

中国科学技术大学计算机学院  
《操作系统原理与设计》报告



实验题目： lab1 Multiboot 启动

学生姓名： 胡毅翔

学生学号： PB18000290

完成日期： 2020 年 2 月 20 日

计算机实验教学中心制

2019 年 09 月

实验目的

- 1. 了解Multiboot启动协议
- 2. 编写一个支持Multiboot启动协议的helloworld
- 3. 学习Git的使用

实验环境

- 1. PC一台
- 2. Windows系统
- 3. Ubuntu
- 4. qemu
- 5. Xserver

原理说明

启动协议

Multiboot启动协议是描述启动加载程序如何加载x86操作系统内容的开放标准。协议允许任何兼容的引导加载程序实现引导任何兼容的操作系统内核。因此，它允许不同的操作系统和引导加载程序协同工作和交互，而不需要特定于操作系统的引导加载程序。因而，它使得在一台计算机上更容易地同时存在不同的操作系统，这也被称为多引导（Multi-booting）。

在该协议下，引导加载程序能将CPU转换到保护模式，使得用户可以访问所有存储空间，且能使用A20总线，同时全局描述表和中断描述表尚未定义。Multiboot协议下的系统内核可以为ELF格式，且在设置12个字节的前提下便能正确运行。

QEMU

QEMU是一个快速的处理器仿真器，它使用动态转化来实现良好的仿真速度。

QEMU有两种仿真模式，一种为完整系统仿真，另一种为用户模式仿真，实验中我们使用的为后者。QEMU可以在没有主机内核驱动程序的情况下运行，但提供了可靠的性能。

VGA输出

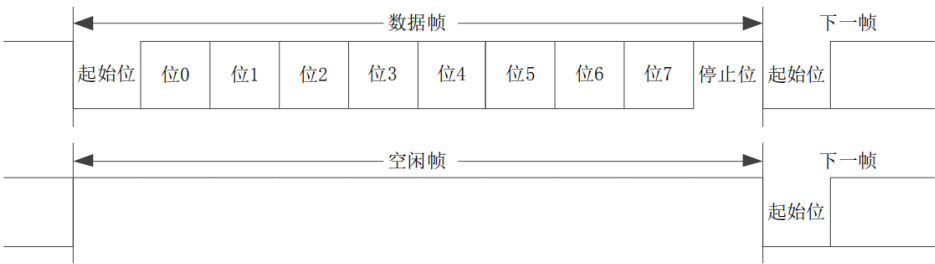
VGA显存的起始地址为0xB8000每个字符输出需要两个字节，一个用于存放ASCII码，一个用于存放该字符的显示属性，格式如下：

显示属性								字符信息							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
闪烁		背景颜色			字符颜色			字符的ASCII码							

输出的代码格式为，movl \$s1,\$s2（其中\$1为字符信息，\$2为输出地址）。

串口输出

本实验中使用的串口遵循UART协议。UART是一种异步串口通信协议，其工作原理是将传输数据的每个字符一位接一位地传输。由于串行接口没有时钟信号，因此需要在收发两侧约定好一个特定的数据收发频率和数据格式。串口协议中支持的数据收发频率（又称波特率，bps）有多种，如 9600、19200、115200、256000 等，以 115200 为例，表示 1s 钟可以传送 115200 位的数据，本实验中，在没有初始化波特率等情况下，直接写UART，QEMU上不出错。串口的收发信号采用相同的数据格式，如下数据帧所示，当没有数据需要发送时，可以发送空闲帧，如下图空闲帧所示。



每一数据帧都包含“起始位+数据位+停止位”，两帧之间可以插入始终为高电平信号的空闲帧，根据协议，数据帧起始位为低电平、停止位为高电平，数据位长度可选择5~8中的任意数字，在数据位和停止位之间还可以包含奇偶校验位。本实验中端口地址为0x3F8，输出字符的汇编代码为：

```
movb $0x46, %a1 /*读字符F的ASCII码，到a1*/
movw $0x3F8, %dx /*读端口地址0x3F8，到dx*/
outb %a1, %dx /*输出字符F*/
```

# 源代码说明

## Makefile

```
ASM_FLAGS= -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector
/*汇编的标记*/
multibootHeader.bin: multiboot.S/*从.S文件编译链接生成.bin文件*/
    gcc -c ${ASM_FLAGS} multiboot.S -o multibootHeader.o/*生成目标文件*/
    ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin
/*按照.ld文件要求链接为.bin文件*/
clean:
    rm -rf ./multibootHeader.bin ./multibootHeader.o
/*删除中间文件*/
```

## multibootHeader.ld

```
OUTPUT_FORMAT("elf32-i386", "elf32-i386", "elf32-i386")/*表明输出格式为elf格式*/
OUTPUT_ARCH(i386)/*表示支持的结构为x86*/
ENTRY(start)/*表示以汇编文件中的start为入口*/

SECTIONS {
    . = 1M; /*从物理内存的1M开始放数据&代码*/
    .text : {
        *(.multiboot_header)/*multibootHeader的12个字节*/
        . = ALIGN(8); /*向后对齐8个字节*/
        *(.text)/*VGA输出所用的地址空间*/
    }
}
```

## multiboot.S

```
/*multiboot.S*/

#define MULTIBOOT_HEADER_MAGIC    0x1BADB002
#define MULTIBOOT_HEADER_FLAGS    0x00000000

    .text

    .globl  start, _start

start:

_start:
    jmp multiboot_entry

    /* Align 32 bits boundary. */
    .align 4

    /* Multiboot header. */
multiboot_header:
    /* magic */
    .long  MULTIBOOT_HEADER_MAGIC
    /* flags */
    .long  MULTIBOOT_HEADER_FLAGS
    /* checksum */
    .long  -(MULTIBOOT_HEADER_MAGIC + MULTIBOOT_HEADER_FLAGS)

multiboot_entry:

    /*UART output part*/
    movb $0x48,%al/* 输出字母H */
    movw $0x3F8,%dx
    outb %al,%dx

    movb $0x45,%al/* 输出字母E */
    outb %al,%dx
    /*输出其他字符只需将对应ASCII码，移动至al寄存器中，重复上述指令即可*/
    /*实验报告中从简，故略去*/

    /*VGA output part*/
    /*绿底白字*/
    movl $0x2f652f68, 0xB8000 /*输出 E H*/
    movl $0x2f6c2f6c, 0xB8004 /*输出 L L*/
    /*输出其他字符只需修改对应ASCII码，并修改地址，重复上述指令即可*/
    /*实验报告中从简，故略去*/

loop:    hlt
```

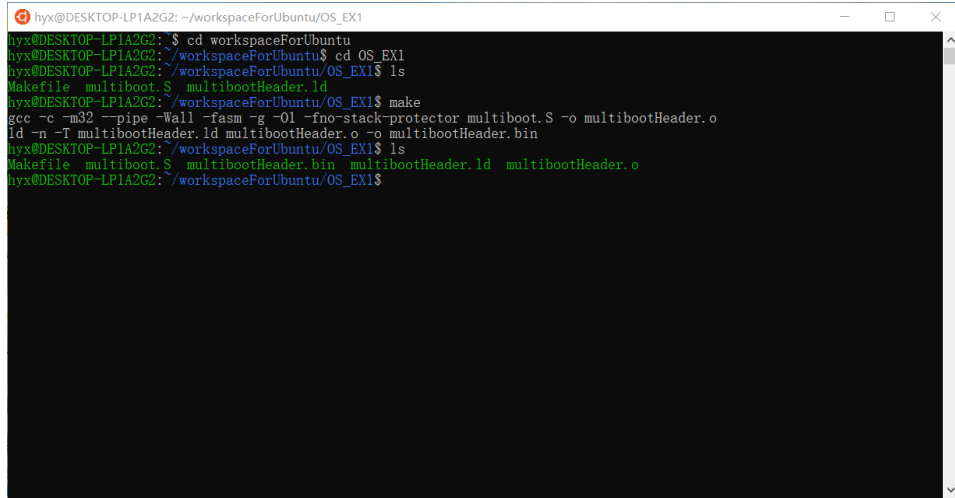
```
jmp loop
```

## 代码布局说明（空间地址）

从物理地址1M的位置开始放数据和代码，前12个字节为multibootHeader的部分：前4个字节为magic的值（0x1BADB002），中间4个字节为flags的值（0x0），最后四个字节为checksum的值（-0x1BADB002）。之后向后对齐8个字节，从第17个字节开始用于程序中的VGA输出及串口输出。

## 编译过程说明

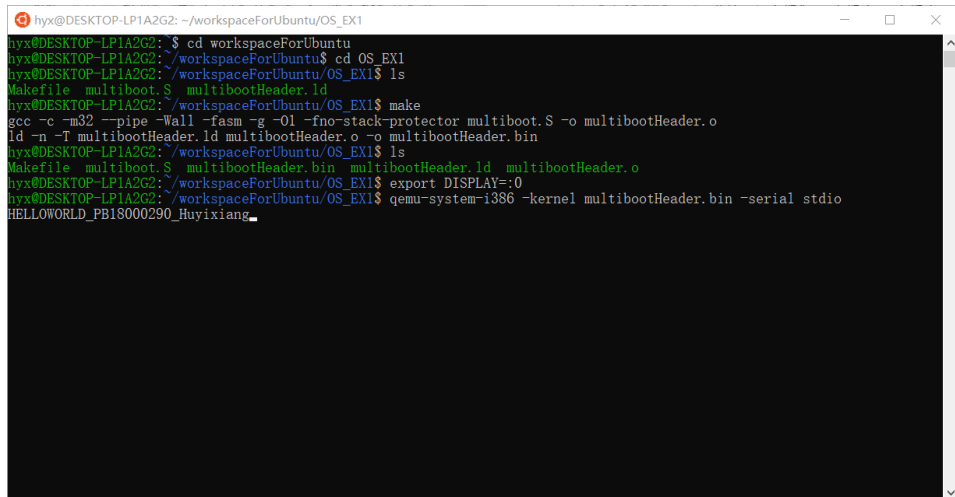
1. 在OS\_EX1目录下存放已完成的Makefile,multiboot.S,multibootHeader.ld文件.
2. 执行make指令.
3. GCC编译multiboot.S文件生成multibootHeader.o文件，而后根据multibootHeader.ld文件要求链接生成multibootHeader.bin文件.
4. 完成编译过程，OS\_EX1目录下增加了编译过程中生成的multibootHeader.o及multibootHeader.bin文件.



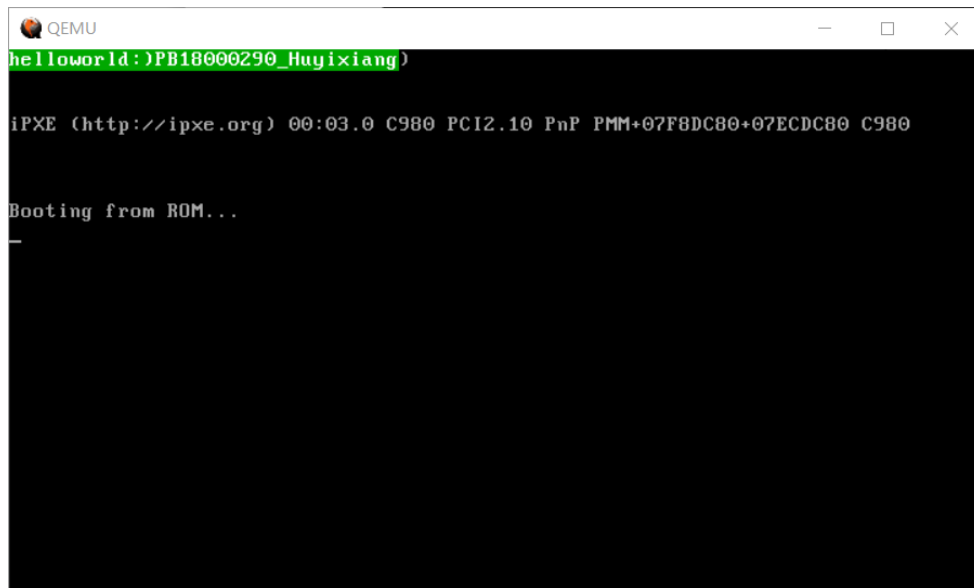
```
hyy@DESKTOP-LP1A2G2: ~/workspaceForUbuntu/OS_EX1
hyy@DESKTOP-LP1A2G2: $ cd workspaceForUbuntu
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu$ cd OS_EX1
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ ls
Makefile  multiboot.S  multibootHeader.ld
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ make
gcc -c -m32 -pipe -Wall -fasm -g -O1 -fno-stack-protector multiboot.S -o multibootHeader.o
ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ ls
Makefile  multiboot.S  multibootHeader.bin  multibootHeader.ld  multibootHeader.o
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$
```

## 运行和运行结果说明

1. 编译链接后生成multibootHeader.bin文件.
2. 启动Xming.
3. 在WSL命令执行中执行export DISPLAY=:0.
4. 在WSL命令执行中执行qemu-system-i386 -kernel multibootHeader.bin -serial stdio.
5. 运行multibootHeader.bin文件.
6. 运行结果如图.



```
hyy@DESKTOP-LP1A2G2: ~/workspaceForUbuntu/OS_EX1
hyy@DESKTOP-LP1A2G2: $ cd workspaceForUbuntu
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu$ cd OS_EX1
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ ls
Makefile  multiboot.S  multibootHeader.ld
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ make
gcc -c -m32 -pipe -Wall -fasm -g -O1 -fno-stack-protector multiboot.S -o multibootHeader.o
ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ ls
Makefile  multiboot.S  multibootHeader.bin  multibootHeader.ld  multibootHeader.o
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ export DISPLAY=:0
hyy@DESKTOP-LP1A2G2: /workspaceForUbuntu/OS_EX1$ qemu-system-i386 -kernel multibootHeader.bin -serial stdio
HELLOWORLD_PB18000290_Huyixiang_
```



7. 关闭QEMU界面后，WSL返回到等待执行命令输入的状态。

## 遇到的问题 and 解决方案说明

---

问题1：运行`apt-get`指令时速度过慢。

解决方案：

1. 在WSL命令行中执行`sudo sed -i 's/archive.ubuntu.com/mirrors.ustc.edu.cn/g' /etc/apt/sources.list`
2. 将`/etc/apt/sources.list`文件中Ubuntu默认的源地址`http://archive.ubuntu.com/`替换为`http://mirrors.ustc.edu.cn`即可。

问题2：编译生成`multibootHeader.bin`文件后，运行时显示`Could not initialize SDL(No available video device) - exiting.`

解决方案：

1. 下载、安装Xming，运行之。
2. 在WSL 命令行中执行 `export DISPLAY=:0.`
3. 重新运行`.bin`文件。