Week3 Report in Class (Fri56)

> 11911839 聂雨荷

Q1

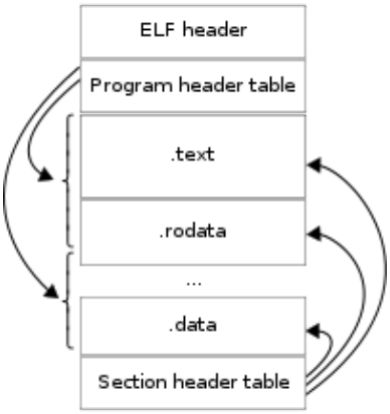
> 请详细描述本节课最小化内核的启动过程

- `make`：生成kernel的可执行文件`ucore.bin`文件，作为我们的硬盘
  - 把entry.S和init.c，stdio.c等几个c文件编译成为.o目标文件，然后链接器[3]将.o文件链 接成可执行文件kernel（elf文件），最后使用objcopy把elf文件转化成为ucore.bin[2]，这是装有我们 最小化操作系统内核的二进制文件
- `make qemu`：用qemu中自带的OpenSB作为的 bootloader，启动我们的内核
  - 给的模拟计算机插电
  - qemu会调用内置的OpenSBI作为我们的bootloader
  - OpenSBI 所做的一件事情就是把 CPU 从 M Mode 切换到 S Mode，接着跳转到一个固定地址 0x80200000，开始执行内核代码

Q2

> ELF和BIN文件的区别是什么

- elf文件包含一个文件头(ELF header)和冗余的调试信息，指定程序每个section的内存布局



- bin文件会存储所有的数据信息，它会把elf文 件指定的每段的内存布局都映射到一块线性的数据里，这块线性的数据（或者说程序）加载到内 存里就符合elf文件之前指定的布局

Q3

> 链接脚本的作用是什么

一般来说，输入文件和输 出文件都有很多section,链接脚本(linker script)的作用，就是描述怎样把输入文件的section映射到输出 文件的section, 同时规定这些section的内存布局。

Q4

> 在init.c（截图）使用cputs函数，使得在最小化内核启动后通过cputs打印出“SUSTech OS”（截图）

kern > init > C init.c > kern_init(void)

```c
#include <clock.h>
#include <console.h>
#include <defs.h>
#include <intr.h>
#include <kdebug.h>
#include <kmonitor.h>
#include <pmm.h>
#include <riscv.h>
#include <stdio.h>
#include <string.h>
#include <trap.h>

int kern_init(void) __attribute__((noreturn));
void grade_backtrace(void);
static void lab1_switch_test(void);

int kern_init(void) {
    extern char edata[], end[];
    memset(edata, 0, end - edata);

    const char *message = "os is loading ...\n";
    cputs(message);

    // clock_init();
    // ----------start-------------
    const char *sustech_msg = "SUSTech OS\n";
    cputs(sustech_msg);

    // -----------end--------------

    while (1)
        ;
}

```

Q5

> 在stdio.c中参考cputs()函数实现double_puts()函数（截图），将输出的字符串每个字符打印两次，如double_puts("SUSTech")应输出"SSUUSSTTeecchh"。在init.c中调用该函数（截图），并使得最小化内核启动后输出“IIDDOONNTTLLOOVVEEOOSS”（截图）。

```c
/*
* [Lab-3] Question 5
* Implement double_put() function, output each char twice
* eg: double_puts("SUSTech") -> "SSUUSSTTeecchh"
*/
int double_puts(const char *str){
    int cnt = 0;
    char c;
    while ((c = *str++) != '\0') {
        cputchar(c);
        cputch(c, &cnt);
    }
    cputch('\n', &cnt);
    return cnt;
}
```

C init.c ●

lab > kern > init > C init.c > kern_init(void)

```
#include <kmonitor.h>
#include <pmm.h>
#include <riscv.h>
#include <stdio.h>
#include <string.h>
#include <trap.h>

int kern_init(void) __attribute__((noreturn));
void grade_backtrace(void);
static void lab1_switch_test(void);

int kern_init(void) {
    extern char edata[], end[];
    memset(edata, 0, end - edata);

    const char *message = "os is loading ...\n";
    cputs(message);

    // clock_init();
    // ----------start-------------
    const char *love_msg = "IDONTLOVEOS";
    double_puts(love_msg);

    // -----------end--------------

    while (1)
        ;
}
```