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。

```
[17373123_2019_jac@stu-113:~/17373123-lab$ readelf -h gxemul/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:          37052 (bytes into file)
  Flags:                             0x50001001, noreorder, o32, mips32
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         3
  Size of section headers:           40 (bytes)
  Number of section headers:         14
  Section header string table index: 11
[17373123_2019_jac@stu-113:~/17373123-lab$ readelf -h src/helloworld
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:               0x8048320
  Start of program headers:          52 (bytes into file)
  Start of section headers:          4412 (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
```

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
```

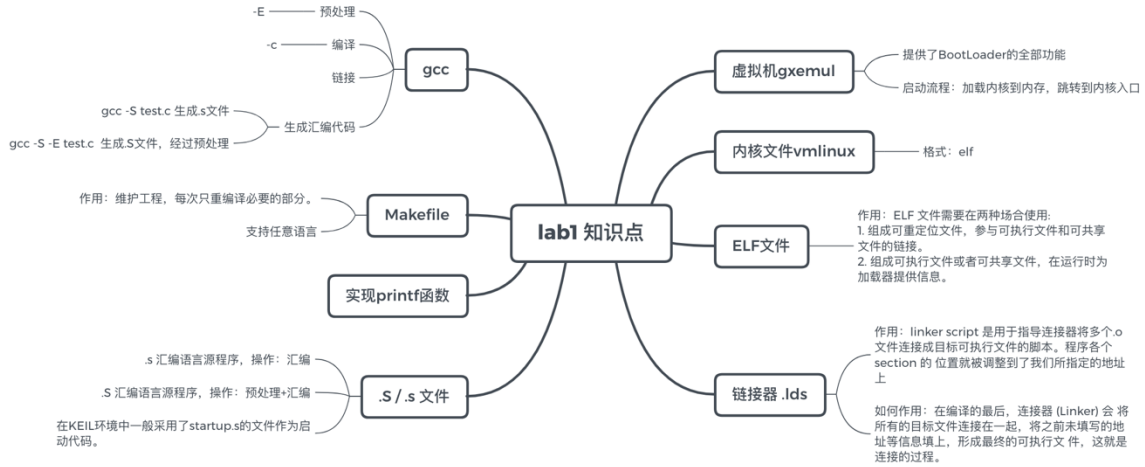
Exercise 1.5
阅读相关代码和下面对于函数规格的说明，补全 lib/print.c 中 lp_Print()函数中缺失的部分来实现字符输出。
A: 增加输出格式判断

```
    /* 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。