

## 北航操作系统实验报告 Lab 1

学号: 71066001

姓名: 陈伟杰

### 思考题

**Thinking 1.1** 请阅读附录中的编译链接详解, 尝试分别使用实验环境中的原生 x86 工具链 (gcc、ld、readelf、objdump 等) 和 MIPS 交叉编译工具链 (带有 mips-linux-gnu-前缀), 重复其中的编译和解析过程, 观察相应的结果, 并解释其中向 objdump 传入的参数含义。

答: objdump 是一个用于反汇编可执行文件、目标文件、共享库等二进制文件的工具。它的常用参数及含义如下:

- D: 反汇编所有节段。
- S: 同时显示反汇编代码和源代码。
- d: 只反汇编代码。
- t: 打印符号表。
- x: 打印所有头部信息。
- j section: 只反汇编指定的节段。
- M reg-names=: 使用指定的寄存器名称。
- r: 显示重定位表。
- G: 显示全局变量。
- g: 显示调试符号。

所以我们可以发现 objdump-DS 就是可以反汇编所有的节段, 并反汇编出源代码  
在这里我先创建了一个.c 的文件, 后来我就编译了它 形成了.o 文件, 最后再反汇编

```
开始连接到 git@10.134.170.244 0.1
Welcome to Ubuntu 22.04.2 LTS (GNU/Linux 5.15.0-46-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:         https://ubuntu.com/advantage
Last login: Thu Mar  9 17:01:59 2023 from 10.134.170.231
git@71066001:~$ cd /71066001/
git@71066001:~/71066001 (make-exercise)$ vim hello.c
git@71066001:~/71066001 (make-exercise)$ mips-linux-gnu-gcc -c hello.c
git@71066001:~/71066001 (make-exercise)$ mips-linux-gnu-ld -o hello hello.o
mips-linux-gnu-ld: 警告: 无法找到项目符号 __start; 缺省为 0000000000400110
git@71066001:~/71066001 (make-exercise)$ mips-linux-gnu-objdump -DS hello > hello.txt
git@71066001:~/71066001 (make-exercise)$ vim hello.txt
git@71066001:~/71066001 (make-exercise)$
```

```
1
2 hello: 文件格式 elf32-tradbigmips
3
4
5 Disassembly of section .MIPS.abiflags:
6
7 004000d8 <.MIPS.abiflags>:
8 4000d8: 00002002 srl a0,zero,0x0
9 4000dc: 01010005 lsa zero,t0,at,0x1
10
11
12 Disassembly of section .reginfo:
13
14 004000f0 <.reginfo>:
15 4000f0: e0000004 sc zero,4(zero)
16
17 400104: 00418160 0x418160
18
19 Disassembly of section .text:
20
21 00400110 <main>:
22 400110: 27bdffff addiu sp,sp,-16
23 400114: afbe080c sw s8,12(sp)
24 400118: 03a0f025 move s8,sp
25 40011c: 2402000c li v0,12
26 400120: afc20000 sw v0,0(s8)
27 400124: 3c020040 lui v0,0x40
28 400128: 24420150 addiu v0,v0,336
29 40012c: afc20004 sw v0,4(s8)
30 400130: 00001025 move v0,zero
31 400134: 03c0e025 move sp,s8
32 400138: 87ae080c lw s8,12(sp)
33 40013c: 27bd0010 addiu sp,sp,16
34 400140: 03e00008 jr ra
35 400144: 00000000 nop
36
37 "hello.txt" 881, 19248
1,0-
```

#Main 的值有改变

**Thinking 1.2** 思考下述问题:

- 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。
- 也许你会发现我们编写的 `readelf` 程序是不能解析 `readelf` 文件本身的, 而我们刚才介绍的系统工具 `readelf` 则可以解析, 这是为什么呢? (提示: 尝试使用 `readelf -h`, 并阅读 `tools/readelf` 目录下的 `Makefile`, 观察 `readelf` 与 `hello` 的不同)

```
git@71066001:~/71066001/tools/readelf (lab1)$ readelf -h hello
ELF 头:
  Magic:      7f 45 4c 46 01 01 01 03 00 00 00 00 00 00 00 00
  类别:                               ELF32
  数据:                               2 补码, 小端序 (little endian)
  Version:                               1 (current)
  OS/ABI:                               UNIX - GNU
  ABI 版本:                               0
  类型:                               EXEC (可执行文件)
  系统架构:                               Intel 80386
  版本:                               0x1
  入口点地址:                               0x8049600
  程序头起点:                               52 (bytes into file)
  Start of section headers:               746252 (bytes into file)
  标志:                               0x0
  Size of this header:                     52 (bytes)
  Size of program headers:                 32 (bytes)
  Number of program headers:               8
  Size of section headers:                 40 (bytes)
  Number of section headers:               35
  Section header string table index: 34
```

答:

```
git@71066001:~/71066001/tools/readelf (lab1)$ readelf -h readelf
ELF 头:
  Magic:      7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  类别:                               ELF64
  数据:                               2 补码, 小端序 (little endian)
  Version:                               1 (current)
  OS/ABI:                               UNIX - System V
  ABI 版本:                               0
  类型:                               DYN (Position-Independent Executable file)
  系统架构:                               Advanced Micro Devices X86-64
  版本:                               0x1
  入口点地址:                               0x1180
  程序头起点:                               64 (bytes into file)
  Start of section headers:               14488 (bytes into file)
  标志:                               0x0
  Size of this header:                     64 (bytes)
  Size of program headers:                 56 (bytes)
  Number of program headers:               13
  Size of section headers:                 64 (bytes)
  Number of section headers:               31
  Section header string table index: 30
```

原因: 因为我们知道 `readelf` 是不支持解析大端存储文件的, 而上面这两张图我们也看到我们所解析出来的文件都是小端文件, 那么 `hello` 和 `readelf` 的不同在于入口点地址的不同

**Thinking 1.3** 在理论课上我们了解到,MIPS 体系结构上电时,启动入口地址为 `0xBFC00000` (其实启动入口地址是根据具体型号而定的,由硬件逻辑确定,也有可能不是这个地址,但一定是一个确定的地址),但实验操作系统的内核入口并没有放在上电启动地址,而是按照内存布局图放置。思考为什么这样放置内核还能保证内核入口被正确跳转到?

(提示:思考实验中启动过程的两阶段分别由谁执行。)

答:实验操作系统的启动过程分为两个阶段:第一阶段由 MIPS 芯片的 **Bootloader** 执行,第二阶段由实验操作系统内核执行。在第一阶段,**Bootloader** 加载实验操作系统内核到指定地址(`0x80100000`),并跳转到内核入口地址(`0x80100000 + 0x28`),该入口地址是在内核编译链接时指定的。因此,即使实验操作系统内核入口不是上电启动地址(`0xBFC00000`),也能保证内核入口被正确跳转到,因为 **Bootloader** 和内核都知道内核入口地址。这种启动方式也增加了操作系统的灵活性,使得内核可以被加载到不同的地址空间中运行。

#### 实验难点:

相对于 lab0 来说,这次实验的难度提升了很多,总体来说还是偏困难,ELF 文件格式的理解,在 exercise 1.2 我遇到了点问题,首先我一开始补充代码的时候,我的理解是错误的,导致我输出的内容到了 1:就不输出了,所以我理解应该要理解 ELF 文件之中的三个结构体并利用文件头地址和偏移量求地址,还有 ELF 文件的编译和运行,这点很重要,在 exercise 1.3 中要知道内核它的位置在哪? Exercise1.4 学会栈指针的使用。

#### 体会和感想

在这次实验虽然是看上去好像很“少”但其实内容很多,学习的知识没有 lab0 那样的清晰易懂,在填代码的部分更多的是你对指导书的理解,我相信经历过这次实验之后我更加深入学会了操作系统,可能后续自己还要继续深入了解 c 语言的指针,因为在这次实验中耗费我时间最长的地方就是指针的操作,总的来说,通过这次实验,我知道对操作系统内核的理解很重要,本次实验会为我们之后较难的 lab 打下良好基础,望自己后面更快完成和理解实验部分。