在Lab1我们首先填写readelf文件，是如下：

int readelf(u\_char \*binary, int size)

{

Elf32\_Ehdr \*ehdr = (Elf32\_Ehdr \*)binary;

int Nr;

Elf32\_Shdr \*shdr = NULL;

u\_char \*ptr\_sh\_table = NULL;

Elf32\_Half sh\_entry\_count;

Elf32\_Half sh\_entry\_size;

// check whether `binary` is a ELF file.

if (size < 4 || !is\_elf\_format(binary)) {

printf("not a standard elf format\n");

return 0;

}

// get section table addr, section header number and section header size.

ptr\_sh\_table=binary+ehdr->e\_shoff; //make table

sh\_entry\_count = ehdr->e\_shnum; //section header number

sh\_entry\_size = ehdr->e\_shentsize; //section header size

// for each section header, output section number and section addr.

Nr=0;

while(sh\_entry\_count--){

shdr = (Elf32\_Shdr \*)ptr\_sh\_table;

printf("%d:0x%x\n", Nr, shdr->sh\_addr);

ptr\_sh\_table += sh\_entry\_size;

Nr++;

}

return 0;

}

然后继续做填写scse0\_3.lds

SECTIONS

{

/\*To do:

fill in the correct address of the key section

such as text data bss ...

\*/

. = 0x80000080;

.except\_vec3: {

\*(.text.exc\_vec3)

}

. = 0x80010000;

.text : {

\*(.text)

}

.data :{

\*(.data)

}

.bss : {

\*(.bss)

}

. = 0x80400000;

end = . ;

}

然后我们也补充start.S代码

#include <asm/regdef.h>

#include <asm/cp0regdef.h>

#include <asm/asm.h>

.data

.section .data.stk

KERNEL\_STACK:

.space 0x8000

.globl mCONTEXT

mCONTEXT:

.word 0

.text

LEAF(\_start) /\*LEAF is defined in asm.h and LEAF functions don't call other functions\*/

.set mips2 /\*.set is used to instruct how the assembler works and control the order of instructions \*/

.set reorder

/\* Disable interrupts \*/

mtc0 zero, CP0\_STATUS

/\* Disable watch exception. \*/

mtc0 zero, CP0\_WATCHLO

mtc0 zero, CP0\_WATCHHI

/\* disable kernel mode cache \*/

mfc0 t0, CP0\_CONFIG

and t0, ~0x7

ori t0, 0x2

mtc0 t0, CP0\_CONFIG

/\*To do:

set up stack

you can reference the memory layout in the include/mmu.h

\*/

li sp, 0x80400000

li t0, 0x80400000

sw t0, mCONTEXT

/\*jump to main\*/

jal main

loop:

j loop

nop

END(\_start)

然后最后部分我们是补充printf代码实现字符串打印

for(;;) {

{

/\* scan for the next '%' \*/

length = 0;

while(\*fmt != '%' && \*fmt != '\0'){

buf[length++] = \*(fmt++);

if(length == LP\_MAX\_BUF){

/\*flush when full\*/

OUTPUT(arg, buf, length);

length = 0;

}

}

/\* flush the string found so far \*/

OUTPUT(arg, buf, length);

/\* are we hitting the end? \*/

if(\*fmt == '\0')

break;

}

/\* we found a '%' \*/

fmt++;

/\* check for long \*/

longFlag = 0;

if (\*fmt == 'l') {

fmt++;

longFlag = 1;

}

/\* check for other prefixes \*/

ladjust = 0;

padc = 0;

while(\*fmt < '1' && \*fmt > '9') {

if (\*fmt == '-') {

ladjust = 1;

fmt++;

} else if (\*fmt == '0' || \*fmt == ' ')

padc = \*(fmt++);

else

break;

}

width = 0;

while (IsDigit(\*fmt)) {

width = width \* 10 + Ctod(\*(fmt++));

}

if (\*fmt == '.') {

prec = Ctod(\*(++fmt));

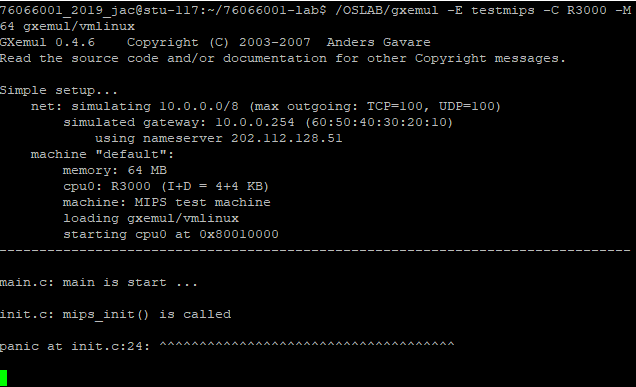
while (IsDigit(\*fmt)) {

prec = prec \* 10 + Ctod(\*(fmt++));

}

}

补充了lp\_print之后我们的实验就结束了，然后使用这个指令 **/OSLAB/gxemul –E testmips –C R3000 –M 64 vmlinux 执行并且检查结果是否跟实验正确的结果相等。在这个lab1我系统能正确的输出实验的要求。结果如下：**



也许你会发现我们的readelf程序是不能解析之前生成的内核文件(内核文件是可执行文件)的，而我们之后将要介绍的工具readelf则可以解析，这是为什么呢？(提示：尝试使用readelf -h，观察不同)

76066001\_2018\_jac@ubuntu:~/76066001-lab/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: 0x80010000

Start of program headers: 52 (bytes into file)

Start of section headers: 37164 (bytes into file)

Flags: 0x50001001, noreorder, o32, mips32

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

76066001\_2018\_jac@ubuntu:~/76066001-lab/readelf$ readelf -h testELF

ELF Header:

Magic: 7f 45 4c 46 01 01 01 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

这因为这个工具(readelf)和objdump命令提供的功能类似，但是它显示的信息更为具体，并且它不依赖BFD库(BFD库是一个GNU项目，它的目标就是希望通过一种统一的接口来处理不同的目标文件），如果使用我们readelf程序只能打印出来他的section header的信息，而且我们程序是简单的程序。但是readelf工具是系统提供的，所以它可以解析我们的内核文件（可执行的文件）。又因为内核文件的machine是MIPS R3000,但我们readelf写的程序只能解析intel 80386的machine,于是我们程序无法解析系统的内核文件。由他的magic number格式也不一样。

**main 函数在什么地方？我们又是怎么跨文件调用函数的呢？**

Main函数地址是在0x400148，如果要跨文件调用函数我们可以使用jal指令然后跳转到另外个想调用函数的栈区地址。