**lab1实验报告**

一、实验思考题

Exercise 1.1

请修改include.mk文件，使交叉编译器的路径正确。之后执行make指令，如果配置一切正确，则会在gxemul目录下生成vmlinux的内核文件。

A:修改后的include.mk文件：

# Common includes in Makefile

#

# Copyright (C) 2007 Beihang University

# Written By Zhu Like ( zlike@cse.buaa.edu.cn )

CROSS\_COMPILE := /OSLAB/compiler/usr/bin/mips\_4KC-

CC := $(CROSS\_COMPILE)gcc

CFLAGS := -O -G 0 -mno-abicalls -fno-builtin -Wa,-xgot -Wall -fPIC

LD := $(CROSS\_COMPILE)ld

Exercise 1.2

阅读./readelf 文件夹中 kerelf.h、readelf.c 以及 main.c 三个文件中的 代码，并完成 readelf.c 中缺少的代码，readelf 函数需要输出 elf 文件的所有 section header 的序号和地址信息，对每个 section header，输出格式为:”%d:0x%x\n”，两 个标识符分别代表序号和地址。

A:readelf.c的补充部分代码如下：

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

ptr\_sh\_table = &binary[ehdr->e\_shoff];

sh\_entry\_count = ehdr->e\_shnum;

sh\_entry\_size = ehdr->e\_shentsize;

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

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

int secnum = 0;

for (secnum = 0; secnum < sh\_entry\_count; secnum++){

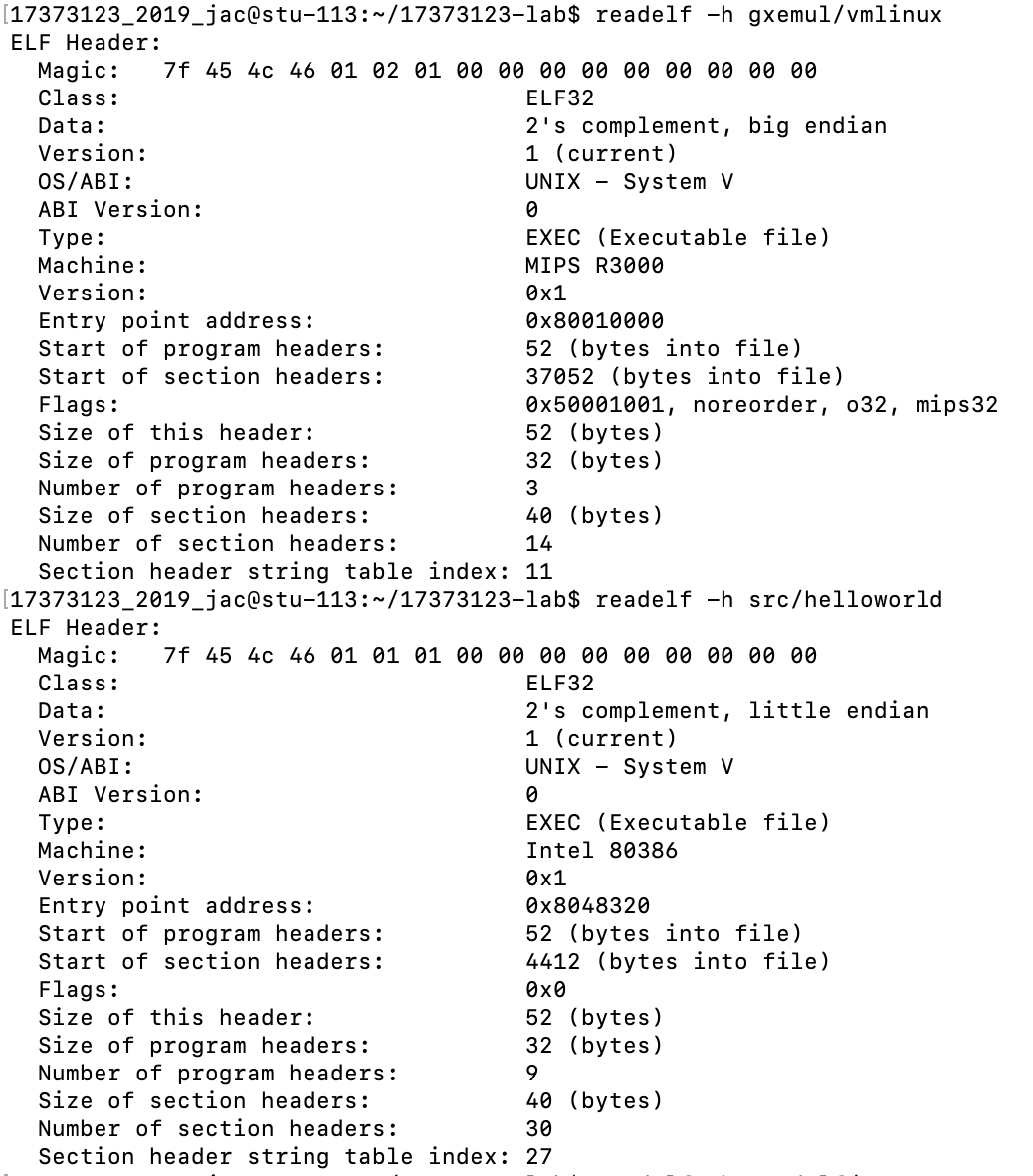
printf("%d:0x%x\n",secnum,shdr->sh\_addr);

shdr++;

}

Thinking 1.1

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)

A: 使用readelf -h的输出结果：观察发现，二者的数据存储大小端不一样，因此无法正常解析。

Exercise 1.3 填写 tools/scse0\_3.lds 中空缺的部分，将内核调整到正确的位置上。

A: . = 0x80010000;

.text : { \*(.text) }

. = 0x08000000;

.data : { \*(.data) }

.bss : { \*(.bss) }

Thinking 1.2 main 函数在什么地方？我们又是怎么跨文件调用函数的呢？

A: main函数在init/main.c这个源文件中。通过编译生成.o文件，然后通过linker对所有目标文件进行链接，链接后填补链接前单一目标文件调用函数语句的地址空缺。

Exercise 1.4

完成boot/start.S中空缺的部分。设置栈指针，跳转到main函数。 使用gxemul –E testmips –C R3000 –M 64 elf-file运行(其中elf-file是你编译生成的vmlinux文件的路径)。

A: 增加设置栈指针，和跳转到main函数的指令

li sp, 0x80400000

jal main

Exercise 1.5

阅读相关代码和下面对于函数规格的说明，补全lib/print.c中lp\_Print()函数中缺失的部分来实现字符输出。

A: 增加输出格式判断

/\* check for flags \*/

while(1) {

c = \*fmt;

if (c == '-') ladjust = 1;

else if (c == '0') padc = '0';

else break;

fmt ++;

}

/\* check for width(number) and precision(.number) \*/

if(IsDigit(c)) {

while(IsDigit(c)){

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

c = \*++fmt;

}

}

if (c == '.') {

c = \*++fmt;

if (IsDigit(c)){

prec = 0;

while (IsDigit(c)) {

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

c = \*++fmt;

}

}

}

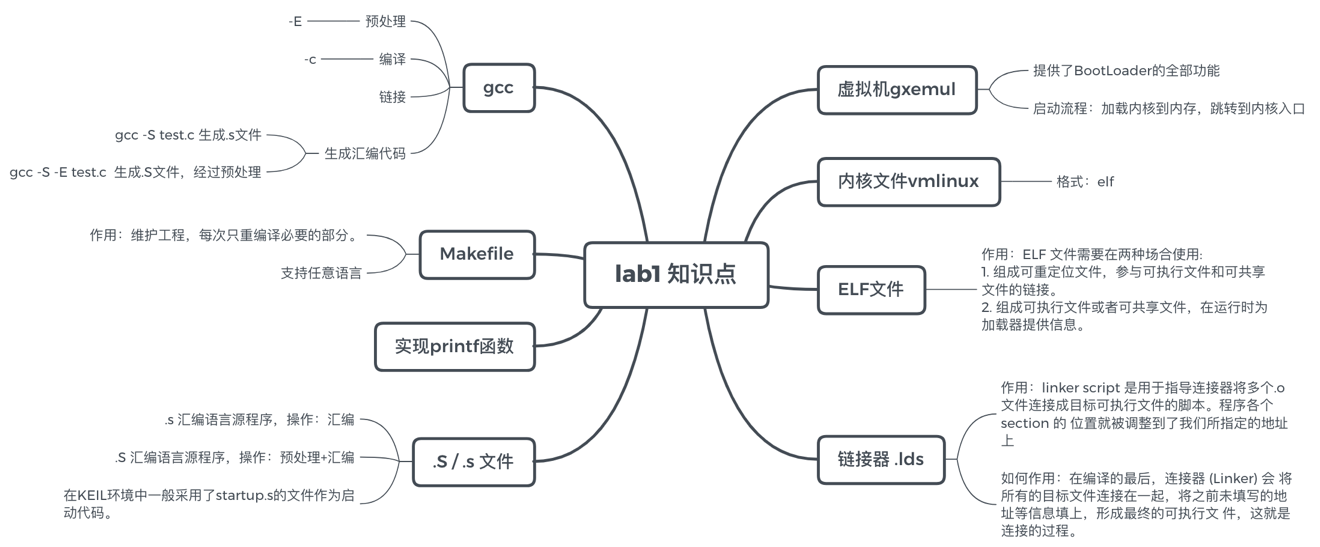
/\* check for length \*/

if (c == 'l') {

longFlag = 1;

c = \*++fmt;

}

二、实验难点图示

三、体会与感想

因为直到最后写完printf才知道自己各个部分有没有写对，所以一旦最后没有出预期的输出，就得一步一步追踪错误源头。而过程中的每一步都是刚接触的，所以做的时候也很迷茫，很不确定，很没把握，因此会花费较长时间在决定怎样写，但其实怎样写都一样，因为我并不知道哪个是对的，可能只是在纠结哪个答案是我最想猜的。

所以，最好能每一步（或每两步）就能知道自己有没有做对。

四、指导书反馈

参考资料太少，指导书更像是实验的“指导”，而不是知识教学，而网上又找不到合适的教学资料。看博客也好，StackOverflow上看问答也好，都只是碎片式的学习，更有一种见招拆招的感觉，碰到什么问题就去查，查完也只是解决了这个问题，而对整个过程、某个概念本身，没有一个系统的整体性的认识和掌握。而上机的时候不能上网查了，全凭灵性和直觉尝试，可能更糟糕。

指导书和cscore，不算碎片的查阅，我完整地看了不下三遍。看完一遍，觉得这个lab内容好多，每个又都浅尝辄止，啥都没讲，我甚至不知道指导书想让我做什么，怎样算是通过。看第二遍，还行，可以跟着开始写题。写完再看一遍，感觉良好，指导书上的简单几句像是一个方向，我应该自己扩充着去找更多更深的资料来学习，指导书真的浅尝辄止。

五、残留难点

all。

复杂的Makefile依旧是问题（语法上）。

ELF文件也不太明白，主要是数据结构不太懂（已经不会C了）。

.S文件的语法框架格式都不懂。