

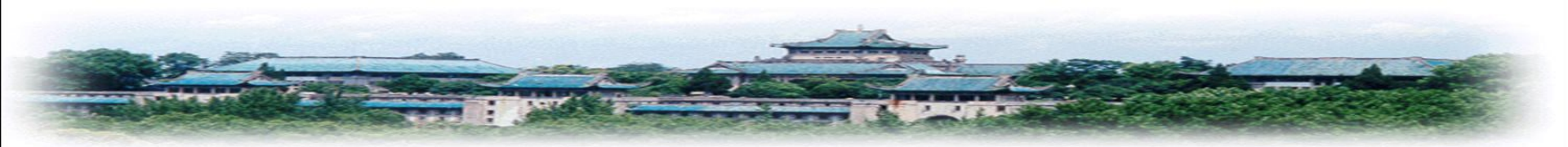
# 操作系统设计及实践

《操作系统原理》配套实验

操作系统课程组

2024年11月



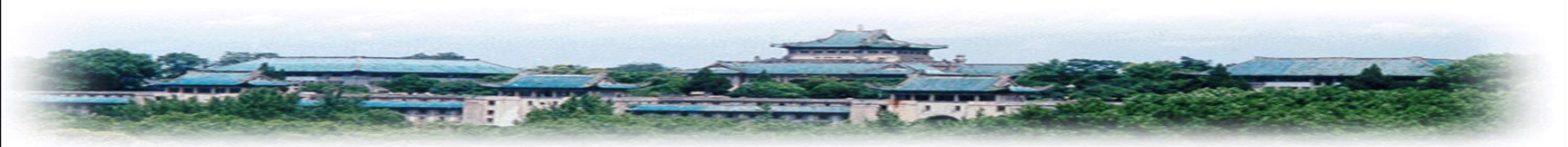


# 操作系统设计实验系列（六）

## 内核雏形



武汉大学



# 一、实验目标

- 如何生成一个内核，能引导该内核
- 对应章节：5.1—5.5





## 二、本次实验内容

1. 汇编和C的互相调用方法
  - 在例程基础上，在汇编与C程序中各添加一个简单带参数的函数调用，让两种语言撰写的程序实现混合调用，功能可自定义。
2. ELF文件格式
  - 依照书上方法，分析你修改的这个可执行文件
3. 使用Loader加载ELF文件
4. 阅读书中给出的代码结构，研究如何加载并扩展内核，对比研究一下真正内核源码的代码组织情况
5. 设计题：修改启动代码，在引导过程中在屏幕上画出一个你喜欢的ASCII图案，并将第三章的内存管理功能代码、你自己设计的中断代码集成到你的kernel文件目录管理中，并建立makefile文件，编译成内核，并引导






## 三、完成本次实验要回答的问题

- 1.汇编和C内定义的函数，相互间调用的方法是怎样的？
- 2.描述ELF文件格式以及作用，和大家学习的PE相比，结构上有什么相同和差异？
- 3.如何从Loader引导ELF的原理？
- 4.对照书中例程代码，这个内核扩展了哪些功能，这些功能流程是怎样的，他们都是在哪些源文件的代码中进行描述的？这些功能彼此有相互关联吗，给出说明？
- 5.书中代码内存的布局是怎样的？在这里有哪些是特权代码，哪些是非特权代码，在处理器控制权切换时，权限变化情况如何？
6. 下载一个真正的内核源文件，分析一下是怎么在管理组织源码文件的。
- 7.完成设计题并能演示。







## 四、需要了解的知识

### 1. 回顾Linux下汇编代码生成

- nasm -f
  - ld -s strip 去掉符号表
- [section .data]
- ...
- [section .text]
- global \_start
- \_start: ...

```
1 ; 编译链接方法
2 ; (ld的'-s'选项意为"strip all")
3 ;
4 ; $ nasm -f elf hello.asm -o hello.o
5 ; $ ld -s hello.o -o hello
6 ; $ ./hello
7 ; Hello, world!
8 ; $
9
10 [section .data] ; 数据在此
11
12 strHello      db    "Hello, world!", 0Ah
13 STRLEN        equ   $ - strHello
14
15 [section .text] ; 代码在此
16
17 global _start ; 我们必须导出 _start这个入口, 以便让链接器识别
18
19 _start:
20     mov     edx, STRLEN
21     mov     ecx, strHello
22     mov     ebx, 1
23     mov     eax, 4      ; sys_write
24     int     0x80        ; 系统调用
25     mov     ebx, 0
26     mov     eax, 1      ; sys_exit
27     int     0x80        ; 系统调用
```

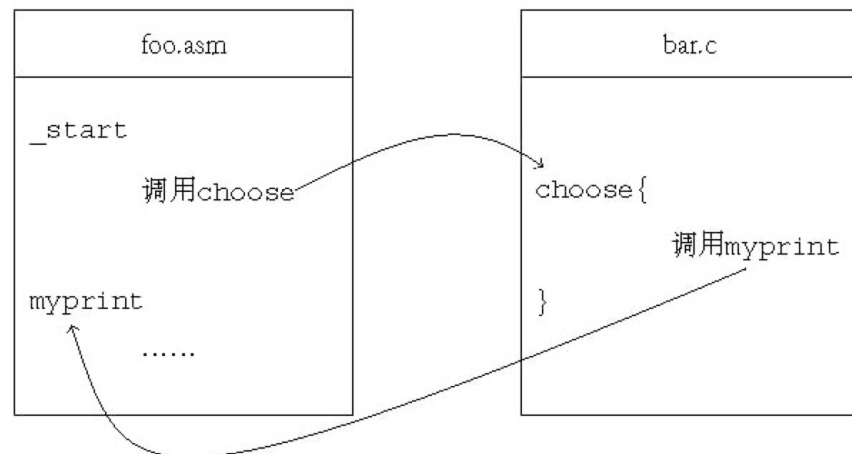


# 四、需要了解的知识

## 2.C和ASM的调用

### - 关键字:

- extern: 引入外部变量、函数的声明
- global: 导出到全局作用域



```
1 ; 编译链接方法
2 ; (ld的 '-s' 选项意为"stripall")
3 ;
4 ; $ nasm -f elf foo.asm -o foo.o
5 ; $ gcc -c bar.c -o bar.o
6 ; $ ld -s hello.o bar.o -o foobar
7 ; $ ./foobar
8 ; the 2nd one
9 ; $
10
11 extern choose ; int choose(int a, int b);
12
13 [section .data] ; 数据在此
14
15 num1st      dd      3
16 num2nd      dd      4
17
18 [section .text] ; 代码在此
19
20 global _start ; 我们必须导出 _start这个入口, 以便让链接器识别
21 global myprint ; 导出这个函数为了让bar.c使用
22
23 _start:
24     push     dword [num2nd] ; '.
```

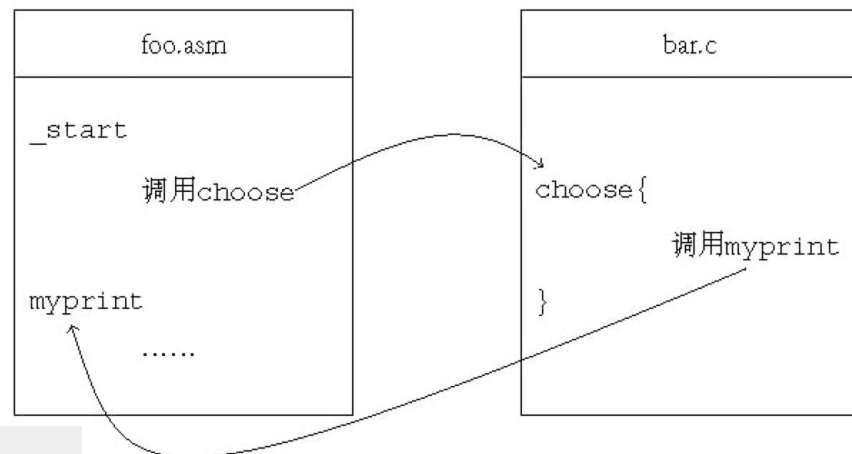
```
25     push     dword [num1st] ; /
26     call     choose ; / choose(num1st, num2nd);
27     add      esp, 8 ; /
28
29     mov      ebx, 0
30     mov      eax, 1 ; sys_exit
31     int      0x80 ; 系统调用
32
33 ; void myprint(char* msg, int len)
34 myprint:
35     mov      eax, [esp + 8] ; len
36     mov      ecx, [esp + 4] ; msg
37     mov      ebx, 1
38     mov      eax, 4 ; sys_write
39     int      0x80 ; 系统调用
40     ret
```

# 四、需要了解的知识

## 2.C和ASM的调用

### - 关键字:

- extern: 引入外部变量、函数的声明
- global: 导出到全局作用域



```
1 void myprint (char* msg, int len);
2
3 int choose(int a, int b)
4 {
5     if(a >= b) {
6         myprint ("the_1st_one\n", 13);
7     }
8     else{
9         myprint("the_2nd_one\n", 13);
10    }
```

```
> ls
bar.c  foo.asm
> nasm -f elf -o foo.o foo.asm
> gcc -c -o bar.o bar.c
> ld -s -o foobar foo.o bar.o
> ls
bar.c  bar.o  foo.asm  foobar  foo.o
> ./foobar
the 2nd one
```

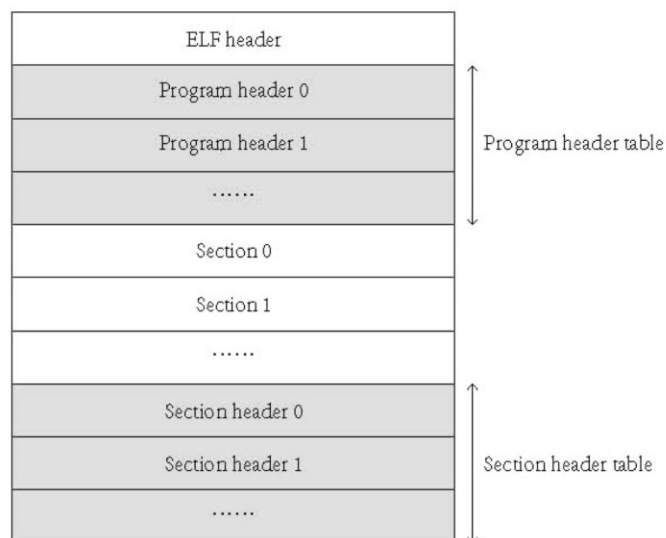


武汉大学



# 四、需要了解的知识

## 3. ELF文件格式



```
1 #define EI_NIDENT      16
2 typedef struct{
3     unsigned char      e_ident [EI_NIDENT];
4     Elf32_Half          e_type;
5     Elf32_Half          e_machine;
6     Elf32_Word          e_version;
7     Elf32_Addr          e_entry;
8     Elf32_Off           e_phoff;
9     Elf32_Off           e_shoff;
10    Elf32_Word           e_flags;
11    Elf32_Half           e_ehsize;
12    Elf32_Half           e_phentsize;
13    Elf32_Half           e_phnum;
14    Elf32_Half           e_shentsize;
15    Elf32_Half           e_shnum;
16    Elf32_Half           e_shstrndx;
17 }Elf32_Ehdr;
```

- ✓ **e\_type**, 它标识的是该文件的类型, 比如e\_type是2, 表明是一个可执行文件
- ✓ **e\_machine**, 体系结构
- ✓ **e\_version**, 文件的版本
- ✓ **e\_entry**, 程序的入口地址
- ✓ **e\_phoff**, Program header table在文件中的偏移量 (以字节计数)。
- ✓ **e\_shoff**, Section header table在文件中的偏移量 (以字节计数)
- ✓ **e\_flags**, 对IA32而言, 此项为0
- ✓ **e\_ehsize**, ELF header大小 (以字节计数)。
- ✓ **e\_phentsize**, Program header table中每一个条目的大小。
- ✓ **e\_phnum**, Program header table中有多少个条目。
- ✓ **e\_shentsize**, Section header table中每一个条目的大小
- ✓ **e\_shnum**, Section header table中有多少个条目
- ✓ **e\_shstrndx**, 包含节名称的字符串表是第几个节 (从零开始数)。

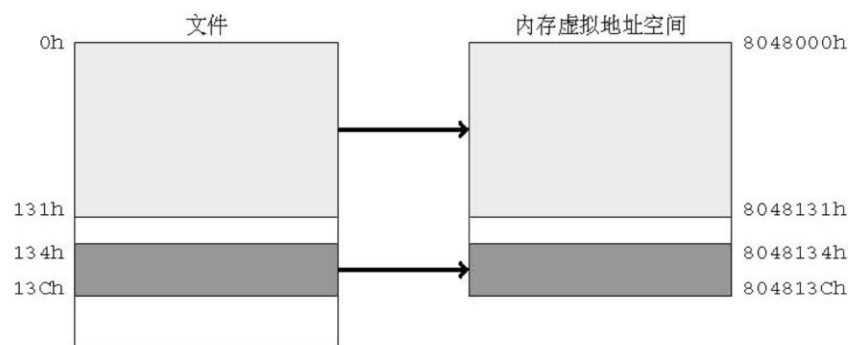


# 四、需要了解的知识

## 3.ELF文件格式：Program Header

- ✓ p\_type, 当前Program header所描述的段的类型。
- ✓ p\_offset, 段的第一个字节在文件中的偏移。
- ✓ p\_vaddr, 段的第一个字节在内存中的虚拟地址。
- ✓ p\_paddr, 在物理地址定位相关的系统中, 此项是为物理地址保留。
- ✓ p\_filesz, 段在文件中的长度。
- ✓ p\_memsz, 段在内存中的长度。
- ✓ p\_flags, 与段相关的标志。
- ✓ p\_align, 根据此项值来确定段在文件以及内存中如何对齐。


名称	Program header 0	Program header 1	Program header 2
p_type	0x1	0x1	0x6474E551
p_offset	0x0	0x134	0
p_vaddr	0x8048000	0x8049134	0
p_paddr	0x8048000	0x8049134	0
p_filesz	0x131	0x8	0
p_memsz	0x131	0x8	0
p_flags	0x5	0x6	0x7
p_align	0x1000	0x1000	0x4



Program header 0 描述的段

Program header 1 描述的段





## 四、需要了解的知识

### 4. 树形目录管理：tree

### 5. Makefile 介绍

```
# Makefile for boot

# Programs, flags, etc.
ASM          = nasm
ASMFLAGS     = -I include/

# This Program
TARGET       = boot.bin loader.bin

# All Phony Targets
.PHONY : everything clean all

# Default starting position
everything : $(TARGET)

clean :
    rm -f $(TARGET)

all : clean everything

boot.bin : boot.asm include/load.inc include/fat12hdr.inc
    $(ASM) $(ASMFLAGS) -o $@ $<

loader.bin : loader.asm include/load.inc include/fat12hdr.inc include/pm.inc
    $(ASM) $(ASMFLAGS) -o $@ $<
```

```
-- a.img
-- bochssrc
-- boot
|   |-- boot.asm
|   |-- include
|   |   |-- fat12hdr.inc
|   |   |-- load.inc
|   |   |-- pm.inc
|   |-- loader.asm
-- include
|   |-- const.h
|   |-- protect.h
|   |-- type.h
-- kernel
|   |-- kernel.asm
|   |-- start.c
-- lib
|   |-- kliba.asm
|   |-- string.asm
```

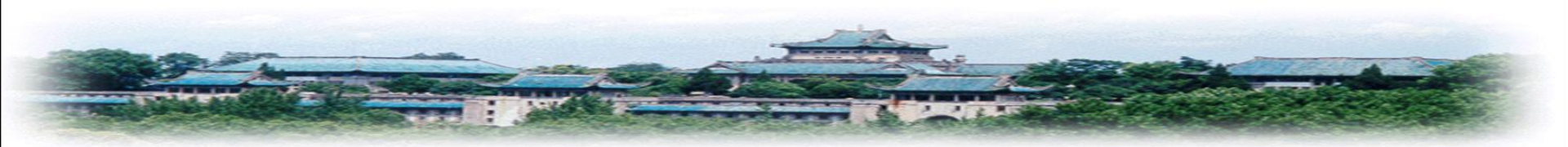




- 注意事项:

1. 出现undefined reference to `\_\_stack\_chk\_fail`错误, 需要在`Makefile`中的`\$(CFLAGS)`后面加上`-fno-stack-protector`, 即不需要栈保护
2. 如果有些同学的Ubuntu系统是64位, 而不是32位, 那么gcc默认编译的目标文件是64位, 无法和32位汇编文件汇编出的目标文件进行链接, 解决方法参考博客 (<https://blog.csdn.net/CurryXu/article/details/77481102>)
3. 由于建立了文件目录, 浏览代码时不容易查找标号对应的定义, 对于使用vim的同学, 推荐使用ctags工具, 方便标号声明和定义的跳转, 对于仍在使用自带编辑器的同学, 可以尝试vscode。
4. 当你阅读到chapter5/f/start.c时, 请留意是如何对gdt进行修改的?





谢 谢！



武汉大学