksplorasi sendiri.

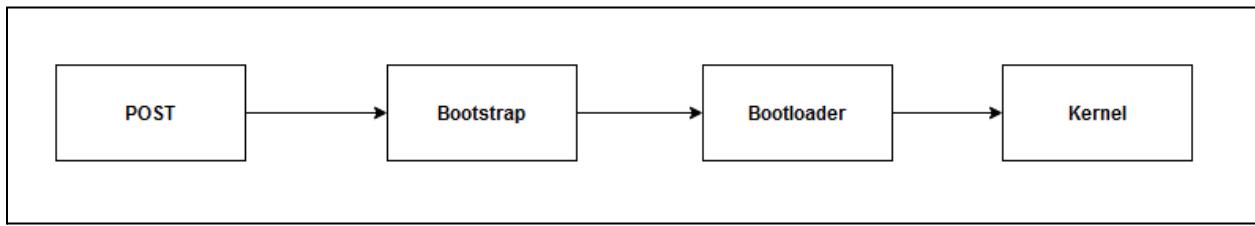
Berikut adalah [link dokumentasi setup VSCode debugger IF2230](#).

Selanjutnya lakukan *login akun GitHub* dan akses *GitHub Classroom* menggunakan tautan yang diberikan oleh asisten. Pada *GitHub Classroom* akan terdapat *kit template* yang dapat digunakan untuk membuat dasar repository tugas besar ini.

Repository template yang diberikan akan memiliki struktur seperti berikut

```
root_template/
├ .vscode/
│   └ settings.json
│   └ launch.json
│   └ tasks.json
└ src/
    └ lib-header/
        └ framebuffer.h
        └ gdt.h
        └ kernel_loader.h
        └ portio.h
        └ stdmem.h
        └ stdtype.h
    └ kernel.c
    └ stdmem.c
    └ portio.c
    └ kernel_loader.s
└ bin/
    └ .gitignore
└ other/
    └ grub1
    └ makefile
```

### 3.2. Booting Sequence & GRUB



Saat komputer pertama kali menyala, komputer akan menjalankan [Power-on self-test \(POST\)](#) dan jika berhasil BIOS akan melanjutkan untuk mengeksekusi program **bootstrap** yang terdapat di **Master Boot Record (MBR)**. Singkatnya MBR merupakan sektor pertama (yang biasanya berukuran 512 bytes) pada *non-volatile partitioned storage* (seperti HDD, *removable drive*, etc).

Bootstrap yang berukuran sangat kecil ini hanya bertugas untuk membaca **bootloader** dari *non-volatile memory* ke *volatile memory* (atau dibaca ke RAM) dan melakukan *jump* / mengeksekusi kode **bootloader** pada RAM. Bootloader bertugas untuk melanjutkan proses **booting** ke sistem operasi. Bootloader memiliki ukuran yang relatif lebih besar dibandingkan **bootstrap** yang hanya berukuran 1 sektor sehingga dapat menyediakan beberapa fungsionalitas sederhana seperti selector untuk OS dan *command line*.

Tugas ini akan menggunakan GRUB sebagai bootloader, tepatnya GRUB Legacy version 0.95 (eltorito). Bootloader GRUB sudah disediakan dalam folder *other* dengan nama file *grub1*. File tersebut akan digunakan untuk pembuatan *disk image*.

Buatlah sebuah file bernama **menu.lst**. File ini digunakan sebagai daftar sistem operasi yang dapat dijalankan oleh GRUB. Berikut adalah isi untuk *menu.lst*

```
default 0
timeout 0

title os
kernel /boot/kernel
```

Contoh *menu.lst* tersebut akan menambahkan OS yang dibuat dengan title “os” dan binary kernel yang berada pada */boot/kernel*. Default OS yang digunakan adalah entry ke-0 (Hanya ada 1 entry sehingga akan memilih “os” yang didefinisikan sebelumnya) dan tidak ada timeout sehingga GRUB secara otomatis akan langsung masuk ke OS tanpa menunggu *menu selection*.

Dokumentasi lebih lengkap dapat dilihat pada [GNU GRUB Manual 0.97](#).

### 3.3. Kernel Sederhana

Template repository telah menyediakan kode-kode awal yang dapat digunakan sebagai dasar sistem operasi. Setelah GRUB memasukkan sistem operasi ke memori, GRUB akan menyerahkan operasi ke sistem operasi. Kode yang akan dieksekusi pertama terdapat pada source file assembly **kernel\_loader.s**, yaitu prosedur bernama **loader**. Prosedur loader akan menyiapkan **stack** dan memanggil fungsi C bernama **kernel\_setup**.

Kode assembly **kernel\_loader.s** yang disediakan template sudah memenuhi untuk bagian ini. Sehingga hanya diperlukan untuk membuat definisi fungsi **kernel\_setup** dengan bahasa C.

Buatlah file **kernel.c** (jika belum ada) yang berisikan kode berikut

#### kernel.c

```
#include "lib-header/portio.h"
#include "lib-header/stdtype.h"
#include "lib-header/stdmem.h"
#include "lib-header/gdt.h"
#include "lib-header/framebuffer.h"
#include "lib-header/kernel_loader.h"

void kernel_setup(void) {
    uint32_t a;
    uint32_t volatile b = 0x0000BABA;
    __asm__ ("mov $0xCAFE0000, %0" : "=r"(a));
    while (TRUE) b += 1;
}
```

Sebelum melakukan kompilasi dan linking, kode kernel wajib diletakkan pada memori **0x100000** keatas. Memori **0x0** hingga **0xFFFFF** digunakan untuk GRUB dan memory mapped I/O. Buatlah file baru **linker.ld** di folder **/src** yang berisikan lokasi & alignment berikut

#### linker.ld

```
ENTRY(loader)                  /* the name of the entry label */

SECTIONS {
    . = 0x00100000;           /* the code should be loaded at 1 MB */

    .multiboot ALIGN (0x1000) : /* align at 4 KB */
    {
        *(.multiboot)       /* GRUB multiboot header */
    }

    .text ALIGN (0x1000) : /* align at 4 KB */
    {
        *(.text)             /* all text sections from all files */
    }

    .rodata ALIGN (0x1000) : /* align at 4 KB */
```

```
{  
    *(.rodata*)           /* all read-only data sections from all files */  
}  
  
.data ALIGN (0x1000) : /* align at 4 KB */  
{  
    *(.data)             /* all data sections from all files */  
}  
  
.bss ALIGN (0x1000) : /* align at 4 KB */  
{  
    *(COMMON)            /* all COMMON sections from all files */  
    *(.bss)               /* all bss sections from all files */  
}  
}
```

Sekarang sistem operasi dapat dikompilasi dan link menjadi satu executable ELF32 dengan perintah berikut pada **root repository**

```
gcc -ffreestanding -fshort-wchar -g -nostdlib -nostdinc -fno-builtin  
-fno-stack-protector -nostartfiles -nodefaultlibs -Wall -Wextra -Werror -m32 -c  
-Isrc src/kernel.c -o bin/kernel.o
```

```
nasm -f elf32 -g -F dwarf src/kernel_loader.s -o bin/kernel_loader.o
```

```
ld -T src/linker.ld -melf_i386 bin/kernel.o bin/kernel_loader.o -o bin/kernel
```

Hasil kompilasi akan menghasilkan executable ELF32 bernama **kernel** pada folder bin

### 3.4. Pembuatan *disk image*

Untuk menjalankan sistem operasi, kernel yang dibuat perlu masukkan kedalam *disk image*. Dengan menggunakan tool **genisoimage**, *disk image* iso dapat dibuat dengan menyusun folder yang akan digunakan sebagai struktur folder iso.

Buatlah sebuah folder **iso** pada folder **/bin/** dengan struktur seperti berikut

```
iso/
└ boot/
    └ grub/
        └ menu.lst
    └ grub1
        ; binary GRUB yang diberikan pada /other/
    └ kernel
        ; kernel executable dengan format ELF32
```

Berdasarkan direktori tersebut, gunakan perintah berikut untuk membuat *disk image iso* dari sistem operasi (asumsi **current working directory** adalah **parent** dari **iso** yaitu berada pada folder **/bin/**)

```
genisoimage -R \
    -b boot/grub/grub1 \
    -no-emul-boot \
    -boot-load-size 4 \
    -A os \
    -input-charset utf8 \
    -quiet \
    -boot-info-table \
    -o OS2023.iso \
    iso
```

Setelah perintah dijalankan, akan dibentuk sebuah image dengan format .iso bernama **OS2023.iso** pada folder **/bin/**

### 3.5. Build Script

Untuk mempermudah proses *development*, diperlukan script untuk melakukan otomatisasi serangkaian proses *build* sistem operasi. Jika menggunakan konfigurasi template VSCode yang diberikan, template tersebut akan menjalankan `make build` sebelum menghubungkan debugger ke QEMU ketika **F5 (Start debugging)** ditekan.

Berikut adalah potongan makefile yang disediakan template

```
kernel:
    @$(ASM) $(AFLAGS) src/kernel_loader.s -o bin/kernel_loader.o
# TODO: Compile C file with CFLAGS
    @$(LIN) $(LFLAGS) bin/*.o -o $(OUTPUT_FOLDER)/kernel
    @echo Linking object files and generate elf32...
    @rm -f *.o

iso: kernel
    @mkdir -p $(OUTPUT_FOLDER)/iso/boot/grub
    @cp $(OUTPUT_FOLDER)/kernel      $(OUTPUT_FOLDER)/iso/boot/
    @cp other/grub1                $(OUTPUT_FOLDER)/iso/boot/grub/
    @cp $(SOURCE_FOLDER)/menu.lst   $(OUTPUT_FOLDER)/iso/boot/grub/
# TODO: Create ISO image
    @rm -r $(OUTPUT_FOLDER)/iso/
```

Lengkapilah kedua **TODO** pada makefile hingga sistem operasi dapat di-build dengan `make build`. Semua flags untuk *compiler*, *assembler*, dan *linker* telah didefinisikan dan disediakan pada template. Gunakan perintah seperti berikut untuk melakukan kompilasi

```
$(CC) $(CFLAGS) kernel.c -o kernel.o
```

Jika mengalami error karena lokasi / nama file yang berbeda, ubah makefile sesuai dengan nama dan lokasi yang sesuai.

### 3.6. Menjalankan Sistem Operasi

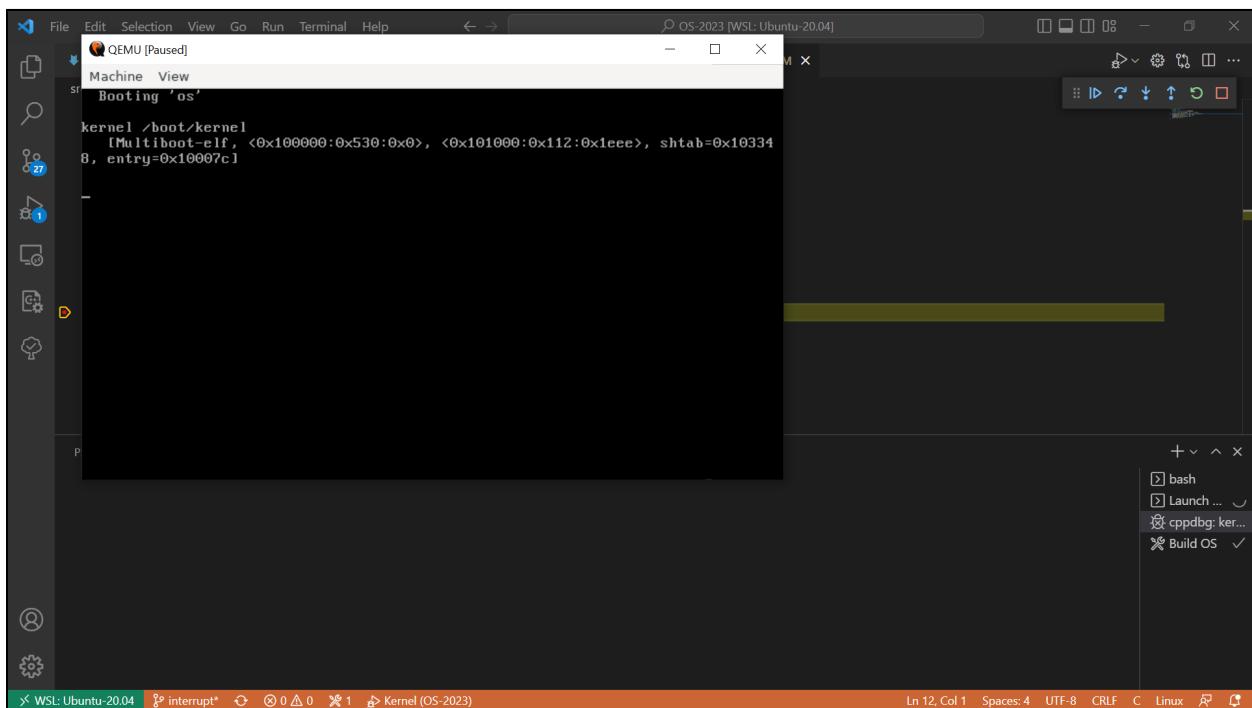
Image .iso yang telah dibuat pada bagian sebelumnya dapat dijalankan menggunakan QEMU dengan perintah berikut pada folder /bin/

```
qemu-system-i386 -s -cdrom OS2023.iso
```

Flag **-s** digunakan untuk membuka gdb server pada localhost:1234, sedangkan **-S** menghentikan eksekusi QEMU hingga debugger telah ter-attach dengan gdb server. Jika ingin langsung masuk tanpa menunggu, hilangkan flag **-S** seperti perintah diatas

Jika menggunakan Visual Studio Code dan debugger sudah terkonfigurasi dengan baik ([link dokumentasi setup VSCode debugger IF2230](#)), dapat digunakan hotkey **F5** untuk menjalankan debugger dan **Shift + F5** untuk menutup debugger.

Jika berjalan dengan baik, QEMU akan menampilkan tampilan kurang lebih seperti berikut



Eksekusi QEMU dengan VSCode Debugger

**Pastikan telah menjalankan seluruh bagian hingga ini dengan baik.** Pastikan bahwa *development environment* telah berjalan dengan baik sebelum melanjutkan untuk melakukan *development*. Berikut adalah *checklist* sebelum melanjutkan ke bagian selanjutnya

- Text editor
- Compilation & build system
- Running OS
- (Opsiional) Debugger

### 3.7. Kernel Development & Simple Text Framebuffer

Selamat! *Development environment* untuk pembuatan sistem operasi sudah siap digunakan. Sebelum melanjutkan development, berikut adalah beberapa poin penting yang perlu diperhatikan pada *operating system development*

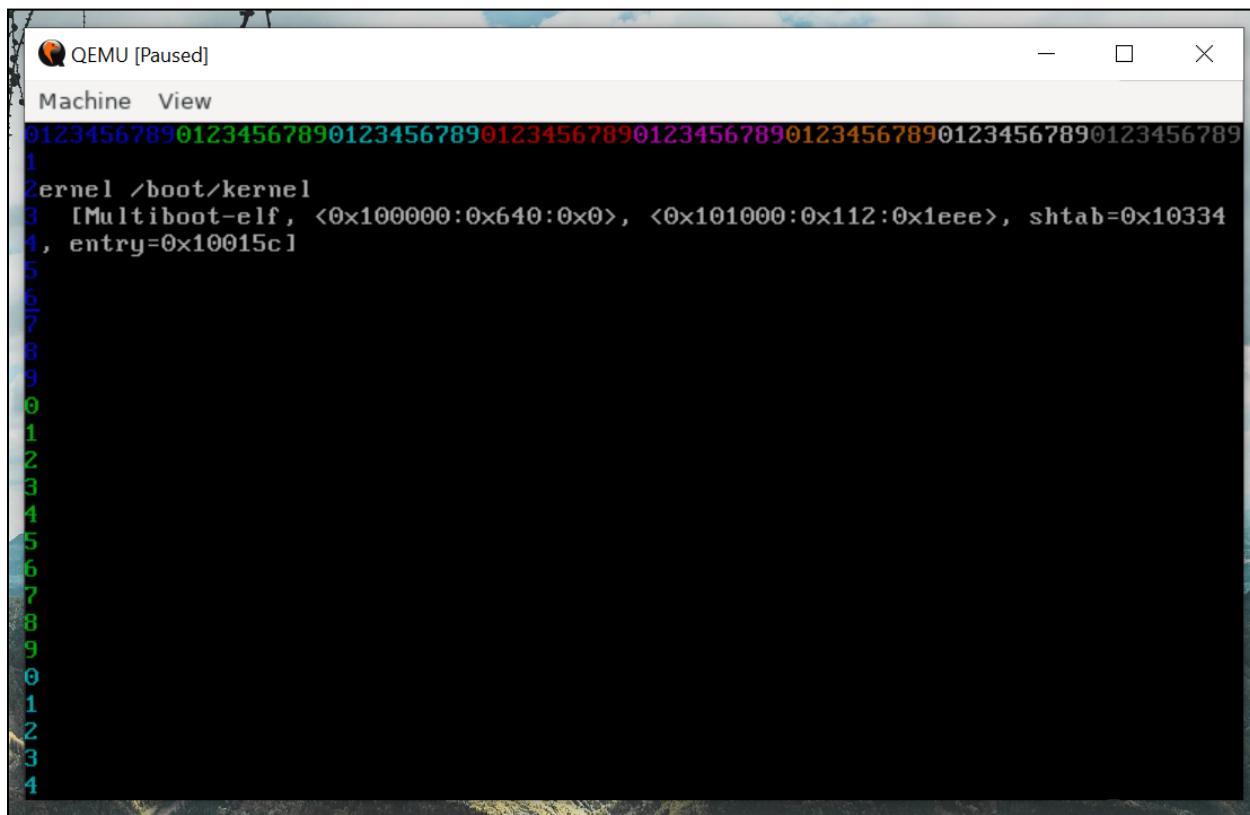
1. Tugas ini adalah *kernel development*, dimana keamanan *memory* **tidak ada** dan **handholding** yang **minimum**
2. **Kernel memiliki akses penuh sistem.** Seluruh memory, register, dan kendali CPU dimiliki oleh kernel. Konsekuensinya, kernel dapat memanipulasi memory tanpa exception sistem operasi seperti segfault

“Remember, with great power comes great responsibility” - Uncle Ben
3. Meskipun tidak ada exception pada tingkat sistem operasi, **masih terdapat exception pada tingkat CPU**. Detail setiap instruksi x86 dan exceptionnya dapat dicek pada [instruction set reference](#) dan Interrupt & Exception
4. Kesalahan dalam penggunaan *array* dan *pointer* pada tingkat kernel akan menyebabkan **memory corruption**
5. *Memory corruption* yang terakumulasi dapat menyebabkan masalah seperti *jump* yang salah, pembacaan memori pada lokasi yang salah, hingga kernel yang tidak dapat berjalan sesuai dengan ekspektasi. **Patut dicatat bahwa dampak memory corruption mungkin tidak langsung terlihat**
6. Dengan berdasarkan poin-poin sebelumnya, sangat direkomendasikan untuk tidak menghilangkan flag **-Wall -Wextra -Werror** dari gcc. Flag -Werror akan membuat semua warning menjadi error dan menghentikan kompilasi. **Gunakan error tersebut untuk berhenti sejenak dan mempelajari lebih lanjut kode.** Error akan menguji pemahaman tentang standar bahasa C dan sistem, bukan sebuah *warning/error message* yang tidak berguna dan di-*ignore*
7. Arsitektur target sistem operasi merupakan **Intel x86**. Dokumentasi penuh CPU arsitektur x86 disediakan langsung dari Intel dapat dicek pada bagian [referensi](#). **Download dan gunakan dokumentasi tersebut sebagai sumber utama.** Tugas ini akan mereferensi beberapa bagian langsung dari dokumentasi tersebut
8. **Sangat dianjurkan untuk menggunakan debugger**, jika tidak menggunakan VSCode, server gdb QEMU dapat dihubungkan dengan *gdb* atau *debugger* lain

Setelah membaca ~~terms and condition~~ poin-poin diatas dan memahami kontrol yang dimiliki kernel, ***let's dive in into kernel development!***

Tugas pertama merupakan pembuatan driver untuk *Text Framebuffer*.

Text Framebuffer merupakan device output sederhana yang menampung karakter dan ber-resolusi 80x25 karakter



Untuk berkomunikasi dengan device, umumnya terdapat dua metode yaitu **memory-mapped I/O** dan **port I/O**. Text Framebuffer menggunakan memory mapped I/O untuk menampung karakter pada layar dan menggunakan port I/O untuk mengontrol kursor.

Implementasikan tiga fungsi yang dideklarasikan berikut

```
framebuffer.h
```

```
void framebuffer_write(uint8_t row, uint8_t col, char c, uint8_t fg, uint8_t bg);
void framebuffer_set_cursor(uint8_t r, uint8_t c);
void framebuffer_clear(void);
```

File *framebuffer.h* telah disediakan pada template repository untuk memudahkan. Semua deklarasi fungsi disertai juga deskripsi fungsi dan parameter menggunakan **doxygen** yang dapat di-hover pada IDE / Editor tertentu

```

src > C kernel.c > kernel_setup(void)
6  #inc void framebuffer_write(uint8_t row, uint8_t col, char c, uint8_t fg, uint8_t bg)
7
8  void Set framebuffer character and color with corresponding parameter values. More details:
9    https://en.wikipedia.org/wiki/BIOS_color_attributes
10
11  Parameters:
12    row - Vertical location (index start 0)
13    col - Horizontal location (index start 0)
14    c - Character
15    fg - Foreground / Character color
16    bg - Background color
17    framebuffer_write(3, 11, '! ', 0, 0xF);
18    framebuffer_set_cursor(0, 0);
19    while (1) b += 1;
20 }

```

Untuk `framebuffer_write()` digunakan memory-mapped I/O yang terletak pada macro `MEMORY_FRAMEBUFFER` (`0xB8000`). Detail susunan memory text framebuffer dapat dicek pada [TextUI - OSDev](#). `memset()` yang disediakan pada `stdmem.c` dapat digunakan untuk memanipulasi memori.

Untuk `framebuffer_set_cursor()` digunakan port I/O, instruksi assembly `in` dan `out` dapat menggunakan fungsi yang telah disediakan pada `portio.c`. Detail penggunaan port I/O untuk kursor dapat dicek pada [Text Mode Cursor - OSDev](#).

Untuk `framebuffer_clear()` membersihkan text framebuffer, implementasi dibebaskan.

Semua fungsi pada bagian ini dapat diimplementasikan hanya menggunakan bahasa C. Namun jika ingin menggunakan assembly diperbolehkan.

Berikut adalah template `framebuffer.c` yang dapat digunakan sebagai referensi

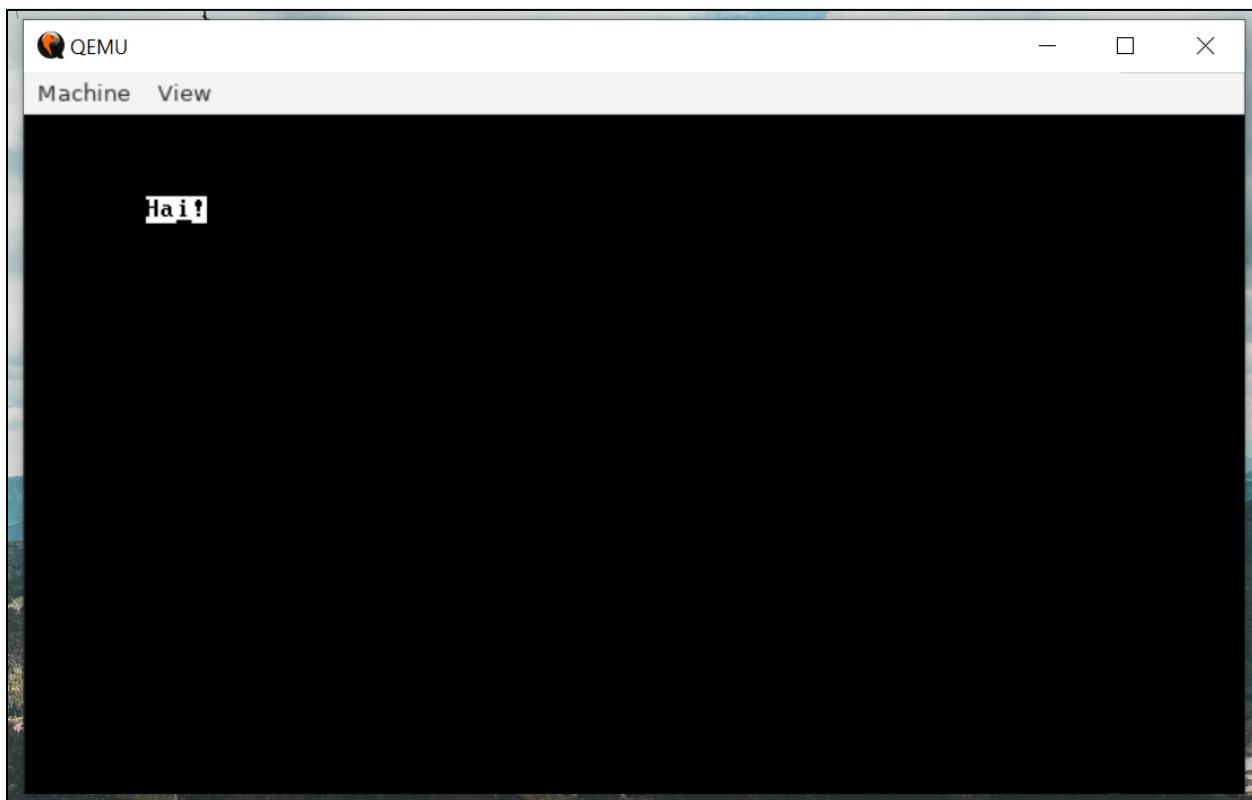
framebuffer.c
<pre> #include "lib-header/framebuffer.h" #include "lib-header/stdtype.h" #include "lib-header/stdmem.h" #include "lib-header/portio.h"  void framebuffer_set_cursor(uint8_t r, uint8_t c) {     // TODO : Implement }  void framebuffer_write(uint8_t row, uint8_t col, char c, uint8_t fg, uint8_t bg) {     // TODO : Implement }  void framebuffer_clear(void) {     // TODO : Implement } </pre>

Edit kode kernel berikut pada kernel.c sebagai pengujian

### kernel.c

```
void kernel_setup(void) {
    framebuffer_clear();
    framebuffer_write(3, 8, 'H', 0, 0xF);
    framebuffer_write(3, 9, 'a', 0, 0xF);
    framebuffer_write(3, 10, 'i', 0, 0xF);
    framebuffer_write(3, 11, '!', 0, 0xF);
    framebuffer_set_cursor(3, 10);
    while (TRUE);
}
```

Lakukan build sistem operasi (gunakan **F5 & Shift + F5** pada VSCode) dan jika berhasil akan menampilkan seperti berikut



### 3.8. Pembuatan Global Descriptor Table (GDT)

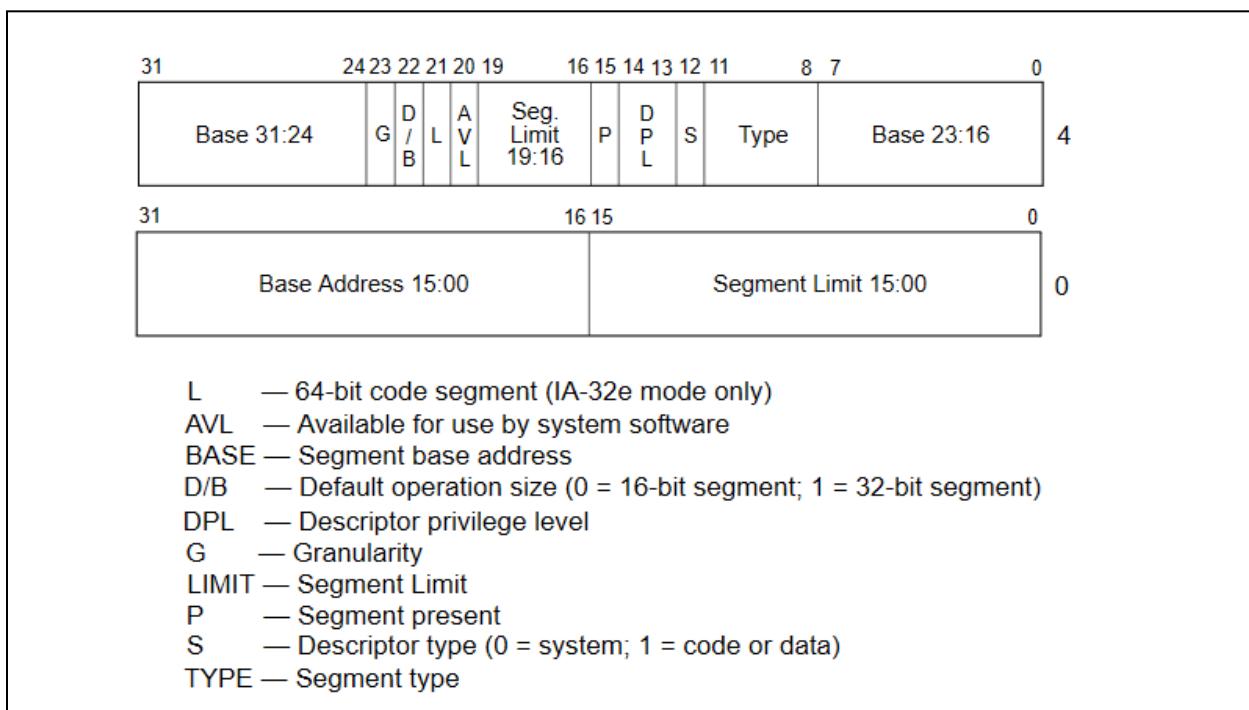
Selanjutnya adalah persiapan untuk memasuki *protected mode* 32-bit. Protected Mode x86 mewajibkan untuk **Global Descriptor Table (GDT)** telah terbaca dengan baik oleh CPU sebelum memasuki mode tersebut.

Dalam lingkungan 32 bit, proteksi terhadap memori dilakukan dengan menggunakan *segmentation* yang dideskripsikan / disimpan dalam bentuk struktur data **Segment Descriptor**. *Segment Descriptor* memiliki ukuran 8 bytes.

Kumpulan semua *segment descriptor* sistem akan disimpan pada struktur data yang bernama **Global Descriptor Table (GDT)**.

Setiap GDT wajib memiliki 1 entri GDT kosong (*null descriptor*) sebagai entri pertama dan 2 entri GDT untuk dipakai kernel. Kernel membutuhkan 1 entri untuk segmen memori berisikan kode dan 1 entri lain untuk segmen memori berisikan data (seperti variabel dan lain-lain).

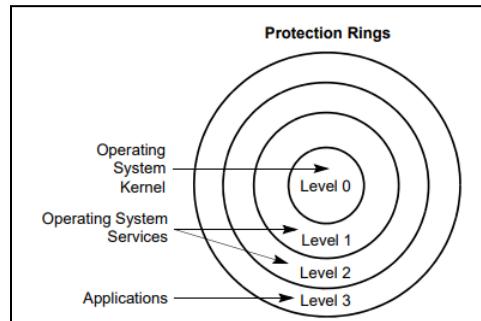
Berikut adalah definisi struktur data *Segment Descriptor* menurut Intel x86 Manual 3A



[Intel x86 Manual 3A](#) - Figure 3-8 Segment Descriptor

Definisi & penjelasan setiap flags dan atribut lain struktur data Segment Descriptor dapat dicek langsung pada Intel Manual pada bagian 3.4.5 Segment Descriptors.

Dalam tugas ini, sistem operasi yang akan dibuat menggunakan model arsitektur sistem operasi dengan dua *protection rings*. Dengan *ring* pertama adalah level 0 untuk kernel (DPL = 0 atau PL0), dan *ring* kedua adalah level 3 untuk *user mode* (DPL = 3 atau PL3).



Berikut adalah tabel berisikan GDT sederhana yang memuat informasi segmen kernel

Entri	Nama	Limit	Base	Type	S	DPL	P	L	D/B	G
0	Null	0	0	0	0	0	0	0	0	0
1	Kernel code segment	0xFFFFF	0	- Code - Not Accessed - Readable - Not Conforming  (0xA/0b1010)	1 (Code or Data Segment )	0 (PL0 / Kernel )	1 (Valid Segment )	0	1 (32-bit operand )	1 (4KB )
2	Kernel data segment	0xFFFFF	0	- Data - Not Accessed - Writable - Direction Up  (0x2/0b0010)	1	0	1	0	1	1

- Cell yang berwarna putih menandai data tersebut disimpan pada GDT
- Cell yang berwarna kuning muda hanya sebagai pembantu / tidak disimpan dimemori

Tugas dari bagian ini adalah membuat GDT yang telah inisiasi datanya sesuai dengan tabel diatas. Template repository telah menyediakan **gdt.h** sebagai panduan dasar penggerjaan bagian ini.

Lengkapilah struct yang telah dideklarasikan pada **gdt.h** dan definisikan GDT beserta GDTR \_gdt\_gdtr pada **gdt.c**. Deklarasi **uint8\_t abc : 5;** memiliki arti variabel abc berukuran tepat 5-bit. Syntax tersebut bernama **bit field**. Berikut adalah contoh gdt.c yang dapat digunakan

### gdt.c

```
#include "lib-header/stdtype.h"
#include "lib-header/gdt.h"

/***
 * global_descriptor_table, predefined GDT.
 * Initial SegmentDescriptor already set properly according to GDT definition in
Intel Manual & OSDev.
 * Table entry : [{Null Descriptor}, {Kernel Code}, {Kernel Data (variable, etc)},
...].
*/
struct GlobalDescriptorTable global_descriptor_table = {
    .table = {
        {
            // TODO : Implement
        },
        {
            // TODO : Implement
        },
        {
            // TODO : Implement
        }
    }
};

/***
 * _gdt_gdtr, predefined system GDTR.
 * GDT pointed by this variable is already set to point global_descriptor_table
above.
 * From: https://wiki.osdev.org/Global\_Descriptor\_Table, GDTR.size is GDT size minus
1.
*/
struct GDTR _gdt_gdtr = {
    // TODO : Implement, this GDTR will point to global_descriptor_table.
    //           Use sizeof operator
};
```

**Catatan penting : Perhatikan urutan definisi struct, pastikan urutan dan alignment tepat 1:1 dengan Intel x86 manual. Kesalahan alignment menyebabkan kegagalan pembacaan pada instruksi lgdt.**

Berikut referensi yang dapat digunakan untuk GDT

- [Segment Descriptor - OSDev](#)
- [Global Descriptor Table - OSDev](#)
- [GDT Tutorial - OSDev](#)
- **Intel x86 Manual 3A - 3.4.3 Segment Registers - 3.5.1 Segment Descriptor Tables**

Untuk mengetes bagian ini telah berjalan dengan baik atau belum, lanjutkan bagian selanjutnya yaitu memasuki **protected mode**.

### 3.9. Memasuki Protected Mode

Setelah GDT telah dibuat, dapat dilanjutkan untuk memasuki x86 *protected mode*. Instruksi pada [Intel x86 Manual 3A - 9.9.1 Switching to Protected mode](#) menjelaskan secara detail *step-by-step* yang perlu dilakukan untuk memasuki *protected mode*, berikut adalah secara singkatnya

1. Melakukan *load* GDT dengan instruksi `lgdt`
2. Set bit ke-0 (Protected mode flag) CR0 (Control Register 0) menggunakan **bitwise or** dengan nilai 1
3. Lakukan *far jump* untuk memasuki *protected mode*
4. Menyesuaikan semua data segment register (ss, ds, es)

Sebagian besar kode assembly `enter_protected_mode` telah disiapkan pada `kernel_loader.s`. Lengkapilah kode assembly tersebut sesuai step yang dideskripsikan diatas.

Berikut referensi yang dapat digunakan

- [Protected Mode - OSDev](#)
- Intel x86 Manual 3A - 9.9.1 Switching to Protected mode

Untuk menguji apakah GDT, pembacaan GDT, dan telah masuk protected mode, tambahkan kode berikut pada `kernel.c`

`kernel.c`

```
void kernel_setup(void) {
    enter_protected_mode(&_gdt_gdtr);
    framebuffer_clear();
    framebuffer_write(3, 8, 'H', 0, 0xF);
    framebuffer_write(3, 9, 'a', 0, 0xF);
    framebuffer_write(3, 10, 'i', 0, 0xF);
    framebuffer_write(3, 11, '!', 0, 0xF);
    framebuffer_set_cursor(3, 10);
    while (TRUE);
}
```

Jika sistem operasi berhasil menampilkan tulisan `Hai!` seperti pada [bagian 3.6 Kernel Development & Simple Text Framebuffer](#), maka sistem operasi telah berhasil membaca GDT dan masuk ke protected mode.

Jika terdapat kesalahan pada definisi atau *load* GDT, instruksi *far jump* dan operasi *mov* segment register akan dapat menyebabkan special CPU exception (**triple fault**) dan melakukan reboot. Hal ini akan menyebabkan **bootloop**.

Usahakan untuk menyelesaikan milestone 1 dengan baik, jika telah mencoba dan masih *stuck* hingga akhir mendekati deadline, tanyakan kepada teman lain atau gunakan asistensi.

## IV. Penilaian

- **Setup (30)**
  - ISO untuk OS berhasil dibuat dengan benar (10)
  - OS dapat dijalankan dengan GRUB (10)
  - Script makefile telah dibuat dan bisa dijalankan (10)
- **Framebuffer (30)**
  - Dapat menulis karakter pada framebuffer (5)
  - Isi layar dapat dihapus (10)
  - Kursor pada layar dapat dipindahkan (15)
- **GDT dan Protected Mode (40)**
  - Definisi GDT tepat sesuai dengan Intel Manual (15)
  - GDT berhasil dibaca & segment register berhasil diubah (10)
  - Sistem operasi dapat memasuki protected mode dengan baik (15)
- **Bonus**



*"sabar cuy, spek wajibnya aja dah bejibun"*  
- tomeat Tom si Kucing

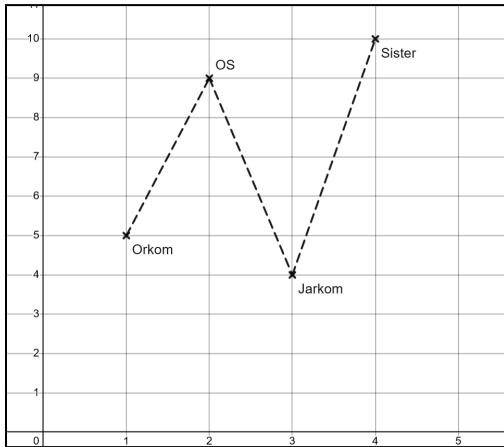
## V. Pengumpulan dan Deliverables

1. Untuk tugas ini Anda diwajibkan menggunakan *version control system git* dengan menggunakan sebuah *repository private* di Github Classroom “**Lab Sister 21**” (gunakan email *std* agar gratis). Berikut adalah [link assignment GitHub Classroom](#).
2. Kit untuk semua milestone tugas besar IF2230 tersedia pada repository GitHub berikut [link repository](#).
3. Kreativitas dalam pengerjaan sangat dianjurkan untuk memperdalam pemahaman. Penilaian sepenuhnya didasarkan dari kriteria penilaian.
4. Setiap kelompok diwajibkan untuk membuat tim dalam Github Classroom dengan **nama yang sama pada spreadsheet kelompok**. Mohon diperhatikan lagi cara penamaan kelompok agar bisa sama dengan nama repository Anda nanti.\*
5. Meskipun commit tidak dinilai, lakukanlah *commit* yang wajar dan sesuai *best practice* (tidak semua kode satu *commit*).
6. File yang harus terdapat pada *repository* adalah file-file *source code* dan *script* (jika ada) sedemikian rupa sehingga jika diunduh dari github dapat dijalankan. Dihimbau untuk tidak memasukkan *binary* dan *temporary files* hasil kompilasi ke *repository* (manfaatkan *.gitignore*).
7. Isikan nama kelompok dan anggotanya pada [link berikut](#), paling lambat tanggal **13 Februari 2023 22.30 WIB**.
8. **Mulai** Jumat, 10 Februari 2023, 18.00 WIB  
**Deadline** Kamis, 2 Maret 2023, 23.59 WIB  
Perubahan kode di luar waktu nilai dan *deadline* akan dikenakan pengurangan nilai.
9. Pengumpulan dilakukan dengan membuat *release* dengan tag **v1.0** pada repository yang telah kelompok Anda buat sebelum deadline. Pastikan tag sesuai format. **Repository team yang tidak memiliki tag ini akan dianggap tidak mengumpulkan Milestone 1.**
10. Kami akan **menindaklanjuti segala bentuk kecurangan** yang terstruktur, masif, dan / atau sistematis.
11. Diharapkan untuk mencoba mengerjakan tugas besar ini terlebih dahulu sebelum mencari sumber inspirasi dari *google*, *repository*, atau teman yang sudah selesai. Namun **dilarang melakukan copy-paste langsung** kode orang lain (selain spesifikasi). Copy-paste kode secara langsung akan dianggap melakukan kecurangan.
12. Tim asisten mungkin mengecek langsung kode yang dibuat dan memberikan feedback melalui commit comment pada GitHub.
13. Dilarang melakukan kecurangan lain yang merugikan peserta mata kuliah IF2230.

14. Jika ada pertanyaan atau masalah penggerjaan harap segera menggunakan sheets QnA mata kuliah IF2230 Sistem Operasi pada [link berikut](#).
15. Terdapat tutorial yang akan dilakukan pada pertengahan milestone. Tutorial tidak bersifat wajib dan dilakukan untuk materi tambahan yang diperlukan dalam tugas besar, memperjelas pekerjaan tugas besar serta forum tanya jawab. Informasi lebih lengkap mengenai tutorial akan disampaikan melalui milis.
16. Selain tutorial, terdapat asistensi opsional bagi kelompok yang membutuhkan bantuan lebih lanjut. Jika membutuhkan asistensi, gunakan form request asistensi IF2230 Sistem Operasi pada [link berikut](#). **Asisten akan mengontak maksimal paling lambat H+1 setelah request**, jika tidak ada kontak hingga batas waktu tercapai, asumsikan tidak ada asisten yang sedang tersedia. Gunakan form yang sama untuk meminta request asistensi lagi, jika masih dibutuhkan.
17. Sekali lagi, **sangat dianjurkan untuk memakai sistem operasi Ubuntu 20.04**

## VI. Tips Pengerojaan

1. Sebagai pembuka, berikut adalah **expected difficulty rating** yang biasanya ada pada **Orkom, Sistem Operasi, Jaringan Komputer, dan Sistem Paralel dan Terdistribusi**



2. Tugas besar ini nantinya akan menggunakan beberapa mata kuliah yang semestinya telah diambil : **IF2110 - Algoritma & Struktur Data, IF2130 - Organisasi dan Arsitektur Komputer**, dan yang diambil bersamaan **IF2211 - Strategi Algoritma**.
3. Tugas ini **SANGAT** memerlukan eksplorasi mendalam sehingga pastikan tenaga baik fisik maupun mental kalian terjaga
4. Dengan *forewarning* diatas, sangat disarankan untuk **tidak mengerjakan tugas ini dekat dengan deadline** (**You have been warned deadlier!**).  
**Tugas besar ini tidak didesain untuk dikerjakan dalam satu hari, expect nilai 0 jika merencanakan mengerjakan H-1**
5. **Kerjakan tugas dengan kelompok**, tugas juga didesain sebagai tugas kelompok, **bukan tugas individu**. Jika masih mengalami kendala, gunakan QnA dan form asistensi yang disediakan oleh asisten
6. Terkait dengan kelompok, pada akhir pengerojaan tugas besar akan terdapat peer assessment. Asisten Lab Sister menggunakan sistem weighting poin peer assessment dengan fungsi tertentu. Dari poin peer assessment & penilaian asisten, **anggota yang tidak berkontribusi sama sekali mungkin mendapatkan nilai 0**.
7. Jika mengalami permasalahan, gunakan debugger untuk melakukan step-by-step instruction, examine memory, dan hal-hal lainnya. Tips penggunaan debugger terdapat pada [IF2230 - Tips Debugger](#)
8. Jika ingin menambahkan struktur data sendiri, gunakan `__attribute__((packed))` pada definisi struktur data. Macro tersebut meminta gcc untuk tidak melakukan alignment

## VII. Referensi

1. Kitab Intel x86 dan x64 Volume 3A - Intel® 64 and IA-32 Architectures Developer's Manual Volume 3A System Programming Guide

<https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-vol-3a-part-1-manual.html.html>

Catatan: Tugas akan menggunakan **Volume 3A - System Programming Guide Part 1**

Volume lain (1, 2ABCD, 3ABCD, 4) & edisi lengkap (5060 pages) dapat diakses pada link berikut: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>

2. Referensi utama - **Catatan penting : Beberapa detail yang ditulis tidak akurat**

<https://littleosbook.github.io>

3. c9x - x86 Instruction Set References

<https://c9x.me/x86/>

4. OSDev - x86 kernel from scratch

[https://wiki.osdev.org/Meaty\\_Skeleton](https://wiki.osdev.org/Meaty_Skeleton)

5. OSDev - Interrupt

<https://wiki.osdev.org/Interrupt>

6. OSDev - Exception

<https://wiki.osdev.org/Exceptions>



~gws~

“Sori bg latar belakangnya pendek orang yang nulis lagi wangy wangy zeta”  
~Dzaky~

“Bocchi best anime girl character”  
~Amar~

“Fuwa fuwa pure pure”  
~Rozan~

“mumumumu.. muri zetai!!”  
~Erik~

“kit-auraaaaaaaaaa”  
~Gare~

“Seperti kata Yamada  
[https://www.youtube.com/watch?v=jpAor9xaeVY&ab\\_channel=Towa%27sApostle%2CNeo%5B1stApostleoftheTwelve%5D](https://www.youtube.com/watch?v=jpAor9xaeVY&ab_channel=Towa%27sApostle%2CNeo%5B1stApostleoftheTwelve%5D)”  
~Andreas~

“Jelas zeta Istri gw. Orang yang diatas gw suka halu ngaku-ngaku suaminya Zeta”  
~Stanley~

“MK (momen ketika) tugas-tugas di semester 4 rilis berturut-turut  
Tetapi...  
kamu adalah seorang informatikawan sejati”  
~Fawwaz~

“Gws sister ‘20, semoga sister ‘21 ada yang meneruskan :)”  
Lock1 / Brush