lab1 实验报告

张翼翔*

2023 年 3 月 11 日

1 思考题

Thinking 1.1

我写了这样一段程序：

```
#include<stdio.h>

int main()
{
        printf("hello world!, mos!");
        return 0;
}
```

使用原生 x86 编译和反汇编:

```
gcc -c helloworld.c -o helloworld
objdump --DS helloworld > helloworld1
```

反汇编结果:helloworld1：

```
文件格式 elf64-x86-64


Disassembly of section .text:

0000000000000000 <main>:
   0:   f3 0f 1e fa             endbr64
   4:   55                      push   %rbp
```

*E-mail:21371055@buaa.edu.cn

```
   5:   48 89 e5                mov    %rsp,%rbp
   8:   48 8d 05 00 00 00 00    lea    0x0(%rip),%rax        # f <main+0xf>
   f:   48 89 c7                mov    %rax,%rdi
  12:   b8 00 00 00 00          mov    $0x0,%eax
  17:   e8 00 00 00 00          call   1c <main+0x1c>
  1c:   b8 00 00 00 00          mov    $0x0,%eax
  21:   5d                      pop    %rbp
  22:   c3                      ret

Disassembly of section .rodata:

0000000000000000 <.rodata>:
   0:   68 65 6c 6c 6f          push   $0x6f6c6c65
   5:   20 77 6f                and    %dh,0x6f(%rdi)
   8:   72 6c                   jb     76 <main+0x76>
   a:   64 21 2c 20             and    %ebp,%fs:(%rax,%riz,1)
   e:   6d                      insl   (%dx),%es:(%rdi)
   f:   6f                      outsl  %ds:(%rsi),(%dx)
  10:   73 21                   jae    33 <main+0x33>
        ...

Disassembly of section .comment:

0000000000000000 <.comment>:
   0:   00 47 43                add    %al,0x43(%rdi)
   3:   43 3a 20                rex.XB cmp (%r8),%spl
   6:   28 55 62                sub    %dl,0x62(%rbp)
   9:   75 6e                   jne    79 <main+0x79>
   b:   74 75                   je     82 <main+0x82>
   d:   20 31                   and    %dh,(%rcx)
   f:   31 2e                   xor    %ebp,(%rsi)
  11:   33 2e                   xor    (%rsi),%ebp
  13:   30 2d 31 75 62 75       xor    %ch,0x75627531(%rip)
  # 7562754a <main+0x7562754a>
  19:   6e                      outsb  %ds:(%rsi),(%dx)
  1a:   74 75                   je     91 <main+0x91>
  1c:   31 7e 32                xor    %edi,0x32(%rsi)
  1f:   32 2e                   xor    (%rsi),%ch
```

```
  21:   30 34 29                xor    %dh,(%rcx,%rbp,1)
  24:   20 31                   and    %dh,(%rcx)
  26:   31 2e                   xor    %ebp,(%rsi)
  28:   33 2e                   xor    (%rsi),%ebp
  2a:   30 00                   xor    %al,(%rax)

Disassembly of section .note.gnu.property:

0000000000000000 <.note.gnu.property>:
   0:   04 00                   add    $0x0,%al
   2:   00 00                   add    %al,(%rax)
   4:   10 00                   adc    %al,(%rax)
   6:   00 00                   add    %al,(%rax)
   8:   05 00 00 00 47          add    $0x47000000,%eax
   d:   4e 55                   rex.WRX push %rbp
   f:   00 02                   add    %al,(%rdx)
  11:   00 00                   add    %al,(%rax)
  13:   c0 04 00 00             rolb   $0x0,(%rax,%rax,1)
  17:   00 03                   add    %al,(%rbx)
  19:   00 00                   add    %al,(%rax)
  1b:   00 00                   add    %al,(%rax)
  1d:   00 00                   add    %al,(%rax)
        ...

Disassembly of section .eh_frame:

0000000000000000 <.eh_frame>:
   0:   14 00                   adc    $0x0,%al
   2:   00 00                   add    %al,(%rax)
   4:   00 00                   add    %al,(%rax)
   6:   00 00                   add    %al,(%rax)
   8:   01 7a 52                add    %edi,0x52(%rdx)
   b:   00 01                   add    %al,(%rcx)
   d:   78 10                   js     1f <.eh_frame+0x1f>
   f:   01 1b                   add    %ebx,(%rbx)
  11:   0c 07                   or     $0x7,%al
  13:   08 90 01 00 00 1c       or     %dl,0x1c000001(%rax)
  19:   00 00                   add    %al,(%rax)
```

```
  1b:   00 1c 00                add    %bl,(%rax,%rax,1)
  1e:   00 00                   add    %al,(%rax)
  20:   00 00                   add    %al,(%rax)
  22:   00 00                   add    %al,(%rax)
  24:   23 00                   and    (%rax),%eax
  26:   00 00                   add    %al,(%rax)
  28:   00 45 0e                add    %al,0xe(%rbp)
  2b:   10 86 02 43 0d 06       adc    %al,0x60d4302(%rsi)
  31:   5a                      pop    %rdx
  32:   0c 07                   or     $0x7,%al
  34:   08 00                   or     %al,(%rax)
        ...
```

使用 mips 交叉编译和反汇编:

```
mips-linux-gnu-gcc -c helloworld.c -o helloworl3
mips-linux-gnu-objdump -DS helloworld3 > helloworld4
```

反汇编结果 helloworld3:

```
helloworld3:     文件格式 elf32-tradbigmips


Disassembly of section .text:

00000000 <main>:
   0:   27bdffe0        addiu   sp,sp,-32
   4:   afbf001c        sw      ra,28(sp)
   8:   afbe0018        sw      s8,24(sp)
   c:   03a0f025        move    s8,sp
  10:   3c1c0000        lui     gp,0x0
  14:   279c0000        addiu   gp,gp,0
  18:   afbc0010        sw      gp,16(sp)
  1c:   3c020000        lui     v0,0x0
  20:   24440000        addiu   a0,v0,0
  24:   8f820000        lw      v0,0(gp)
  28:   0040c825        move    t9,v0
  2c:   0320f809        jalr    t9
  30:   00000000        nop
  34:   8fdc0010        lw      gp,16(s8)
```

```
  38:   00001025        move    v0,zero
  3c:   03c0e825        move    sp,s8
  40:   8fbf001c        lw      ra,28(sp)
  44:   8fbe0018        lw      s8,24(sp)
  48:   27bd0020        addiu   sp,sp,32
  4c:   03e00008        jr      ra
  50:   00000000        nop
        ...

Disassembly of section .reginfo:

00000000 <.reginfo>:
   0:   f2000014        0xf2000014
        ...

Disassembly of section .MIPS.abiflags:

00000000 <.MIPS.abiflags>:
   0:   00002002        srl     a0,zero,0x0
   4:   01010005        lsa     zero,t0,at,0x1
        ...

Disassembly of section .pdr:

00000000 <.pdr>:
   0:   00000000        nop
   4:   c0000000        ll      zero,0(zero)
   8:   fffffffc        0xfffffffc
        ...
  14:   00000020        add     zero,zero,zero
  18:   0000001e        0x1e
  1c:   0000001f        0x1f

Disassembly of section .rodata:

00000000 <.rodata>:
   0:   68656c6c        0x68656c6c
   4:   6f20776f        0x6f20776f
```

```
   8:   726c6421        0x726c6421
   c:   2c206d6f        sltiu   zero,at,28015
  10:   73210000        madd    t9,at
        ...

Disassembly of section .comment:

00000000 <.comment>:
   0:   00474343        0x474343
   4:   3a202855        xori    zero,s1,0x2855
   8:   62756e74        0x62756e74
   c:   75203130        jalx    480c4c0 <main+0x480c4c0>
  10:   2e332e30        sltiu   s3,s1,11824
  14:   2d317562        sltiu   s1,t1,30050
  18:   756e7475        jalx    5b9d1d4 <main+0x5b9d1d4>
  1c:   31292031        andi    t1,t1,0x2031
  20:   302e332e        andi    t6,at,0x332e
  24:   地址 0x0000000000000024 越界。


Disassembly of section .gnu.attributes:

00000000 <.gnu.attributes>:
   0:   41000000        mftc0   zero,c0_index
   4:   0f676e75        jal     d9db9d4 <main+0xd9db9d4>
   8:   00010000        sll     zero,at,0x0
   c:   00070405        0x70405
```

由表 1可以得知，我们最常用的 objdump -DS [executable file] 这一命令中-DS 表示将所有的 section 反汇编，并将反汇编代码和原码交替显示。

Thinking 1.2

问题 1

结果如下:

```
0:0x0
1:0x80100000
2:0x801024e0
```

表 1: objdump 一些常用参数的含义

parameter	meanings
-d	将代码反汇编，反汇编那些包含指令机器码的 section
-D	和-d 类似，反汇编所有 section，无论该 section 是否包含指令机器码
-S	将源码和反汇编代码交替展示，应该和-d/-D 命令搭配使用
-C	将底层符号名解释为用户级符号名，此外还将去除任何系统预添加的下划线，使得 C++ 内置函数名具有可读性
-l	在反汇编代码中展示源代码的文件名和行号
-j	仅反汇编指定的 section, 可以有多个-j 参数来选择多个 section
-f	对于所有的 objfile 文件，展示全部头文件的总结信息
-F	展示文件按相对于数据区域的地址偏移，无论符号是否被展示
-h	展示所有来自 object file 的 section 头文件的总结信息

```
3:0x801024f8
4:0x80102510
5:0x0
6:0x0
7:0x0
8:0x0
9:0x0
10:0x0
11:0x0
12:0x0
13:0x0
14:0x0
15:0x0
16:0x0
17:0x0
```

问题 2

```
readelf -h ../../target/mos
ELF 头:
  Magic:    7f 45 4c 46 01 01 01 00 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 - System V
  ABI 版本:                          0
  类型:                              EXEC (可执行文件)
  系统架构:                          MIPS R3000
  版本:                              0x1
  入口点地址:               0x801020b0
  程序头起点:          52 (bytes into file)
  Start of section headers:          24144 (bytes into file)
  标志:             0x1001, noreorder, o32, mips1
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         4
  Size of section headers:           40 (bytes)
  Number of section headers:         18
  Section header string table index: 17
```

```
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
```

我们自己编写的 readelf 程序是不能解析 readelf 文件本身。readelf 是 64 位的，而 readelf 解析的文件是 32 位的，字长不匹配所以无法解析自身。也就是机器字长限制我们自己编写的 readelf 程序不能解析其本身。

Thinking 1.3

这是由启动两个阶段执行硬件的不同决定的。

- 第一阶段: 硬件初始化，为下一阶段初始化 RAM，并且装入下一阶段到 RAM；设置堆栈到并跳转到下一个阶段入口；
- 第二阶段: 初始化本阶段所需硬件设备，载入内核和根文件系统，为内核设置启动参数，跳转到内核入口。

第一阶段是 ROM 或者 Flash 发挥作用，第二阶段是 RAM 发挥作用。第一阶段不直接跳转转内核，而是为第二阶段做准备，在第二阶段才从磁盘载入内核和操作系统，设置相应的参数，进而跳转到内核入口，所以加点时的地址不应是内核地址。

2 难点分析

lab1 的内容为内核制作、启动和 printf。我在 elf 文件格式理解、系统内核位置设置、以及 printk 函数的书写方面均感觉困难。

elf 文件格式理解

我们一定要熟记这张 ELF 文件结构框架图:

在实验中，一开始我没有分清指导书中 section 和 segment 的区别，造成了很大的麻烦。两者的区别是：

- 段的信息需要在运行时刻使用；
- 节的信息需要在程序编译和链接的时候使用。

在获取节头表的时候，也有一些要注意的地方。

图 1: elf 文件基本格式

```
// Get the address of the section table,
// the number of section headers and the size of a
// section header.
const void *sh_table;
Elf32_Half sh_entry_count;
Elf32_Half sh_entry_size;
/* Exercise 1.1: Your code here. (1/2) */
//sh is the core
sh_table = binary + ehdr->e_shoff;
sh_entry_count = ehdr->e_shnum;
sh_entry_size = ehdr->e_shentsize;
// For each section header, output its index and the section address.
// The index should start from 0.
```

节头表的地址等于文件头地址加上节头表所在处与此文件头的偏移，这一点看似简单，但若不深入了解 elf 的文件结构并且记住图 1，就会遇到麻烦。

内核位置设置

一开始，我虽然查找到了正确的内核起始地址，但是在编译运行内核的时候总是产生 **kernel panic** 的错误，让我百思不得其解。通过阅读指导书，我了解到，“*”是一个通配符，匹配所有的相应的节。只有使用形如.bss:*(.bss) 的方式，才能让输入文件中的节放到输出文件中的节中，才能完成内核的正确设置。这里的学习让我印象深刻。

printk 函数的书写

printk 函数的书写，最重要的是阅读指导书中的关于函数的附录，同时仔细阅读函数中已经给出的代码和提示，理解函数的核心逻辑：

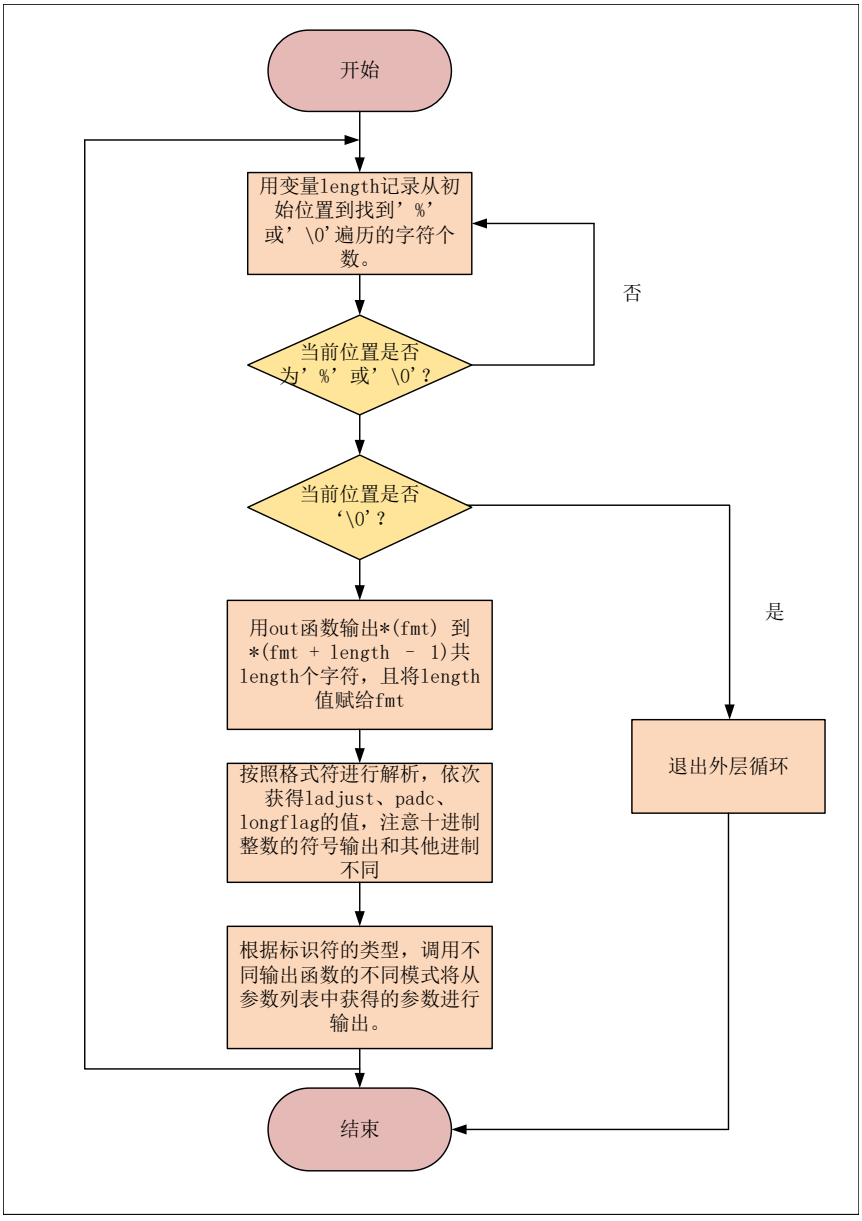

图 2: printk 函数工作流程图

之后问题就迎刃而解。

3 实验体会

本次作业的任务量不大，但是思维含量极高，可以说为之后的实验奠定了最重要的基础。我不到 3 小时就完成了课下实验，但是深入理解其中的实验原理却花费了我十余个小时的时间。特别是内核启动部分，初看幽微难明，但是随着我们熟练使用各种工具从不同方面剖析它们的结构，不断总结它们的特点和性质，一个清晰的内核启动过程就逐步复现在我们眼前。让我们以这次小小的内核实验作为我们整个操作系统伟大征程的起点吧！