练习 4：分析 bootloader 加载 ELF 格式的 OS 的过程

通过阅读 bootmain.c，了解 bootloader 如何加载 ELF 文件。

通过分析源代码和通过 qemu 来运行并调试 bootloader&OS，bootloader 如何读取硬盘扇区的？ bootloader 是如何加载 ELF 格式的 OS？

4.1bootloader 如何读取硬盘扇区的？

对 bootmain.c 中与读取磁盘扇区相关的代码进行分析

4.1.1waitdisk 函数

```
36 /* waitdisk - wait for disk ready */
37 static void
38 waitdisk(void) {
39     while ((inb(0x1F7) & 0xC0) != 0x40)
40         /* do nothing */;
41 }
```

函数的作用是连续不断地从 0x1F7 地址读取磁盘的状态，直到磁盘不忙为止

4.1.2readsect 函数

```
static void
readsect(void *dst, uint32_t secno) {
    // wait for disk to be ready
    waitdisk();//等磁盘不忙为止

    outb(0x1F2, 1);                         // 往0x1F2写入要读的扇区，要读一个扇区，所以count = 1
    outb(0x1F3, secno & 0xFF);//输入LBA参数的0-7位
    outb(0x1F4, (secno >> 8) & 0xFF);//输入LBA参数的8-15位
    outb(0x1F5, (secno >> 16) & 0xFF);//输入LBA参数的16-23位
    outb(0x1F6, ((secno >> 24) & 0xF) | 0xE0);//输入LBA参数的24-27位，28位为0表示读取disk(0)，其余位(29-31)被强制为1
    outb(0x1F7, 0x20);                      // cmd 0x20 - read sectors(向磁盘发出读命令0x20)

    // 等待磁盘
    waitdisk();

    // 从0xF0读数据，读取到dst位，SIZE除以4是因为DW是以四个字节为一个单位
    insl(0x1F0, dst, SECTSIZE / 4);
}
```

其基本功能为读取一个磁盘扇区，关于具体代码的含义如图注释

根据上述代码，将读取磁盘扇区的过程总结如下：

1 等待磁盘直到其不忙；

2 往 0x1F2 到 0X1F6 中设置读取扇区需要的参数，包括读取扇区的数量以及 LBA 参数；

3 往 0x1F7 端口发送读命令 0X20；

4 等待磁盘完成读取操作；

5 从数据端口 0X1F0 读取出数据到指定内存中；

4.2bootloader 是如何加载 ELF 格式的 OS？

bootloader 加载 ELF 格式的 OS 的代码位于 bootmain.c 中的 bootmain 函数中，接下来根据代码来开始分析加载过程

```
bootmain(void) {
    // read the 1st page off disk
    readseg((uintptr_t)ELFHDR, SECTSIZE * 8, 0);

    // 判断是否是合法的ELF文件
    if (ELFHDR->e_magic != ELF_MAGIC) {
        goto bad;
    }
```

首先，从磁盘的第一个扇区（第零个扇区为 bootloader）中读取 OS kenerl 最开始的 4kB 代码，然后判断其最开始四个字节是否等于指定的 ELF_MAGIC，用于判断该 ELF header 是否合法；

```
struct proghdr *ph, *eph;

// EFL头部有program header表的偏移，让ph指向program header表的第一项，并循环将所有程序加载进内存
ph = (struct proghdr *)((uintptr_t)ELFHDR + ELFHDR->e_phoff);
//END OF PH，标记了EFL文件头部的结尾
eph = ph + ELFHDR->e_phnum;
//按照程序头表的描述把EFL文件的数据加载进内存
for (; ph < eph; ph ++) {
    readseg(ph->p_va & 0xFFFFFF, ph->p_memsz, ph->p_offset);
}
```

接下来从 ELF 头文件中获取 program header 表的位置，以及该表的入口数目，然后遍历该表的每一项，并且从每一个 program header 中获取到段应该被加载到内存中的位置（Load Address，虚拟地址），以及段的大小，然后调用 readseg 函数将每一个段加载到内存中，至此完成了将 OS 加载到内存中的操作；

```
// 根据EFL表头的入口信息找到内核入口并开始执行，不返回
((void (*)(void))(ELFHDR->e_entry & 0xFFFFFF))();
```

最后一个步骤就是从 ELF header 中查询到 OS kernel 的入口地址，然后使用函数调用的方式跳转到该地址上去。