Lab1实验报告

思考题部分

Thinking 1.1

objdump 在指导书中后跟的参数为 -DS 经查阅资料后得知 -D指反汇编所有节，-S指尽可能反汇编出源代码。

使用原生x86编译工具链反汇编出的可执行文件的main部分：

```
0000000000001149 <main>:
     1149:       f3 0f 1e fa             endbr64
     114d:       55                      push   %rbp
     114e:       48 89 e5                mov    %rsp,%rbp
     1151:       48 8d 05 ac 0e 00 00    lea    0xeac(%rip),%rax        # 2004
<_IO_stdin_used+0x4>
     1158:       48 89 c7                mov    %rax,%rdi
     115b:       e8 f0 fe ff ff          call   1050 <puts@plt>
     1160:       b8 00 00 00 00          mov    $0x0,%eax
     1165:       5d                      pop    %rbp
     1166:       c3                      ret
```

使用交叉编译工具链反汇编出的可执行文件的main部分：

```
004006e0 <main>:
   4006e0:       27bdffe0        addiu   sp,sp,-32
   4006e4:       afbf001c        sw      ra,28(sp)
   4006e8:       afbe0018        sw      s8,24(sp)
   4006ec:       03a0f025        move    s8,sp
   4006f0:       3c1c0042        lui     gp,0x42
   4006f4:       279c9010        addiu   gp,gp,-28656
   4006f8:       afbc0010        sw      gp,16(sp)
   4006fc:       3c020040        lui     v0,0x40
   400700:       24440830        addiu   a0,v0,2096
   400704:       8f828030        lw      v0,-32720(gp)
   400708:       0040c825        move    t9,v0
   40070c:       0320f809        jalr    t9
   400710:       00000000        nop
   400714:       8fdc0010        lw      gp,16(s8)
   400718:       00001025        move    v0,zero
   40071c:       03c0e825        move    sp,s8
   400720:       8fbf001c        lw      ra,28(sp)
   400724:       8fbe0018        lw      s8,24(sp)
   400728:       27bd0020        addiu   sp,sp,32
   40072c:       03e00008        jr      ra
   400730:       00000000        nop
        ...
```

Thinking 1.2

• 尝试使用我们编写的 readelf 程序，解析之前在 target 目录下生成的内核 ELF 文件。

解析的结果如下，该结果为mos文件所有节头中的地址信息,经验证与系统readelf的结果相同：

```
0:0x0
1:0x80400000
2:0x80401a80
3:0x80401a98
4:0x80401ab0
5:0x0
6:0x0
7:0x0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
```

个人编写的可执行文件readelf在编译时采用的是默认标准的gcc格式，因此输出的可执行文件是x86系统下默认的64位可执行文件（ELF64），而readelf本身设计是解析32位的可执行文件，因此自己编写的readelf无法解析readelf本身，而hello则在编译链接时特别增加了*-m32 -static -g*的后缀，因此生成的是32位ELF文件，可以利用自己编写的readelf文件进行解析。

Thinking 1.3

由于启动过程分为两个阶段，第一阶段是由bootloader为操作系统创建必要的硬件环境，并将内核镜像由磁盘导入RAM中，之后才是由内核完成各类初始化工作，完成造作系统的完全启动。因此，体系结构上电时给的启动入口地址往往是占用内存较小的Bootloader的相关代码，而bootloader则会根据内存布局将内核导入到相应的位置。因此，由于bootloader的存在与其stage1,stage2的工作原理，即使实验操作系统的内核入口并没有放在上电启动地址，仍能保证内核入口被正确地跳转到。

难点分析

1.ELF文件格式，架构的深入理解

在经过Lab1的训练和总结后，个人建立了这样的理解：

ELF头(Struct Elf32_Ehdr)

ELF头（一个结构体）是整个ELF文件的标志与特征，存放了该可执行文件中的一些特殊数据。其中在本次作业中重要的部分数据：

e_shoff ： 节头表相较于整个文件的偏移量（字节），根据文件自身的地址加上偏移量可以跳转至节头表处，遍历节头表内的信息。

e_shentsize ： 节头表中每一项（节头）的大小，在获得节头表的起始地址（节头表第一项的地址）后，可以循环增加该变量获得第i个节头的地址，从而获取相应的信息。

e_shnum：节头表表项的数目，可用于在遍历时确定上限。

节头(Struct Elf32_Shdr)

节头表是由一定数量的节头构成的连续的数据区域，节头中则存储着对应节的相关信息，主要在链接器进行链接时使用。

sh_addr：节头所对应的节的具体地址，而非节头自身的地址。

sh_offset：该节头相较于ELF头的偏移量。

sh_size：该节的大小。

在理解了ELF头，节头的结构，相关数据对应的实际意义后，才对ELF文件建立起一个较为有逻辑的认识。也才能完成exercise1.1部分的相关测试。

节(section)

在链接过程中，目标文件被看成节的集合，并使用节头表来描述各个节的组织。其中.txt, .data, .bss为最重要的三个节。

2. printk的具体实现

printk在初看实现代码时感觉非常复杂，让人看起来无从下手，代码填空的形式也使人较难以把握其整体思路。但多加分析，将其分为格式符处理，按格式符输出即可。格式符处理部分，根据格式符的标准形式，改变long_flag（long型变量）, neg_flag（正负标记）, ladjust（左右对齐标记）,padc（补位字符），用于第二步的格式化输出。

在格式化输出时，需要注意默认的print_num是以无符号数的方式进行输出，遇到负数时若直接输出则会输出其补码对应的正数，需要提前进行取反。

实验体会

在本次实验中，我的体会与感受如下：

1. 阅读长篇代码，把握代码逻辑思路的能力需要加强。不仅是os的lab实验，oo的课上实验也是以代码填空的形式出现。不同于自己写代码的时候能够按照自己的，熟悉的思路进行，进行代码填空则更需要能够理清代码的整体思路，明确自己需要干什么，才能较好的完成任务。只根据附近代码照猫画虎一是难以获得实际的提高，二是如果出现了和其他不同的地方（例如printk中输出%d相关的内容时要额外考虑负数情况）也难以直接模仿出正确的结果。

2. 指导书的内容需要仔细反复阅读，不能只为了完成任务而看当前任务相关的部分。例如在上次Lab1的课上实验时就吃了亏，忽略了$?的用法导致无法完成任务。在Lab1之后系统内核之间都是相互链接补充的，在阅读指导书时要阅读各个部分，理清思路，将知识吃透。