**Lab1 Multiboot启动**

PB22000079 吴一凡

一、实验目的及环境

本次实验要求自己搭建实验环境并实现一个最原始的操作系统，该系统包含一个 Multiboot 启动头和一个最简单的操作系统内核。其中Multiboot 启动头用于启动最初最简单的 OS 内核。

对于实验环境，本人选用了在Win11下结合WSL2和Ubuntu，利用VcXsrv可视化工具运行Linux系统以及Qemu的实验环境进行实验。

1. 实验原理说明

Multiboot启动协议

Multiboot启动协议是一套多种操作系统共存时的启动协议，实际上Multiboot提供了一个规范作为一个开放标准。Multiboot协议为多种内核提供了一种统一的引导方式，同时该方式可以由任意的符合Multiboot的引导加载程序引导。该协议明确了bootloader和操作系统间的相关接口，从而使得所有符合规范的bootloader可以互不影响的引导加载所有依规范编辑的操作系统。

最简单的Multiboot Header包含如下的三个域：

魔数域，填充值为0x1BADB002；

flags域，即标志域，指出OS映像需要bootloader提供的特性，本次实验为0即可；

校验和域，当checksum + magic + flags = 0时校验通过，Multiboot Header构建完成。

QEMU

Qemu是纯软件实现的虚拟化模拟器，几乎可以模拟任何硬件设备。通过Qemu可以实现虚拟的裸机环境并运行本实验所实现的操作系统内核，虚拟机与Qemu模拟的硬件实现接口的交互，而Qemu将这些指令转译给实际的硬件做处理。

VGA及串口输出

本实验通过直接向VGA显存写入内容来实现字符输出，实验中字符界面规格为：25 行 80 列，VGA 显存的范围为：0xB8000 + 0x1000，我们从起始地址 0xB8000 开始写入要显示的文本。VGA 显存显示一个字符需要两个字节，一个用于存放字符的 ASCII 码，另一个用于存放该字符的显示属性（如前景色、背景色等），可以使用mov指令加载指定的内存然后输出。

对于串口输出，串口端口地址为 0x3F8。串口输出属于端口映射 I/O，因此调用 out 进行串口输出。我们只需要提供 outb 所需的两个参数即可。具体来说，首先将串口端口地址存入 dx 寄存器，然后将要输出的字符的 ASCII 码存入 al 寄存器，最后调用 out 指令即可。

1. 源代码说明

MultibootHeader部分

以下是参数定义和声明部分：

MAGIC\_ITEM\_NAME = 0x1BADB002

FLAGS\_ITEM\_NAME = 0x00000000

CHECKSUM\_ITEM\_NAME = 0x00000000 - FLAGS\_ITEM\_NAME - MAGIC\_ITEM\_NAME

.section ".multiboot\_header"

.align 4

.long MAGIC\_ITEM\_NAME

.long FLAGS\_ITEM\_NAME

.long CHECKSUM\_ITEM\_NAME

在这个部分，定义了MultibootHeader需要的magic，flags和checksum三个域，以满足Multiboot协议规定的magic + flags + checksum = 0 的要求。

VGA和串口输出

查阅资料，得知显存中可能会有未经初始化的内容，于是为了程序的完备性，使用对显存进行初始化来解决这个问题，具体的实现中发现在本机上即使不做初始化仍然能正确运行，故这部分不在实验报告中给出而附在代码中。

vgaout:

movl $0x2f652f48, 0xB8000 #e H

movl $0x2f6C2f6C, 0xB8004 #l l

movl $0x2f202f6F, 0xB8008 #space o

......

uartout:

movw $0x3F8, %dx

movb $0x48, %al

outb %al, %dx

movb $0x65, %al

outb %al, %dx

......

hlt

上述串口输出部分的第一行表示将串口端口的地址放入dx寄存器中，然后不断地使用mov语句将要输出的内容写入内存并用out来输出。最后使用hlt语句停机并让处理器进入暂停状态。

1. 代码布局说明

代码布局（地址空间）来源于链接描述文件(MultibootHeader.ld)中的SECTIONS部分：

SECTIONS {

. = 1M;

.text : {

\*(.multiboot\_header)

. = ALIGN(8);

\*(.text)

}

}

注意到这里的参数(.multiboot\_header)应当与我们在MultibootHeader.S中定义的section名称保持一致,以让Qemu正确的识别到Header。根据文件还可以得知，输出文件.text代码段的偏移值为1M，从内存1M处开始存储代码段落。

1. 编译过程说明

根据Makefile文件的内容：

ASM\_FLAGS = -m32 --pipe -Wall -fasm -g -O1 -fno-stack-protector

multibootHeader.bin: multibootHeader.S

gcc -c ${ASM\_FLAGS} multibootHeader.S -o multibootHeader.o

ld -n -T multibootHeader.ld multibootHeader.o -o multibootHeader.bin

clean:

rm -rf ./multibootHeader.bin ./multibootHeader.o

可以得知，本次实验的编译过程实际上是利用gcc从.S源文件编译出.o文件，然后利用.ld文件从.o文件链接得到最终的.bin文件。同时由clean部分我们知道，键入指令make clean可以删除.o和.bin文件。

1. 运行结果

在Ubuntu下指定文件夹打开终端，命令行下先后键入:

make

qemu-system-i386 -kernel multibootHeader.bin -serial stdio

得到如下的结果，这表明Qemu正常启动且串口输出正常。

