操作系统实验报告

实验一 qemu+multiboot启动

学号：PB18111683
姓名：童俊雄
完成时间：2020-03-02

一、 原理说明

根据multiboot协议和x86汇编规范，在ubuntu中编写符合要求的汇编文件multibootHeader.S、用于指定相应的二进制代码布局的链接描述文件multibootHeader.ld和用于引导make指令执行编译操作的Makefile文件，通过make指令编译生成可被执行的.bin文件，再通过qemu读取.bin文件到内存中并运行该程序。

在运行过程中，该程序直接在VGA显存中写入需要输出的内容，并将内容显示在屏幕上；另外该程序还在端口地址中依次串口输出需要输出的字符。

二、 源代码说明

Makefile说明：

（黑色部分为代码，紫色为说明）

```
ASM_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector

multibootHeader.bin: multibootHeader.S
```
（核心部分，声明make指令的具体操作）
```
    gcc -c ${ASM_FLAGS} multibootHeader.S -o multibootHeader.o
```
（用gcc编译multibootHeader.S生成目标文件multibootHeader.o）
```
    ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin
```
（根据multibootHeader.ld的部署要求，把.o文件链接成.bin文件）
```
clean:
    rm -rf ./multibootHeader.bin ./multibootHeader.o
```
（声明make clean指令的操作：清除.o和.bin文件，以便重新编译）

multibootHeader.ld说明：

```
OUTPUT_FORMAT("elf32-i386", "elf32-i386", "elf32-i386")
```
（格式为elf）
```
OUTPUT_ARCH(i386)
```
（表示支持结构为x86）
```
ENTRY(start)
```
（以汇编文件中定义的start为入口）
```
SECTIONS
```
（表示代码如何排布）

{ . = 1M;
（从物理内存的1M，即0x100000位置开始放代码）

    .text : {
        *(.multiboot_header)

（在1M位置放入12字节的.multiboot_header）

         . = ALIGN(8);

（8个字节对齐）

          *(.text)
        }
}

multibootHeader.S说明：

```
.section PB18111683
.text
.global    start

start:
            .long 0x1badb002
```

（魔数，检验是否为muitboot协议的“接头暗号”）

```
            .long 0x0
            .long -0x1badb002

si:    //output through serial interface
```

（串口输出）

```
        movw $0x3f8, %dx
```

（往EDX寄存器存入端口地址0x3f8）

```
            nop
            nop
```

（留空，对齐代码）

```
//output every character of "helloworldPB18111683"
        movb $0x68, %al
```

（0x68为字符‘h’的ASCII码，存入EAL寄存器）

```
        outb %al, %dx
```

（将EAL寄存器中的字符输出到串口）

```
        movb $0x65, %al
```

（类似上两行，此处为字符‘e’）

        outb %al, %dx[i]

（si部分此后的代码与上面四行类似，在此不再赘述，贴在文末注释中）

```
VGA:
//output in vga, each command writes two chars into the vga video memory
```

movl $0x2f652f68, 0xB8000

（十六进制数‘2f’对应VGA显示属性中的绿底白字；十六进制数‘65’和‘68’分别是‘e’和‘h’的ASCII码，分别存放于显存的第二个地址（0xB8002）和第一个地址（0xB8000）中）

movl $0x2f6c2f6c, 0xB8004[ii]

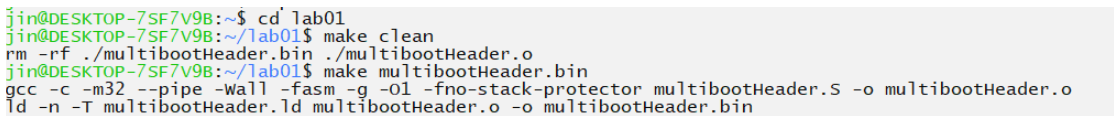
（VGA部分此后的代码与上面两行类似，在此不再赘述，贴在文末注释中）

jl VGA（在VGA中持续显示，可通过在shell中键盘输入ctrl+C终止）

三、 代码布局（地址空间）说明

由multibootHeader.ld的代码说明可知，multibootHeader.bin文件中只有一个section（即.text），此section所在的位置是从1M处开始的，代码对齐方式为8字节对齐，前12个字节的内容为魔数，由于是8字节对齐，SI输出和写VGA显存的代码是从第16个字节开始放置的。

四、 编译过程说明



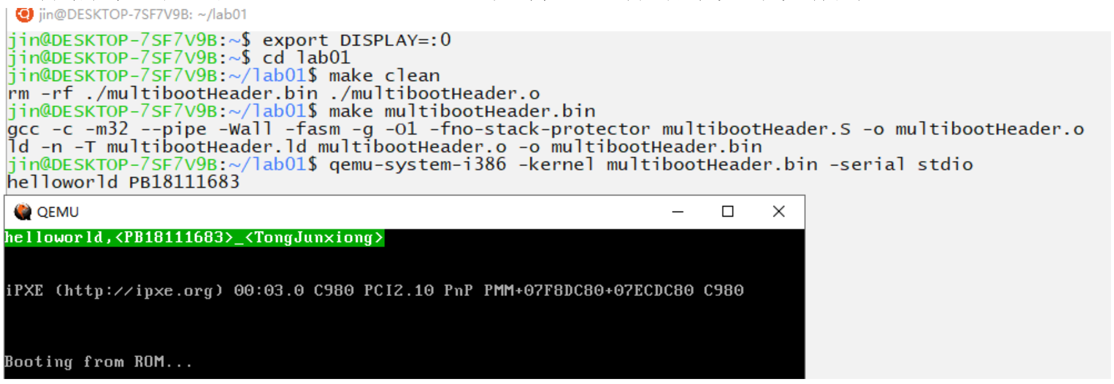
```
jin@DESKTOP-7SF7V9B:~$ cd lab01
jin@DESKTOP-7SF7V9B:~/lab01$ make clean
rm -rf ./multibootHeader.bin ./multibootHeader.o
jin@DESKTOP-7SF7V9B:~/lab01$ make multibootHeader.bin
gcc -c -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector multibootHeader.S -o multibootHeader.o
ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin
```

执行make multibootHeader.bin指令，根据Makefile文件可知，编译过程为：

1. 用gcc编译multibootHeader.S生成目标文件multibootHeader.o
2. 根据multibootHeader.ld的部署要求，把.o文件链接成.bin文件

五、 运行和运行结果说明

通过指令qemu-system-i386 -kernel multibootHeader.bin -serial stdio
运行编译生成的multibootHeader.bin文件，运行结果如下如所示：



jin@DESKTOP-7SF7V9B: ~/lab01

```
jin@DESKTOP-7SF7V9B:~$ export DISPLAY=:0
jin@DESKTOP-7SF7V9B:~$ cd lab01
jin@DESKTOP-7SF7V9B:~/lab01$ make clean
rm -rf ./multibootHeader.bin ./multibootHeader.o
jin@DESKTOP-7SF7V9B:~/lab01$ make multibootHeader.bin
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
jin@DESKTOP-7SF7V9B:~/lab01$ qemu-system-i386 -kernel multibootHeader.bin -serial stdio
helloworld PB18111683
```

QEMU

```
helloworld,<PB18111683>_<TongJunxiong>


iPXE (http://ipxe.org) 00:03.0 C980 PCI2.10 PnP PMM+07F8DC80+07ECDC80 C980


Booting from ROM...
```

如图所示，

1. VGA输出绿底白字的“helloworld,<PB18111683>_<TongJunxiong>”字样；
2. 串口输出“helloworld”和学号“PB18111683”

六、 遇到的问题和解决办法

问题：由于本人对Linux命令行和汇编语言知之甚少，这次实验可以说是有些无从下手，写汇编过程中make指令报错频频；

解决：花了大量时间查阅相关资料，并请教已经较为了解的同学。

问题：由于第一次使用ubuntu，实验中要用到的make和gcc等工具都没有安装好；

解决：按照系统提示用install指令下载工具。

问题：在运行最后一条指令时提示“Could not initialize SDL(No available video device)”

解决：在群内求助，得到胡同学的指导，按照手册下载安装Xming并运行，并输入WSL指令“export DISPLAY=:0”

---

[i]

```
movb $0x6c, %al
outb %al, %dx
movb $0x6c, %al
outb %al, %dx
movb $0x6f, %al
outb %al, %dx
movb $0x77, %al
outb %al, %dx
movb $0x6f, %al
outb %al, %dx
movb $0x72, %al
outb %al, %dx
movb $0x6c, %al
outb %al, %dx
movb $0x64, %al
outb %al, %dx
movb $0x20, %al
outb %al, %dx
movb $0x50, %al
outb %al, %dx
movb $0x42, %al
outb %al, %dx
movb $0x31, %al
outb %al, %dx
movb $0x38, %al
outb %al, %dx
movb $0x31, %al
outb %al, %dx
movb $0x31, %al
outb %al, %dx
movb $0x31, %al
```

```
outb %al, %dx
movb $0x36, %al
outb %al, %dx
movb $0x38, %al
outb %al, %dx
movb $0x33, %al
outb %al, %dx
```

[ii]

```
movl $0x2f772f6f, 0xB8008
movl $0x2f722f6f, 0xB800C
movl $0x2f642f6c, 0xB8010
movl $0x2f3c2f2c, 0xB8014
movl $0x2f422f50, 0xB8018
movl $0x2f382f31, 0xB801C
movl $0x2f312f31, 0xB8020
movl $0x2f362f31, 0xB8024
movl $0x2f332f38, 0xB8028
movl $0x2f5f2f3e, 0xB802C
movl $0x2f542f3c, 0xB8030
movl $0x2f6e2f6f, 0xB8034
movl $0x2f4a2f67, 0xB8038
movl $0x2f6e2f75, 0xB803C
movl $0x2f692f78, 0xB8040
movl $0x2f6e2f6f, 0xB8044
movl $0x2f3e2f67, 0xB8048
```