

操作系统 Lab1实验报告

实验思考题

Thinking 1.1

对于 `objdump` 指令：

- `-D` 指令是指反汇编所有section
- `-S` 指令是指在反汇编的同时输出源代码

```
git@20373785:~/20373785$ objdump -H
Usage: objdump <option(s)> <file(s)>
Display information from object <file(s)>.
At least one of the following switches must be given:
-a, --archive-headers    Display archive header information
-f, --file-headers       Display the contents of the overall file header
-p, --private-headers    Display object format specific file header contents
-P, --private=OPT,OPT... Display object format specific contents
-h, --[section-]headers  Display the contents of the section headers
-x, --all-headers        Display the contents of all headers
-d, --disassemble        Display assembler contents of executable sections
-D, --disassemble-all   Display assembler contents of all sections
    --disassemble=<sym>  Display assembler contents from <sym>
-S, --source              Intermix source code with disassembly
    --source-comment[=<txt>] Prefix lines of source code with <txt>
-s, --full-contents       Display the full contents of all sections requested
-g, --debugging           Display debug information in object file
-e, --debugging-tags      Display debug information using ctags style
-G, --stabs               Display (in raw form) any STABS info in the file
-W[!LIaprmfFsoRtUuTgAckK] or
    --dwarf[=rawline,=decodedline,=info,=abbrev,=pubnames,=aranges,=macro,=frames,
    =frames-interp,=str,=loc,=Ranges,=pubtypes,
    =gdb_index,=trace_info,=trace_abbrev,=trace_aranges,
    =addr,=cu_index,=links,=follow-links]
    Display DWARF info in the file
--ctf=SECTION             Display CTF info from SECTION
-t, --syms                Display the contents of the symbol table(s)
-T, --dynamic-syms        Display the contents of the dynamic symbol table
-r, --reloc               Display the relocation entries in the file
-R, --dynamic-reloc       Display the dynamic relocation entries in the file
@<file>                   Read options from <file>
-v, --version             Display this program's version number
-i, --info                List object formats and architectures supported
-H, --help                Display this information
```

对于以下代码：

```
1  /* hello.c */
2  int main() {
3      int a;
4      int b = 1;
5      int c = 0;
6      b = 2;
7      c = myhello(a, b);
8      return 0;
```

```

9   }
10
11  /* myhello.c */
12  int myhello(int x, int y) {
13      return x + y;
14  }

```

通过 `Makefile` 使用 MIPS 交叉编译器进行编译和反汇编：

```

1  /* Makefile */
2  CROSS_COMPILE := /OSLAB/compiler/usr/bin/mips_4KC-
3  CC             := $(CROSS_COMPILE)gcc
4  LD             := $(CROSS_COMPILE)ld
5  OBJDUMP        := $(CROSS_COMPILE)objdump
6  TARGET         := hello.c
7  FUNC           := myhello.c
8
9  all:
10     $(CC) -c $(FUNC) -o myhello_c.o
11     $(CC) -E $(TARGET) > hello_E.txt
12     $(CC) -c $(TARGET) -o hello_c.o
13     $(OBJDUMP) -DS hello_c.o > dump_c.txt
14     $(LD) -o hello_exe -N hello_c.o myhello_c.o
15     $(OBJDUMP) -DS hello_exe > dump_exe.txt

```

- 对 `hello.c` 文件的预处理结果如下：

```

1  # 1 "hello.c"
2  # 1 "<built-in>"
3  # 1 "<command line>"
4  # 1 "hello.c"
5  int main() {
6      int a;
7      int b = 1;
8      int c = 0;
9      b = 2;
10     c = myhello(a, b);
11     return 0;
12 }

```

- 对 `hello.c` 只编译而不链接，产生可重定位目标文件 `hello.o` 后反汇编（部分）结果如下：

```

1  /* dump_c.txt */
2  hello_c.o:      file format elf32-tradbigmips
3
4  Disassembly of section .text:
5
6  00000000 <main>:
7      0:  3c1c0000      lui      gp,0x0
8      4:  279c0000      addiu    gp,gp,0
9      8:  0399e021      addu     gp,gp,t9
10     c:  27bdf fd0      addiu    sp,sp,-48
11    10:  afbf002c      sw       ra,44(sp)
12    14:  afbe0028      sw       s8,40(sp)
13    18:  03a0f021      move     s8,sp

```

14	1c:	afbc0010	sw	gp,16(sp)
15	20:	24020001	li	v0,1
16	24:	afc2001c	sw	v0,28(s8)
17	28:	afc00018	sw	zero,24(s8)
18	2c:	24020002	li	v0,2
19	30:	afc2001c	sw	v0,28(s8)
20	34:	8fc40020	lw	a0,32(s8)
21	38:	8fc5001c	lw	a1,28(s8)
22	3c:	8f990000	lw	t9,0(gp)
23	40:	0320f809	jalr	t9
24	44:	00000000	nop	
25	48:	8fdc0010	lw	gp,16(s8)
26	4c:	afc20018	sw	v0,24(s8)
27	50:	00001021	move	v0,zero
28	54:	03c0e821	move	sp,s8
29	58:	8fbf002c	lw	ra,44(sp)
30	5c:	8fbe0028	lw	s8,40(sp)
31	60:	27bd0030	addiu	sp,sp,48
32	64:	03e00008	jr	ra
33	68:	00000000	nop	
34	6c:	00000000	nop	

- 链接 `hello.o` 和 `myhello.o` 生成可执行文件 `hello_exe` 后反汇编结果（部分）如下：

```

1  /* dump_exe.txt */
2  hello_exe:      file format elf32-tradbigmips
3
4  Disassembly of section .text:
5
6  004000b0 <main>:
7      4000b0:      3c1c0001      lui      gp,0x1
8      4000b4:      279c80a0      addiu    gp,gp,-32608
9      4000b8:      0399e021      addu     gp,gp,t9
10     4000bc:      27bdffd0      addiu    sp,sp,-48
11     4000c0:      afbf002c      sw       ra,44(sp)
12     4000c4:      afbe0028      sw       s8,40(sp)
13     4000c8:      03a0f021      move     s8,sp
14     4000cc:      afbc0010      sw       gp,16(sp)
15     4000d0:      24020001      li       v0,1
16     4000d4:      afc2001c      sw       v0,28(s8)
17     4000d8:      afc00018      sw       zero,24(s8)
18     4000dc:      24020002      li       v0,2
19     4000e0:      afc2001c      sw       v0,28(s8)
20     4000e4:      8fc40020      lw       a0,32(s8)
21     4000e8:      8fc5001c      lw       a1,28(s8)
22     4000ec:      8f99802c      lw       t9,-32724(gp)
23     4000f0:      0320f809      jalr     t9
24     4000f4:      00000000      nop
25     4000f8:      8fdc0010      lw       gp,16(s8)
26     4000fc:      afc20018      sw       v0,24(s8)
27     400100:      00001021      move     v0,zero
28     400104:      03c0e821      move     sp,s8
29     400108:      8fbf002c      lw       ra,44(sp)
30     40010c:      8fbe0028      lw       s8,40(sp)
31     400110:      27bd0030      addiu    sp,sp,48
32     400114:      03e00008      jr       ra
33     400118:      00000000      nop

```

```

34 40011c: 00000000 nop
35
36 00400120 <myhello>:
37 400120: 27bdfff8 addiu sp,sp,-8
38 400124: afbe0000 sw s8,0(sp)
39 400128: 03a0f021 move s8,sp
40 40012c: afc40008 sw a0,8(s8)
41 400130: afc5000c sw a1,12(s8)
42 400134: 8fc30008 lw v1,8(s8)
43 400138: 8fc2000c lw v0,12(s8)
44 40013c: 00621021 addu v0,v1,v0
45 400140: 03c0e821 move sp,s8
46 400144: 8fbe0000 lw s8,0(sp)
47 400148: 27bd0008 addiu sp,sp,8
48 40014c: 03e00008 jr ra
49 400150: 00000000 nop
50 ...

```

- 对两次反汇编结果中 `main` 函数部分进行比较可以发现，在生成可执行文件后，编译器设置了代码存储的地址，同时也对于调用 `myhello()` 函数的地址进行了设置，通过全局指针进行传递，而这样的结果也与指导书中的示例类似。

```

1  /* dump_c.txt */
2  3c: 8f990000 lw t9,0(gp)
3  40: 0320f809 jalr t9
4  /* dump_exe.txt */
5  4000ec: 8f99802c lw t9,-32724(gp)
6  4000f0: 0320f809 jalr t9

```

Thinking 1.2

- 解析 `vmlinux` :


```

git@20373785:~/20373785/gxemul$ readelf -h vmlinux
ELF Header:
  Magic:   7f 45 4c 46 01 02 01 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@20373785:~/20373785/gxemul$

```

- 解析 testELF :

```

git@20373785:~/20373785/readelf$ readelf -h testELF
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 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@20373785:~/20373785/readelf$

```

- 由于内核文件 `vmlinux` 是大端存储的，而我们的 `readelf` 文件只能解析小端存储的文件，因此无法解析内核文件 `vmlinux`，会提示内存溢出。

```
git@20373785:~/20373785/readelf$ ./readelf ../gxemul/vmlinux
Segmentation fault (core dumped)
git@20373785:~/20373785/readelf$
```

Thinking 1.3

MIPS体系结构上电时的启动入口地址 `0xBFC00000` 位于 `kseg1` 内，通过将高3位清零的方法映射到物理地址 `0x1FC00000`，即CPU从 `0x1FC00000` 开始取第一条指令，因此可以将 `bootloader` 的代码放在这里，由 `bootloader` 完成硬件启动和初始化后再将内核镜像文件加载到内存中，也就可以保证内核入口能够被正确跳转到。

Thinking 1.4

在加载程序段时可以将 `.bss` 段放在整个段的末尾，由于 `.bss` 段存放的是程序中未初始化的全局变量，因此可以在这部分内存中填入0，这样可以在一定程度上避免后面程序段中的数据与当前程序产生冲突导致当前程序的数据丢失。

Thinking 1.5

```

/*
o    4G -----> +-----0x100000000
o                  |      ...      | kseg3
o                  +-----0xe000 0000
o                  |      ...      | kseg2
o                  +-----0xc000 0000
o                  | Interrupts & Exception | kseg1
o                  +-----0xa000 0000
o                  | Invalid memory      | /\
o                  +-----Physics Memory Max
o                  |      ...      | kseg0
o VPT, KSTACKTOP-----> +-----0x8040 0000-----end
o                  | Kernel Stack      | KSTKSIZE      /\
o                  +-----
o                  | Kernel Text      | PDMAP
o KERNBASE -----> +-----0x8001 0000 |
o                  | Interrupts & Exception | \\/
o ULIM -----> +-----0x8000 0000-----
o                  | User VPT      | PDMAP      /\
o UVPT -----> +-----0x7fc0 0000 |
o                  | PAGES      | PDMAP
o UPAGES -----> +-----0x7f80 0000 |
o                  | ENVS      | PDMAP
o UTOP, UENVS -----> +-----0x7f40 0000 |
o UXSTACKTOP -/ | user exception stack | BY2PG
o                  +-----0x7f3f f000 |
o                  | Invalid memory      | BY2PG
o USTACKTOP -----> +-----0x7f3f e000 |
o                  | normal user stack    | BY2PG
o                  +-----0x7f3f d000 |
a                  |
a                  +-----
a                  |
a                  |
a                  |
a                  |
a UTEXT -----> +-----
o                  | 2 * PDMAP      | \\/
a    0 -----> +-----
o
*/

```

- 内核入口位于 0x80000000 处。
- main 函数位于 0x80010000 处。
- 在内核的入口函数中可以设置需要跳转到的地址，通过 jal 指令使其跳转到 main 函数地址处。
- 在进行编译链接的时候，编译器会将跨文件函数对应的地址加载到跳转指令处，在进行跨文件调用时，函数首先将需要被保存的变量以及返回地址存入栈中，再通过跳转指令跳转到被调用函数的地址，函数调用结束后将返回地址出栈，再跳转回原函数中，最后将相关变量从栈中恢复出来。

Thinking 1.6

```
1  /* Disable interrupts */
2  mtc0 zero, CP0_STATUS #将0写入协处理器0的12号状态寄存器，从而关闭中断
3  .....
4  /* disable kernel mode cache */
5  mfc0 t0, CP0_CONFIG #将协处理器0的16号参数寄存器中的内容存入t0寄存器
6  and t0, ~0x7 #将t0寄存器中的值的后三位清零
7  ori t0, 0x2 #将t0寄存器中的值的低位第2位置一
8  mtc0 t0, CP0_CONFIG #将t0中的值写入协处理器0的16号参数寄存器
9  #上述代码禁用了内核cache
```

实验难点

Exercise1.2

在刚开始完成本题时，没有充分理解基本概念和代码的意图，因此绕了很多弯路。反复阅读指导书之后，逐渐理清了 `program header`、`section header`、`segments`、`sections` 等基本概念以及 `hereIf.h` 中的结构体的变量定义，代码的填充就变得容易了许多：

$$shdr = (e_shoff + binary) + e_shentsize * i$$

在理解了代码原理之后也使得完成 `lab1-1-exam` 时比较顺畅。

printf 的实现

在一开始实现 `lp_Print()` 函数时没有仔细阅读指导书，而是依赖代码中的注释来填写，导致格式符的判断顺序一直出错，在回看指导书之后根据格式符原型才明确了格式符的判断顺序。

$$\%[flags][width][.precision][length]specifier$$

- flag:
 - "-"
 - "0"
- width: "数字"
- .precision: ".数字"
- length: "l"
- specifier

由此也我不免产生了一些疑惑，是否在 `lp_Print()` 函数代码中的注释的顺序有些许问题？

具体描述发布在了mooc平台的讨论区：[讨论 | 关于 Exercise1.5 踩过的坑](#)

体会与感想

lab1课下内容总计用时10个小时左右，期间偶尔会有卡住的地方，但最后也都还算顺利地解决了。但目前整体看来，自己对于将要实现的操作系统理解仍然不是很深入，尚且停留在跟随着指导书上的任务说到这里学到哪里，因此掌握内容的广度和深度都还很局限。

虽然指导书中提供了许多参考资料，但大多篇幅比较长，迫于课内外的压力没有时间从头到尾完整看下来，只能囫圇吞枣在一些内容上扫几眼，个人感觉效果也并不是很好。除此之外，对于当前实现的内核的整个代码体系，我的理解也还不够完全，对于部分注释较少甚至没有注释的代码，我并不能很完整的理解其作用以及相互之间的调用关系。

希望随着课程的深入，我能够尽快对整个代码体系有一个全面而深入的认识，从而提升自己在之后实验中的完成效率和质量。