Lab1 实验报告

各种名词的含义及关系

CPU（Central Processing Unit，中央处理器）：执行程序指令，处理数据和控制其他硬件部件的操作。

内存（Memory）：内存是计算机用来存储程序和数据的临时存储空间。分为RAM（随机存储器）和ROM（只读存储器）。RAM 用于临时存储数据和程序，而 ROM 则用于存储启动引导程序和固化的系统信息。

BIOS（Basic Input/Output System，基本输入/输出系统）：BIOS 是计算机启动时的固化软件，它包含了计算机最基本的操作指令，用来初始化计算机硬件，并将控制权交给操作系统。

磁盘（Disk）：磁盘是计算机用来永久存储数据的设备，包括硬盘驱动器（HDD）和固态硬盘（SSD）。它用来存储操作系统、应用程序和用户数据。

主引导扇区（Master Boot Record，MBR）：主引导扇区是位于磁盘的第一个扇区，用来存储引导加载程序和磁盘分区表的信息。它是计算机启动时用来加载操作系统的重要组成部分。末尾有魔数0x55和0xaa。

加载程序（Bootloader）：加载程序是一段小型程序，负责引导计算机启动时加载操作系统。它通常存储在主引导扇区，并在计算机启动时被 BIOS 加载到内存中执行。

操作系统（Operating System，OS）：操作系统是管理计算机硬件和软件资源的系统软件。它负责管理计算机的文件系统、输入输出、内存管理、进程调度等任务，为用户和应用程序提供一个友好的界面和环境。

计算机加电后，BIOS 会读取磁盘的主引导扇区，加载到内存0x7c00的位置，然后执行其中的引导加载程序（Bootloader）。引导加载程序负责加载操作系统的核心部分，并将控制权交给操作系统。

1.1. 在实模式下实现一个Hello World程序

在实模式下通过陷入屏幕中断调用BIOS打印字符串

根据BIOS中断向量表，实模式下可以通过 int $0x10 中断进行屏幕上的字符串显示

输出hello world的部分可以参考index中的实现，使用displayStr

```
# TODO: This is lab1.1
/* Real Mode Hello World */

.code16

.global start
start:
    movw %cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    movw $0x7d00, %ax
    movw %ax, %sp # setting stack pointer to 0x7d00
```

```
    # TODO:通过中断输出Hello World
    pushw $13 # pushing the size to print into stack
    pushw $message # pushing the address of message into stack
    callw displayStr # calling the display function
loop:
    jmp loop

displayStr:
    pushw %bp
    movw 4(%esp), %ax
    movw %ax, %bp
    movw 6(%esp), %cx
    movw $0x1301, %ax
    movw $0x000c, %bx
    movw $0x0000, %dx
    int $0x10
    popw %bp
    ret

message:
    .string "Hello, World!\n\0"
```

1.2. 在保护模式下实现一个Hello World程序

实模式切换保护模式的过程中：

首先通过cli指令关闭中断，

然后打开A20数据总线，

加载 GDTR，

再设置 CR0 的PE位（第0位）为 1，

最后通过长跳转设置 CS 进入保护模式，

初始化 DS， ES， FS， GS， SS

```
# TODO: This is lab1.2
/* Protected Mode Hello World */

.code16

.global start
start:
    movw %cs, %ax
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    # TODO:关闭中断
    cli #clear interuption

    # 启动A20总线
    inb $0x92, %al
    orb $0x02, %al
```

```
    outb %al, $0x92

    # 加载GDTR
    data32 addr32 lgdt gdtDesc # loading gdtr, data32, addr32

    # TODO：设置CR0的PE位（第0位）为1
    movl %cr0, %eax
    orl $0x01,%eax
    movl %eax, %cr0

    # 长跳转切换至保护模式
    data32 ljmp $0x08, $start32 # reload code segment selector and ljmp to
start32, data32

.code32
start32:
    movw $0x10, %ax # setting data segment selector
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %fs
    movw %ax, %ss
    movw $0x18, %ax # setting graphics data segment selector
    movw %ax, %gs

    movl $0x8000, %eax # setting esp
    movl %eax, %esp
    # TODO:输出Hello World
    pushl $13
    pushl $message
    calll displayStr

loop32:
    jmp loop32

message:
    .string "Hello, World!\n\0"

displayStr:
    movl 4(%esp), %ebx
    movl 8(%esp), %ecx
    movl $((80*5+0)*2), %edi
    movb $0x0c, %ah
nextChar:
    movb (%ebx), %al
    movw %ax, %gs:(%edi)
    addl $2, %edi
    incl %ebx
    loopnz nextChar
    ret
```

根据GDT表项的结构，填出相应内容

其中段界限(limit)都为`0xfffff`

段基址(base)：代码段与数据段为`0x0`，视频段的基地址为`0xb8000`

类型(type)：代码段为`0xa`， 数据段为`0x2`

```
.p2align 2
gdt: # 8 bytes for each table entry, at least 1 entry
    # .word limit[15:0],base[15:0]
    # .byte base[23:16],(0x90|(type)),(0xc0|(limit[19:16])),base[31:24]
    # GDT第一个表项为空
    .word 0,0
    .byte 0,0,0,0

    # TODO: code segment entry
    .word 0xffff,0
    .byte 0,0x9a,0xcf,0

    # TODO: data segment entry
    .word 0xffff,0
    .byte 0,0x92,0xcf,0

    # TODO: graphics segment entry
    .word 0xffff,0x8000
    .byte 0x0b,0x92,0xcf,0

gdtDesc:
    .word (gdtDesc - gdt -1)
    .long gdt
```

1.3. 在保护模式下加载磁盘中的Hello World程序运行

bootMain函数通过elf指针，调用readSect接口读取磁盘中的内容，并将其加载到0x8c00处，然后跳转到加载的程序开始执行

```
void bootMain(void) {
    //TODO
    void (*elf)(void) = (void(*)(void))0x8c00;
    readSect((void*)elf, 1);  //loading sector 1 to 0x8c00
    elf();  //jumping to the program
}
```

其中可以看到在代码框架 app/Makefile 中设置的该Hello World程序入口地址为0x8c00

实验结果

成功在终端中打印Hello, world!