Lab1 实验报告

231880101 孙俊晖

邮箱：2033282932@qq.com

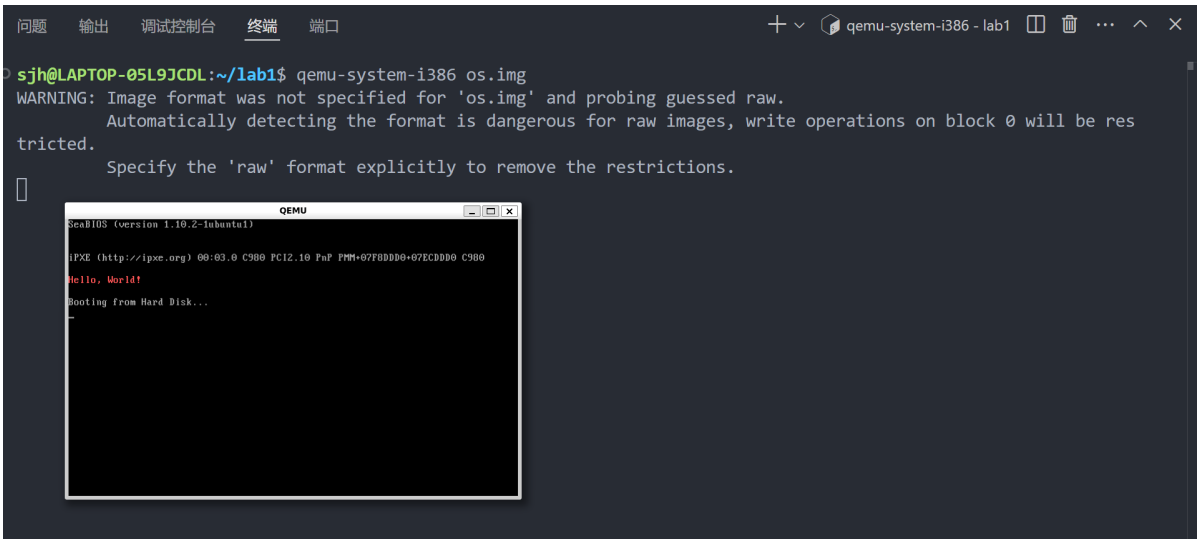
一、实验完成情况

1.实验进度

我完成了所有内容😜。

2.实验结果

我成功实现了从实模式切换至保护模式，在保护模式下读取磁盘1号扇区中的Hello World程序至内存中的相应位置，跳转执行该Hello World程序，并在终端中打印"Hello, World"



3.实验修改的代码位置

我修改了**bootloader**文件夹中的**boot.c**文件和**start.s**文件。

对于**boot.c**文件，我完成了其中**bootMain**函数的定义。

```
#define SECTSIZE 512
void bootMain(void) {
    char buffer[SECTSIZE];
    void (*helloFunc)() = (void (*)())0x8c00;
    readSect(buffer, 1);
    memcpy((void *)0x8c00, buffer, SECTSIZE);
    helloFunc();
}
```

而在start.s文件中，我首先完成了start里面的空缺部分，具体来说，包括**关闭中断、启动A20总线、加载GDTR、启动保护模式、设置CR0的PE位（第0位）为1、长跳转切换至保护模式**

```
start:
    movw %cs, %ax
```

```
    movw %ax, %ds
    movw %ax, %es
    movw %ax, %ss
    #以下是我添加的代码
    #关闭中断
    cli
    #启动A20总线
    inb $0x92, %al
    orb $0x02, %al
    outb %al, $0x92
    #加载GDTR
    lgdt gdtDesc
    #启动保护模式,设置CR0的PE位（第0位）为1
    movl %cr0, %eax
    orl $1, %eax
    movl %eax, %cr0
    #长跳转切换至保护模式
    ljmp $0x08, $start32
```

此外，我还完成了三个GDT表项的定义，包括**代码段描述符**、**数据段描述符**和**视频段描述符。**

```
gdt:
  #GDT第一个表项必须为空
  .word 0x0000
  .word 0x0000
  .byte 0x00
  .byte 0x00
  .byte 0x00
  .byte 0x00
  #代码段描述符
  .word 0xFFFF
  .word 0x0000
  .byte 0x00
  .byte 0x9A
  .byte 0xCF
  .byte 0x00
  #数据段描述符
  .word 0xFFFF
  .word 0x0000
  .byte 0x00
  .byte 0x92
  .byte 0xCF
  .byte 0x00
  #视频段描述符
  .word 0xFFFF
  .word 0x8000
  .byte 0x0B
  .byte 0x92
  .byte 0x40
  .byte 0x00
```

最后，我完成了gdtDesc表项的定义，这部分很简短，两行代码就能搞定。

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

二、遇到的问题

在实现保护模式下加载磁盘程序时，我在**boot.c**文件的**bootMain**函数中遇到了一个关键问题：由于**未调用memcpy**将磁盘读取的数据复制到目标内存地址0x8C00，导致程序加载后无法执行，屏幕始终无输出。通过QEMU的调试日志，我发现用户程序代码并未被正确加载到内存，进而意识到memcpy的缺失使得磁盘数据停留在临时缓冲区，未能映射到操作系统预期的入口地址。修正这一问题后，我不仅理解了memcpy在数据加载中的桥梁作用——它是连接磁盘I/O与内存执行的关键步骤，更认识到底层开发中对内存操作的精确性要求：即使是一个函数的遗漏，也可能导致整个启动链的中断。此外，这次调试让我对Bootloader的工作流程有了更直观的认识——从读取扇区到内存复制，再到跳转执行，每一步都需严格遵循硬件约定。这次经历不仅锻炼了我的问题定位能力，也让我意识到，系统级编程既是技术挑战，更是对耐心与细致的高度考验。

三、思考题解答

在计算机启动过程中，讲义2.1标题中提到的各种名词和它们之间的关系如下：

1. CPU（中央处理单元）

定义：负责执行指令和处理数据的核心硬件。

作用：加电后，CPU从固定地址（BIOS的ROM地址）开始执行第一条指令，完成初始化后逐步移交控制权给其他组件。

2. 内存

定义：存储数据和指令的硬件，分为RAM（易失性）和ROM（非易失性）。

分类：

基本内存（0~640KB）：供操作系统和应用程序使用。

上位内存（640KB~1MB）：分配给BIOS、显存等硬件设备的ROM。

扩展内存（1MB以上）：现代操作系统的主要内存区域。

关键地址：

0x7c00：BIOS将MBR加载到该地址，并跳转至此执行引导程序。

3. BIOS（基本输入输出系统）

定义：固化在主板ROM中的固件程序。

作用：

1. 加电后执行自检（POST）。
2. 扫描可启动设备，读取其主引导扇区（MBR）到内存的0x7c00。
3. 检查MBR末尾的魔数0x55AA，若合法则移交控制权给MBR中的引导程序。

4. 磁盘

定义：存储操作系统和程序的外部存储设备。

关键结构：

主引导扇区（MBR）：磁盘的首扇区（512字节），包含引导程序和分区表。

魔数0x55AA：位于MBR末尾，用于标识合法的可启动设备。

5. 主引导扇区（MBR）

定义：磁盘的第一个扇区（0号柱面、0号磁头、0号扇区）。

组成：

引导程序（446字节）：负责加载操作系统的核心代码。

分区表（64字节）：记录磁盘分区的信息。

魔数0x55AA（2字节）：标识合法MBR。

作用：被BIOS加载到0x7c00后，由其中的引导程序继续启动流程。

6. 加载程序（Bootloader）

定义：位于MBR中的小程序，负责加载操作系统内核到内存。

功能：

1. 从磁盘读取操作系统的代码和数据到内存。
2. 跳转到操作系统的入口地址，移交控制权。

7. 操作系统

定义：管理硬件资源和提供服务的核心软件。

启动流程：

1. BIOS加载MBR中的引导程序。
2. 引导程序加载操作系统内核到内存。
3. 操作系统初始化硬件和软件环境，进入用户交互模式。

8.关系

这些名词通过**硬件初始化、数据加载、控制权移交**三个核心环节紧密关联：

1. 硬件初始化由**CPU**和**BIOS**完成，确保基础环境。
2. 数据加载依赖**内存**和**磁盘**交互，**主引导扇区**与**加载程序**是关键枢纽。
3. 控制权移交从**BIOS**到**加载程序**，最终到**操作系统**，实现从固件到软件的完整启动流程。

四、实验心得

本次实验让我深刻理解了计算机从硬件初始化到软件加载的全流程。通过手动配置GDT表项和切换保护模式，我认识到分段机制如何通过基地址、限长和权限位实现内存保护与资源共享。在显存操作中，直接写入0xB8000地址并设置字符属性字节的经历，让我体会到硬件映射的直观性与底层编程的精确性——每一个小错误都可能导致屏幕显示异常甚至系统崩溃。调试过程中，QEMU的日志输出与分段测试

策略让我学会在缺乏可视化工具时如何逆向追踪问题，例如通过逐项验证代码段、数据段描述符的配置来定位保护模式切换失败的原因。此外，从实模式到保护模式的跳转逻辑让我意识到，计算机启动不仅是代码的执行，更是硬件状态与软件约定的精密配合。这次实验不仅巩固了操作系统启动流程的理论知识，更让我对“代码控制硬件”这一概念有了具象化的认知，为后续学习中断处理、进程调度等核心机制奠定了扎实的实践基础。