

lab1 实验文档

1 实验说明

本实验是系列实验中的第一个，用于按照 Multiboot 协议启动一个操作系统。

2 实验目的

实现一个最初的操作系统，该系统包含一个 Multiboot 启动头和一个最初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。

3 实验要求

- (1) 所完成的 Multiboot 启动头能够被 bootloader 识别
- (2) 能够启动所完成的操作系统内核
- (3) 这个操作系统内核无需复杂的功能，只需在屏幕和串口上输出特定内容。

4 实验准备（预备知识和准备工作）

4.1 预备知识：操作系统开发过程

简单而言，操作系统的开发过程如下：

准备一个操作系统开发平台和一个操作系统运行平台。

我们在开发平台中进行操作系统的设计与实现，生成操作系统映像；然后将操作系统映像安装到运行平台上；最后运行并测试这个操作系统。

操作系统开发平台，通常称为主机端，可以是真实或虚拟的平台，一般已安装好主机操作系统和必要的开发软件。对于初学者，建议使用虚拟平台，避免因实验误操作破坏真实计算机中的系统或数据。

主机操作系统与开发软件之间具有一定的依赖关系。因为开发软件是运行在主机操作系统中的。

开发软件的选择：要有操作系统的开发能力，可以仿照 Linux 内核（或其他开源操作系统）的开发软件。

操作系统运行平台，通常称为目标端，可以是真实或虚拟的计算机，又称目标机。开发好的操作系统在目标机上运行。

4.2 开发环境及其搭建

基于 Linux 的开发环境及其搭建

4.2.1 WSL 是什么？

WSL 是 windows subsystem for linux 的简写，指的是 windows 的一个子系统，这个子系统的作用是在 windows 下运行 linux 操作系统。更多请查看微软 WSL 文档

4.2.2 配置方案

(1) WSL + QEMU

- WSL + VcXsrv（可视化工具）：教程
为方便操作，推荐按 PPT 中方法建立共享文件夹
- 换源：WSL 默认源下载 QEMU 可能会出错，所以需要换源：教程
- QEMU：Ubuntu 命令行中输入指令 `sudo apt-get install qemu`
- 下载其他软件：看 Makefile 文件想想还需要下什么

(2) VMware + QEMU

- VMware：查看文档《windows 安装 VMware 教程》

其他方案可以自己尝试配置

4.3 启动协议和 Multiboot 启动协议简介

Multiboot 启动协议是一套多种操作系统共存时的启动引导协议。现今，许多操作系统都拥有自己的 boot loader（即引导程序，负责载入最终的操作系统映像的一个或一组程序），若要使多种操作系统在机器上共存，则需要借助于特定的引导机制，显然，这很可能对使用多操作系统的便利性与兼容性带来影响。为解决这一问题，Multiboot 启动协议应运而生。简单来讲，这份协议明确了 boot loader 与操作系统间的接口，所有符合协议规范的 boot loader 可以读取所有符合协议规范的操作系统。

本次实验采用的协议版本为 Multiboot Specification version 0.6.96，实验目的为实现 Multiboot header 部分，并在 vga 屏幕和串口上输出特定内容。本次实验要求比较简单，我们只需实现最简单的 Multiboot header 格式要求即可。

最简单的 Multiboot Header 要求如下：

Offset 偏移量	Type 数据类型	Filed 域名称	Note 备注
0	u32	magic	必要项，值应为 0x1BADB002
4	u32	flags	必要项，在此次实验中值可以为 0
8	u32	checksum	必要项，应满足 $\text{magic} + \text{flags} + \text{checksum} = 0$
.....

即一个 Multiboot Header 应至少具有以上共 12 个字节的内容。下面是 Header 中各个域的作用：

- (1) magic 为魔数域，其值是协议开发者人为规定的，作为 Multiboot Header 的标志。
- (2) flags 域指出 OS 映像需要 boot loader 提供或支持的一些特性。本实验中，可以让 flags 的值为 0，以后我们将对 flags 有更深入的了解。
- (3) checksum 域为校验和，顾名思义，用于校验。当满足 $\text{magic} + \text{flags} + \text{checksum} = 0$ 时，校验通过，Multiboot Header 构建完成。

协议要求 Multiboot Header 必须完整地包含在 OS 映像前 8192 个字节内，且通常来说位置要靠前越好。

4.4 屏幕输出之 VGA 简介

本实验直接向 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）。

Attribute 显示属性	Character 字符的 ASCII 码
15 ~ 8	7 ~ 0

显示属性 Attribute 的内容规定如下：

Blink 闪烁	Background Color 背景色	Foreground Color 前景色
7	6 ~ 4	3 ~ 0

写显存输出字符属于内存映射 I/O，因此使用 mov 进行输出。格式为

```
movx source, destination
```

其作用为将 source 的值写入 destination。movl 适用于 32 位的值，因此我们用 movl 语句向显存写入内容。

接下来，查阅所要显示字符的 ASCII 码，遵照实验原理部分 VGA 输出字符的格式要求，逐个输出字符即可。

```
start:
    movl $0x2f652f48, 0xb8000    /* e H */
    movl $0x2f6c2f6c, 0xb8004    /* l l */
    movl $0x2f202f6f, 0xb8008    /* \space o */
    ... ..
```

4.5 串口输出之 UART 简介

在 QEMU 中，即使 OS 没有对波特率等进行初始化，也能正确输出。

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。

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。以下代码输出“H”。

```
movw $0x3f8, %dx
movb $0x48, %al
outb %al, %dx
```

4.6 make 和 Makefile 简介

Makefile 文件内容如下

```
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
```

可以看出，编译过程为先用 gcc 从 multibootHeader.S 源文件编译出 multibootHeader.o 文件，再根据链接描述文件 multibootHeader.ld 从 multibootHeader.o 文件链接得到 multibootHeader.bin 文件。

ASM_FLAGS 是一个变量，它定义了本次编译中 gcc 指令的一系列参数。

有了 makefile 文件后，在 ubuntu 的 shell 中，进入源码所在文件夹，输入指令

```
make
```

即可完成编译。

此外，makefile 文件中包含了 clean 部分。如果输入指令

```
make clean
```

可以删除掉之前编译生成的 multibootHeader.o 与 multibootHeader.bin 文件。

4.7 链接和链接脚本简介

链接描述文件（.ld 文件）中的 SECTIONS 部分定义了程序的各个段的起始地址和长度。

```
OUTPUT_FORMAT("elf32-i386", "elf32-i386", "elf32-i386")
OUTPUT_ARCH(i386)
ENTRY(start)

SECTIONS {
    . = 1M;
    .text : {
        *(.multiboot_header)
        . = ALIGN(8);
        *(.text)
    }
}
```

根据 ld 文件，输出文件 .text 代码段的起始偏移量为 1M，即从内存 1M 处开始存储代码段。 .text 代码段含有两部分：

第一部分为 multibootHeader.S 中的 .multiboot_header 代码段，构建了 Multiboot Header，以供 Qemu 识别。此段总长度为 12 字节。

第二部分为 multibootHeader.S 中的.text 代码段，按八字节对齐，因此偏移量为 16。
用表格表示：

Offset (Base = 1M)	Section Name	Note
0	.multiboot_header	multiboot 协议头部，供 QEMU 识别
16	.text	代码段，实现 VGA 输出、串口输出等

5 实验内容

- (1) 参照 4.2.1 及 PPT，搭建开发环境
- (2) 完成 Makefile 和 ld 文件，已经给出，可以根据实际需求修改
- (3) 补全汇编代码中的 Multiboot Header 部分
- (4) 完成功能代码：在屏幕上输出特定内容
- (5) 完成功能代码：初始化串口并在串口输出特定的内容

提交文件框架图如图

```
/
├── doc
│   └── report.pdf.<--你的实验报告
└── src
    ├── Makefile.<--已给出
    ├── multibootHeader.S.<--需补全
    ├── multibootHeader.bin.<--编译后生成
    └── multibootHeader.ld.<--已给出
```

6 实验示例

6.1 编译运行指令

使用 make 指令生成.bin 文件

使用 qemu-system-i386 -kernel multibootHeader.bin -serial stdio, 即可用 QEMU 运行我们生成好的 multiboot OS 核 multibootHeader.bin。

指令详解:

qemu-system-i386 指定平台为 i386

-kernel 指定内核文件

-serial stdio 指定串行终端为标准输入输出

拓展:

终端选项 (-serial -parallel -monitor)

终端包括三类: 串行终端, 并行终端, qemu 命令监控终端。

如果运行 QEMU 后会发现无法拖动鼠标到 QEMU 显示屏外的区域, 请使用 Ctrl + Alt 解决此问题

6.2 实验结果

```
PS C:\Users\kohoy> wsl
tutu@LAPTOP-U0308ESD:/mnt/c/Users/kohoy$ cd ~/
tutu@LAPTOP-U0308ESD:~$ cd OS
tutu@LAPTOP-U0308ESD:~/OS$ cd ex1
tutu@LAPTOP-U0308ESD:~/OS/ex1$ cd src
tutu@LAPTOP-U0308ESD:~/OS/ex1/src$ qemu-system-i386 -kernel multibootHeader.bin -serial stdio
Hello world!PB20000113_konghaoyu
```



7 提交要求

7.1 截止日期：

2024 年 3 月 15 日 23 点 59 分，提交到 BB 平台。

7.2 提交内容：

按照“实验内容”中给出文件框架，提交实验报告和源代码，压缩为 zip 文件，命名为学号 + 姓名 + lab1。

7.3 实验要求：

完成汇编文件 multibootHeader.S

运行 make 指令能够成功生成 multibootHeader.bin

能够使用 qemu 启动，并且成功实现 vga 输出，串口输出

打印内容应显示特异性，如学号加姓名，类似“PB22000001_abo”

7.4 实验报告要求：

需要简单对实验原理进行描述（无需对 ppt 内容进行复制粘贴）

需要对代码运行方法、编译过程进行说明

需要对你的源代码和地址空间分配等做出一定解释，并且在报告中附上你的显示效果截图

可以对自己在实验中所遇到的问题和解决方法进行描述（不做硬性要求）