

# 操作系统lab1实验报告

姓名：高远 学号：19374242 班级：202113

## 实验思考题

### Thinking 1.1:

objdump中的参数-D：反汇编所有section，-S:显示指定section的完整内容，-DS为查看源码和汇编码的对照列表。

```
gy20001218@DESKTOP-IUQKTPH: ~  
Hello.o:      file format elf64-x86-64  
  
Disassembly of section .text:  
0000000000000000 <main>:  
0:  f3 0f 1e fa      endbr64  
4:  55               push  %rbp  
5:  48 89 e5         mov   %rsp,%rbp  
8:  48 8d 3d 00 00 00 00  lea   0x0(%rip),%rdi    # f <main+0xf>  
f:  e8 00 00 00 00     callq 14 <main+0x14>  
14: b8 00 00 00 00     mov   $0x0,%eax  
19: 5d               pop   %rbp  
1a: c3               retq  
  
Disassembly of section .rodata:  
0000000000000000 <.rodata>:  
0:  48               rex.W  
1:  65 6c           gs insb (%dx),%es:(%rdi)  
3:  6c             insb  (%dx),%es:(%rdi)  
4:  6f             outsl %ds:(%rsi),(%dx)  
5:  20 57 6f        and   %dl,0x6f(%rdi)  
8:  72 6e           jb    76 <main+0x76>  
a:  64             fs  
...  
  
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 39          and   %bh,(%rcx)  
f:  2e 34 2e        cs xor $0x2e,%al  
12: 30 2d 31 75 62 75  xor   %ch,0x75627531(%rip)    # 75627549 <main+0x75627549>  
18: 6e             outsb %ds:(%rsi),(%dx)  
19: 74 75           je    90 <main+0x90>  
1b: 31 7e 32        xor   %edi,0x32(%rsi)  
1e: 30 2e          xor   %ch,(%rsi)  
20: 30 34 29        xor   %dh,(%rcx,%rbp,1)  
23: 20 39          and   %bh,(%rcx)  
25: 2e 34 2e        cs xor $0x2e,%al  
28: 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)  
"b.txt" 92L, 3552C
```

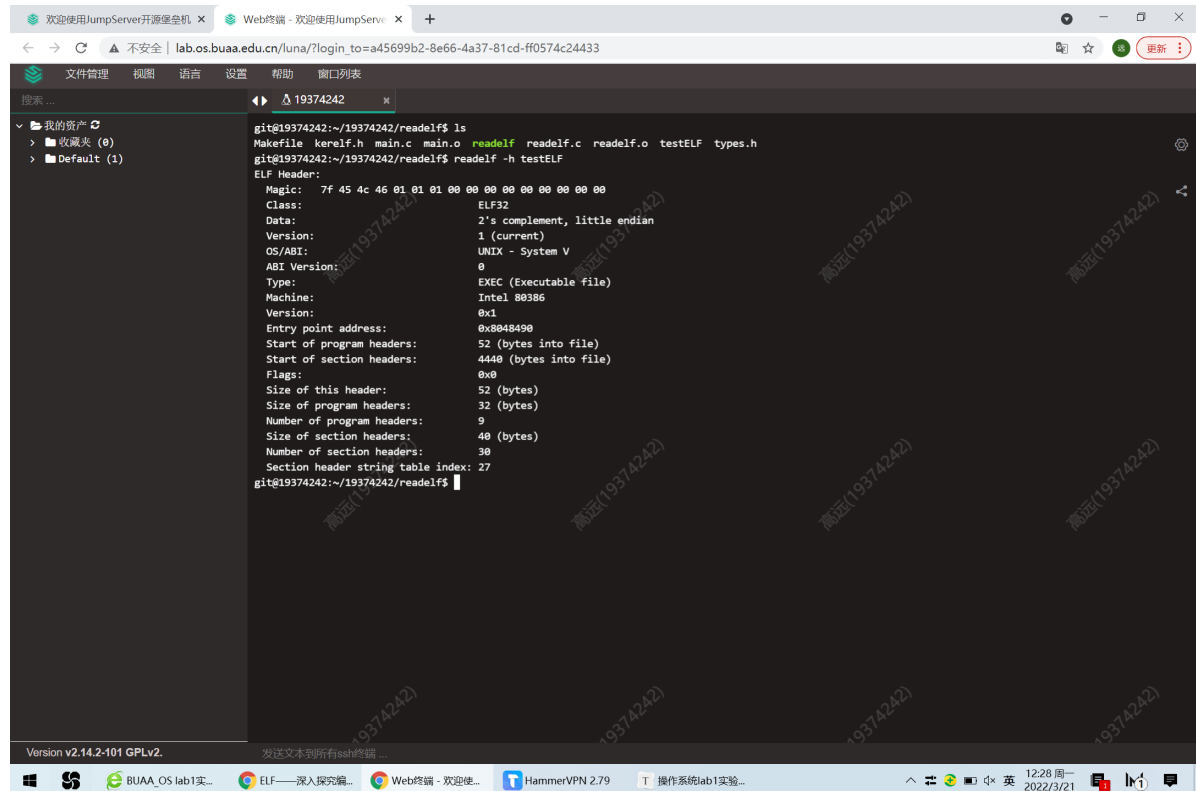
```
gy20001218@DESKTOP-IUQKTPH: ~  
hello: file format elf64-x86-64  
  
Disassembly of section .interp:  
0000000000000318 <.interp>:  
318: 2f (bad)  
319: 6c inab (%rdi), %esi, (%rdi)  
31a: 69 62 36 34 2f 6c 64 imul $0x46c2f34, 0x36(%rdi), %esp  
321: 2d 6c 69 6e 75 sub $0x756e696e, %eax  
326: 78 2d js 355 <_init-0xcab>  
328: 78 38 js 362 <_init-0xc9e>  
32a: 36 2d 36 34 2e 73 ss sub $0x732e3436, %eax  
330: 6f outsl %eax, (%rdi), (%rdi)  
331: 2e 32 00 xor %cs, (%rax), %al  
  
Disassembly of section .note.gnu.property:  
0000000000000338 <.note.gnu.property>:  
338: 04 00 add $0x0, %al  
33a: 00 00 add %al, (%rax)  
33c: 10 00 adc %al, (%rax)  
33e: 00 00 add %al, (%rax)  
340: 05 00 00 00 47 add $0x47000000, %eax  
345: 4e 55 rex.WRX push %rbp  
347: 00 02 add %al, (%rdi)  
349: 00 00 add %al, (%rax)  
34b: c0 04 00 00 rolb $0x0, (%rax, %rax, 1)  
34f: 00 03 add %al, (%rbx)  
351: 00 00 add %al, (%rax)  
353: 00 00 add %al, (%rax)  
355: 00 00 add %al, (%rax)  
...  
  
Disassembly of section .note.gnu.build-id:  
0000000000000358 <.note.gnu.build-id>:  
358: 04 00 add $0x0, %al  
35a: 00 00 add %al, (%rax)  
35c: 14 00 adc $0x0, %al  
35e: 00 00 add %al, (%rax)  
360: 03 00 add (%rax), %eax  
362: 00 00 add %al, (%rax)  
364: 47 rex.RXB  
365: 4e 55 rex.WRX push %rbp  
367: 00 68 24 add %ch, 0x24(%rax)  
36a: 42 b2 fb rex.X mov $0xfb, %dl  
36d: c7 (bad)  
36e: 1e (bad)  
36f: d4 (bad)  
370: 08 5b d3 or %b1, -0x2d(%rbx)  
373: 99 cltd  
374: d1 3f (%rdi)  
376: c0 c6 95 rol $0x95, %dh
```

## Thinking 1.2:

用原有的readelf解析内核文件

```
lab1——内核、Bo... ELF——深入探究... Web终端——欢迎使... HammerVPN 2.79  
12:24 周一 2022/3/21  
Version v2.14.2-101 GPLv2. 发送文本到所有ssh终端...  
文件管理 视图 语言 设置 帮助 窗口列表  
搜索... 19374242 x  
我的资产  
收藏夹 (0)  
Default (1)  
git@19374242:~/19374242$ cd gxemul/  
git@19374242:~/19374242/gxemul$ ls  
elfinfo r3000 r3000_test test vmlinux  
git@19374242:~/19374242/gxemul$ readelf -h vmlinux  
ELF Header:  
Magic: 7f 45 4c 46 01 02 01 00 00 00 00 00 00 00 00 00  
Class: ELF32  
Data: 2's complement, big endian  
Version: 1 (current)  
OS/ABI: UNIX - System V  
ABI Version: 0  
Type: EXEC (Executable file)  
Machine: MIPS R3000  
Version: 0x1  
Entry point address: 0x0  
Start of program headers: 52 (bytes into file)  
Start of section headers: 36652 (bytes into file)  
Flags: 0x1001, noreorder, o32, mips1  
Size of this header: 52 (bytes)  
Size of program headers: 32 (bytes)  
Number of program headers: 2  
Size of section headers: 40 (bytes)  
Number of section headers: 14  
Section header string table index: 11  
git@19374242:~/19374242/gxemul$
```

使用原有的readelf解析testELF文件结果如下：



```
git@19374242:~/19374242/readelf$ ls
Makefile  kerelf.h  main.c  main.o  readelf  readelf.c  readelf.o  testELF  types.h
git@19374242:~/19374242/readelf$ readelf -h testELF
ELF Header:
  Magic:   7f 45 4c 46 01 01 00 00 00 00 00 00 00 00 00 00
  Class:       ELF32
  Data:       2's complement, little endian
  Version:     1 (current)
  OS/ABI:      UNIX - System V
  ABI Version: 0
  Type:        EXEC (Executable file)
  Machine:     Intel 80386
  Version:     0x1
  Entry point address: 0x8048490
  Start of program headers: 52 (bytes into file)
  Start of section headers: 4440 (bytes into file)
  Flags:       0x0
  Size of this header: 52 (bytes)
  Size of program headers: 32 (bytes)
  Number of program headers: 9
  Size of section headers: 40 (bytes)
  Number of section headers: 30
  Section header string table index: 27
git@19374242:~/19374242/readelf$
```

注意到第六个魔数不同，01表示小头编码，而02表示大头编码。因为vmlinux文件是大头编码，而我们的readelf文件目前只能解析小端存储的文件，因此大头的数据传入程序中时会显示错误。

## Thinking 1.3:

mips体系结构上电时，在启动入口地址硬件初始化，并为加载内核准备RAM空间，然后将内核的代码复制到RAM空间，并且设置堆栈，最后跳转到内核的入口函数，从而正确跳转到内核入口。

## Thinking 1.4:

为了保证页面不冲突，加载程序时尽量避免存在共享页面和冲突页面，如果上一个程序的加载页为v 那么下一个程序的加载页则为v+1。

## Thinking 1.5:

内核的入口在0x80000000，main函数在0x80010000。内核进入main函数的方法是在start.S内部使用跳转指令 `jal` 跳转到指定的函数地址。跨文件调用函数通过跳转指令来调用，同时在跳转之前需要将数据存入栈中。

## Thinking 1.6:

```
// 禁止全局中断
mtc0 zero, CP0_STATUS //将 STATUS 清零
mfc0 t0, CP0_CONFIG //将 Config 数取出
and t0, ~0x7 //将 t0 中的数低三位清零
ori t0, 0x2 //t0 中的数第二位置一
mtc0 t0, CP0_CONFIG //将 t0 赋值到 Config
// 后三位为 K0可写的域，用来决定固定的ksech 0区是否经过高速缓存。如果要经过高速缓存，其确切行为如何。
```

# 实验难点图示

1.补全readelf.c代码题花了我很长的时间，主要是对ehdr和shdr指针没有充分了解，不明白他们所指代的实体是什么，尤其是将ehdr指针强制转换为shdr指针的那一步花了我很多的时间去思考。

、

```
sh_entry_count=ehdr->e_shnum;
sh_entry_size=ehdr->e_shentsize;
ptr_sh_table=binary+ehdr->e_shoff;
for(Nr=0;Nr<sh_entry_count;Nr++)
{
    shdr=( Elf32_Shdr *) (ptr_sh_table+Nr*(ehdr->e_shentsize));
    printf("%d:0x%x\n",Nr,shdr->sh_addr);
}
```

、

2.这短短两行代码也花了我很长的时间去思考，一开始记错了lui的作用，一直把0x8040写成0x80400000，后来才记起来lui的功能，这里的错误我也debug了很久

```
lui    sp,0x8040
jal    main
```

3.最后就是print函数，一开始一直想不明白这个代码块的作用是什么，后来才知道是输出普通的字符串，其次我也没有充分了解OUTPUT各个参数代表了什么，我一直在想它第一个参数arg该填什么，差不多debug了四个小时才成功。

```
{
    /* scan for the next '%' */
    char *cur=fmt;
    while(1)
    {
        if(*cur=='%') break;
        if(*cur=='\0') break;
        cur++;
    }
    /* flush the string found so far */
    OUTPUT(arg,fmt,cur-fmt);
    fmt=cur;
    /* are we hitting the end? */
    if(*fmt=='\0') break;
}
```

## 体会与感想

本次实验差不多花了我两天的时间才完成，尤其是在readelf.c和print.c两个函数补充里花费了我大量的时间，一方面，是因为这两题确实有着一定的难度，另一方面也是因为我急于求成，没有彻底理解便想着做题，导致浪费了很多时间。不过相应的，在做完lab1后，我的收获也是巨大的，我初步了解了计算机启动时的一系列操作，了解了elf文件的种类和实现方式，明白了print函数到底是如何实现并链接至可执行文件的，可以说，这次试验让我痛并快乐着，使我受益匪浅。

# 指导书反馈

---

希望指导书可以对各个操作有一个总的叙述，帮助我们更好地理解每步操作的意义和扮演的角色。

## 残留难点

---

在指导书mips汇编那一节，我对fp指针和sp指针各自的作用还不是非常的了解，对GXemul的作用也不是非常的清楚。