Lab1 实验报告

一、实验目标

学习multiboot启动协议，了解操作系统的启动过程。

二、实验原理

multiboot启动协议规定了引导加载程序和一个操作系统之间的接口，从而使任意一个遵循该协议的启动器能启动任意一个遵循该协议的操作系统。该协议并未规定一个boot loader要如何工作，而只是提供一个规范的接口让boot loader和OS就如何启动OS这一问题进行通信。这主要包含三个方面：

1. 操作系统镜像应该具有的格式（Multiboot header）
2. 引导加载程序启动操作系统时机器的状态。
3. 引导加载程序给OS传递信息的格式。

该实验中我们在.S文件中编写了Multiboot header。

QEMU是一个通用且开源的机器模拟器和虚拟化工具。在本实验中，它为我们编写的带有Multiboot header的OS的运行提供系统模拟硬件（包含虚拟VGA和串UART），同时它也提供了遵循multiboot协议的启动器程序。

VGA的显存起始位置为0xB8000，输出一个字符需要两个字节，除了字符本身需要外，另一个字节存储显示属性。

三、源代码说明

首先是multiboot header：`magic`必须为`0x1BADB002`，`flags`字段设为`0x00000000`，`checksum`字段由`-(0x1BADB002+0x00000000)`计算得到。由于`flags[16],flags[2]`为0，所以`header_addr`等5个字段及`mode_type`等4个字段均无需设置。

然后是使用VGA输出字符串。连续使用多条`movl $0x71xx71yy, 0xB80zz`指令。输出格式统一采用白底蓝字，前一个字符对应编码`0xyy`，后一字符对应`0xxx`，例如输出PB两个字符（P、B的ASCII编码分别为0x50、0x42）的指令为`movl $0x71427150, 0xB8000`。然后由于一条指令将4字节写入VGA显存，所以下一条指令的地址必须增加4（Bytes），例如输出PB后需要输出20，那么指令为`movl $0x71307132, 0xB8004`，输出地址变为了`0xB8004`。最后使用了8条`movl`指令成功输出了学号-名字缩写。

最后是UART串口输出。首先使用`movw $0x3F8, %dx`指令设置串口地址；然后使用`movb $0x4C, %al`将L的编码写入al寄存器（因为`out`不接受立即数参数），再用`out %al, %dx`输出L；重复上述步骤，不需要更改输出端口地址，输出若干字符。

四、代码布局说明

文件multibootHeader.S中，multiboot header必须以longword（32位）的方式对齐；由于只有`magic`、`flags`和`checksum` 3个必要的字段，所以multiboot header只占用96位，即12个字节。

使用VGA输出一个字符需要两个字节，除了字符本身需要外，另一个字节存储显示属性。每条`movel`指令需要输出两个字符，即往VGA显存写入4个字节，而输出地址以字节为单位，所以两个`movl`指令中间目标地址相差为4。

五、编译过程说明

生成bin文件：`make`指令，执行MakeFilew，等价于以下两条指令：

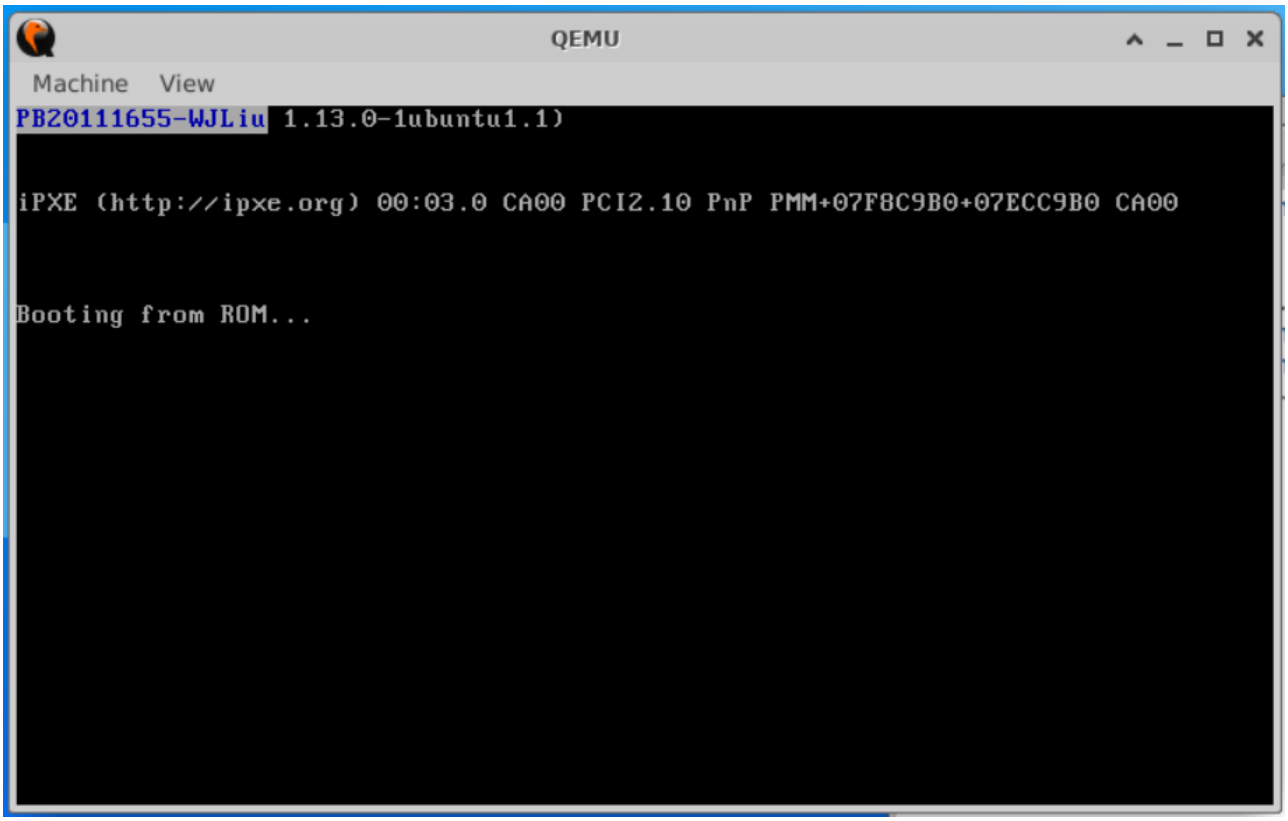
```
gcc -c -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
multibootHeader.S -o multibootHeader.o
ld -n -T multibootHeader.ld multibootHeader.o -o
multibootHeader.bin
```

运行QEMU：`qemu-system-i386 -kernel multibootHeader.bin -serial stdio`

六、实验结果

执行`make`指令后生成了multibootHeader.bin文件（还有multibootHeader.o文件）。

执行`qemu-system-i386 -kernel multibootHeader.bin -serial stdio`后出现QEMU窗口，输出如下图：



而shell窗口如下：