Lab1 实验报告

19373135 田旗舰

一、实验思考题

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)

这是因为内核文件是大端存储，文件中每一个字节都是高字节在低地址、低字节在高地址中，我们的 readelf 只能解析默认的小端存储，而 readelf 工具则实现了两种存储的解析。对 vmlinux 和 testELF 使用 readelf -h 可以证实这一点，DATA 一栏不同。

```
19373135@stu-116:~/19373135-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:          37228 (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
```

```
19373135@stu-116:~/19373135-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
```

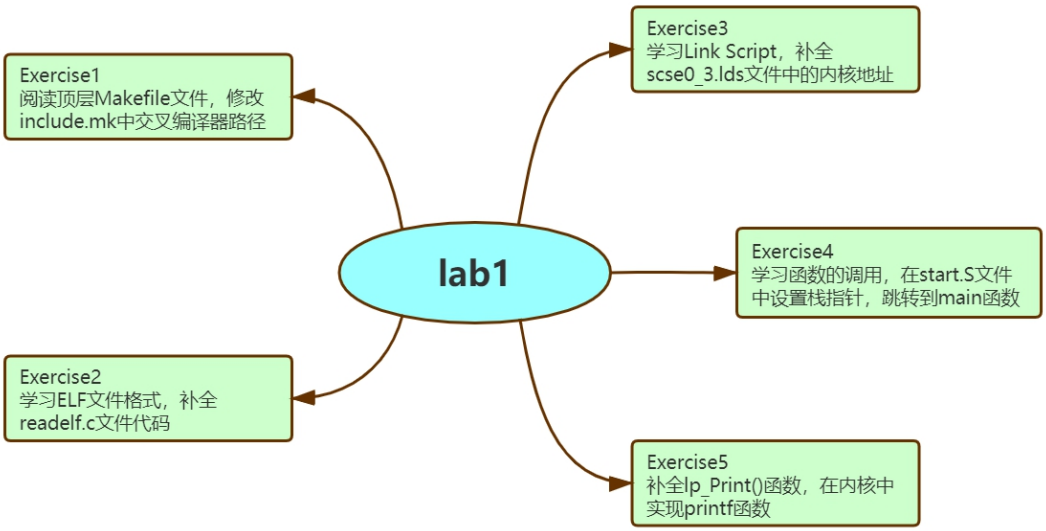
Thinking 1.2

内核入口在什么地方？main 函数在什么地方？我们是怎么让内核进入到想要的 main 函数的呢？又是怎么进行跨文件调用函数的呢？

内核入口是 _start 函数，在 boot/start.S 中被定义，main 函数在 init/main.c 中，我们通过执行跳转指令 jal，跳转到 main 函数。跨文件调用函数时，由于每个函数会被分配自己的地址，因此调用过程为首先进行保护数据，例如入栈等，然后跳转指令跳转到指定位置。

二、实验难点图示

整个 lab1 的流程图

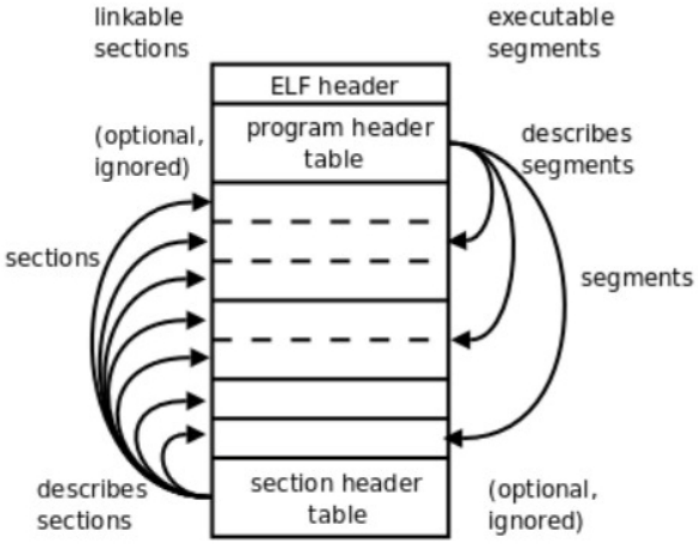


我认为本次实验最难的部分并非实验本身，而是对整个系统启动的过程的理解，例如在做 exercise 1 的时候虽然我能根据教程将路径修改正确，但其实当时在做的时候并不知道究竟在干什么，包括后面补全 tools/scse0_3.lds 将内核调整到正确的位置，以及设置栈指针，虽然能够根据内存图找到正确的地址，但是对整个系统启动的每个步骤的理解还尚有欠缺，当然 lab1 仅仅是开始，想要彻底弄懂肯定还需要后续的学习。

本次实验的难点

1.elf 文件格式的理解

在编写 readelf 函数时，必须要理解 elf 文件的格式，弄清各个数据的含义，然后找到我们需要的数据进行输出。




```c
typedef struct {
        unsigned char   e_ident[EI_NIDENT];     /* Magic number and other info */
        Elf32_Half      e_type;                 /* Object file type */
        Elf32_Half      e_machine;              /* Architecture */
        Elf32_Word      e_version;              /* Object file version */
        Elf32_Addr      e_entry;                /* Entry point virtual address */
        Elf32_Off       e_phoff;                /* Program header table file offset */
        Elf32_Off       e_shoff;                /* Section header table file offset */
        Elf32_Word      e_flags;                /* Processor-specific flags */
        Elf32_Half      e_ehsize;               /* ELF header size in bytes */
        Elf32_Half      e_phentsize;            /* Program header table entry size */
        Elf32_Half      e_phnum;                /* Program header table entry count */
        Elf32_Half      e_shentsize;            /* Section header table entry size */
        Elf32_Half      e_shnum;                /* Section header table entry count */
        Elf32_Half      e_shstrndx;             /* Section header string table index */
} Elf32_Ehdr;
```

```c
/* Section segment header.  */
typedef struct{
        Elf32_Word sh_name;             /* Section name */
        Elf32_Word sh_type;             /* Section type */
        Elf32_Word sh_flags;            /* Section flags */
        Elf32_Addr sh_addr;             /* Section addr */
        Elf32_Off  sh_offset;           /* Section offset */
        Elf32_Word sh_size;             /* Section size */
        Elf32_Word sh_link;             /* Section link */
        Elf32_Word sh_info;             /* Section extra info */
        Elf32_Word sh_addralign;        /* Section alignment */
        Elf32_Word sh_entsize;          /* Section entry size */
}Elf32_Shdr;
```

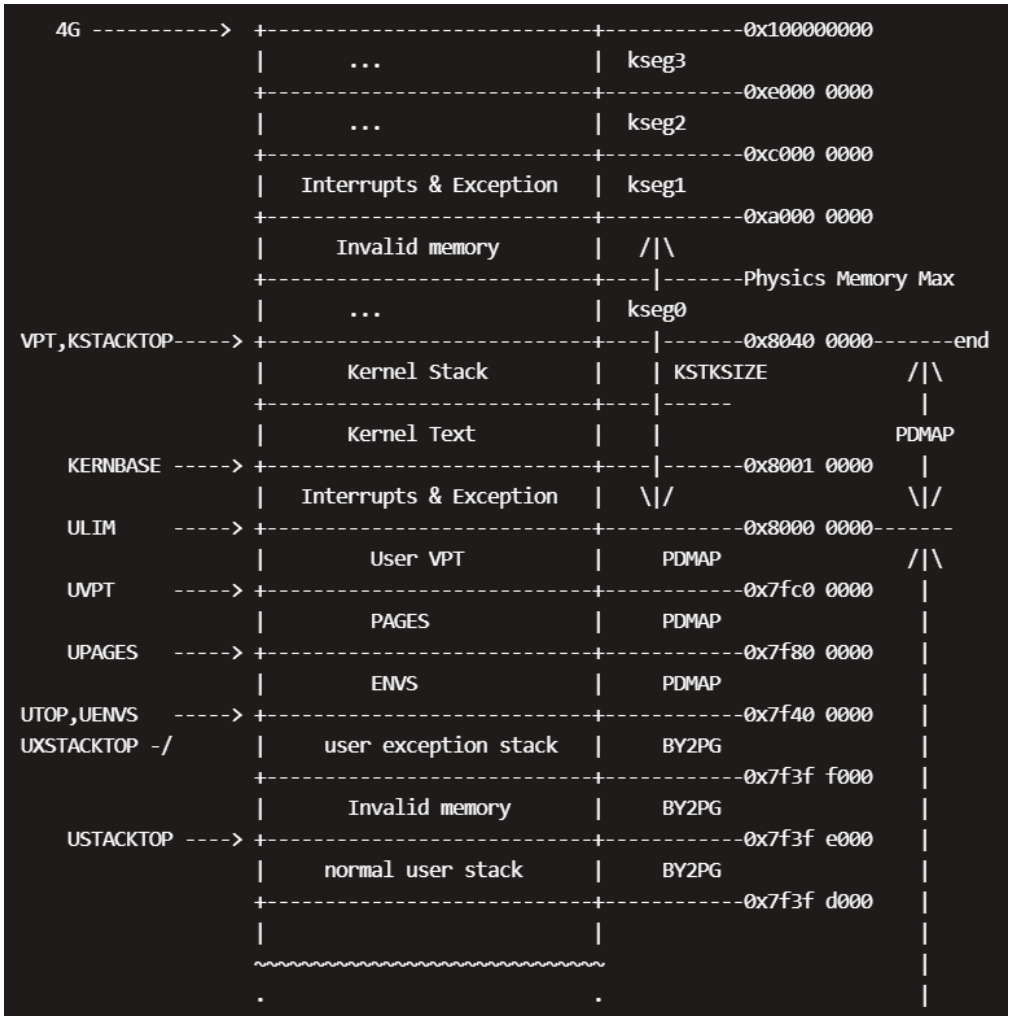
```c
/* Program segment header.  */

typedef struct {
        Elf32_Word      p_type;                 /* Segment type */
        Elf32_Off       p_offset;               /* Segment file offset */
        Elf32_Addr      p_vaddr;                /* Segment virtual address */
        Elf32_Addr      p_paddr;                /* Segment physical address */
        Elf32_Word      p_filesz;               /* Segment size in file */
        Elf32_Word      p_memsz;                /* Segment size in memory */
        Elf32_Word      p_flags;                /* Segment flags */
        Elf32_Word      p_align;                /* Segment alignment */
} Elf32_Phdr;

```

2.内存图的理解

在补充地址时，关键是要看懂各个内存区间的含义，找到对应的地址。



三、体会与感想

就实验本身而言我认为本次实验并不太难，我大概花了 5 个小时就完成了教程的全部内容，但除此之外，我还花费了 7、8 个小时去阅读实验代码、反复学习教程内容、学习理论课课件等，去理解整个系统启动的过程。lab1 正式进入操作系统的学习，虽然我们的实验是简化后的操作系统，但仅仅是刚开始的启动就让我感受到了它的复杂性，虽然我现在还没能彻底理解，但我希望能通过后续的学习能够有进一步的理解。

四、指导书反馈

ELF——深入探究编译与链接 这一节感觉有些混乱，不太好理解，尽管有所注意，但在开始阅读时还是难以区分哪些是在阐述关于 Linux 实验环境，哪些是是关于我们将要编写的操作系统，如果能用标题的形式区分或许会更加易于阅读。

在 exercise5 的 print.c 文件中，check for long 应该在 check for other prefixes 之后，但注释却将 check for long 写在了前面，有一定的误导性，希望能对注释进行修改.

五、残留难点

对于本次实验的各个 exercise1 以及上机的具体内容已经能够掌握了，但是对于整个系统其他部分的构成、功能以及不同部分之间的关系还有些不太清晰，对于这些内容需要我们去阅读整个实验代码，如何才能高效地阅读实验代码，理清各个部分的关系是我在本次实验中残留的一个难点。