x01.treeos

ubuntu-16.04 + gcc-5.4 学习 linux-0.12，一棵长满钻石的小树。

1 准备工作

1.0 预备知识

- **资源下载**
  oldlinux
- **md 基础**
  code, title 等需要空行环绕

```
title: # h1, ## h2 ...
picture: ![Alt text](/path/img)
strong: *斜体* **粗体** ***斜粗体***
code: ``` multi-line ```  ` one-line `
换行: 两个空格+回车
引用: > 一级引用， >> 二级引用
链接: [Markdown](http://address)
分割线: *** or --- or ___
列表: 1. or * or - 加空格
| 表格: | head | head | head |
| ----- | ---- | ---- |
| cell  | cell | cell |
删除线: ~~content~~
转义: \ 加 \`~*_-+.~! 之一
```

- **diff example**

```
DifDir =
dif:
	diff -r -y -E -Z -b --suppress-blank-empty --suppress-common-lines
$(DifDir)execute.c $(DifDir)../crowbar/execute.c > diff.txt
```

1.1 计算机组成

- **组成原理**：
  1. 地址线，数据线，控制线
  2. 8259A, 8237A, 8253, 8042
  3. ISA， EISA, PCI， AGP， PCIE
- **主存储器、BIOS 和 CMOS 存储器**:
- **控制器**:
  MDA: 0xb0000-0xb2000

CGA: 0xb8000-0xbc000

- **IO端口**：
  数据端口， 命令端口，状态端口

端口地址范围	分配说明
0x000-0x01f	8237A DMA 控制器 1
0x020-0x03f	8259A 可编程中断控制器 1
0x040-0x05f	8253/8254A 定时计数器
0x060-0x06f	8042键盘控制器
0x070-0x07f	访问 CMOS RAM/实时时钟 RTC 端口
0x080-0x09f	DMA 页面寄存器端口
0x0a0-0x0bf	8259A 可编程中断控制器 2
0x0c0-0x0df	8237A DMA 控制器 2
0x0f0-0x0ff	协处理器访问端口
0x170-0x177	IDE 硬盘控制器 1
0x1f0-0x1f7	IDE 硬盘控制器 0
0x278-0x27f	并行打印机端口 2
0x2f8-0x2ff	串行控制器 2
0x378-0x37f	并行打印机端口 1
0x3b0-0x3bf	单色 MDA 显示控制器
0x3c0-0x3cf	彩色 CGA 显示控制器
0x3d0-0x3df	彩色 EGA/VGA 显示控制器
0x3f0-0x3f7	软盘控制器
0x3f8-0x3ff	串行控制器 1

1.2 内核编程语言与环境

- **as86 汇编器** 已改用 gcc 编译 gas 代码
- **GNU as 汇编**
  间接操作数，绝对跳转、调用操作数前应加 *

```
mov var, %eax      ; 地址 var 的内容放入 %eax
mov $var, %eax     ; var 的地址放入 %eax
jmp *%eax      ; 间接跳转,目标：%eax 的值
jmp *(%eax)    ; %eax 所指地址处读取跳转目标地址
.align  .ascii  .asciz  .byte
```

- **C 语言程序** `$^: 所有先决； $<: 第一先决； $@: 目标；`

1.3 保护模式

- **内存管理**
  x86 cpu 小端处理器
- **分页机制**
- **保护**
  访问数据段的特权级检查 DPL CPL RPL
  代码段转移时特权级检查 JMP CALL

1.4 实现两个进程交错运行

- **bootsect.s**
  不再使用 as86
- **head.s**
  每个目录都新建 Makefile，而 c.img, diskb.img 移到总目录外，以便备份。

1.5 ucore 实验

- **常用命令**

```
apt-get install <package>
apt-get remove <package>
apt-cache search <pattern>
spt-cache show <package>

diff -r -u -P proj_a_original proj_a_mine > diff.patch
cd proj_b
patch -p1 -u < ../diff.patch

sudo apt-get install build-essential
```

1.6 图表

一些常用的图表，放在这里，以便查找(图表已移到项目外)

1. **段描述符**

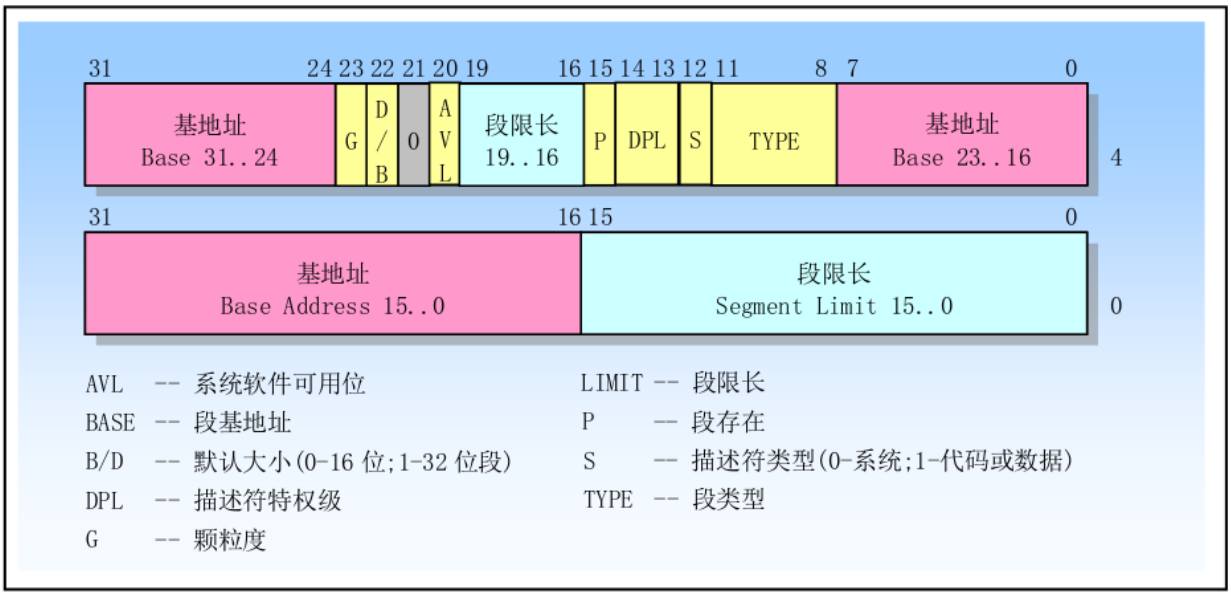

图 4-13 段描述符通用格式

2. **8259A中断**

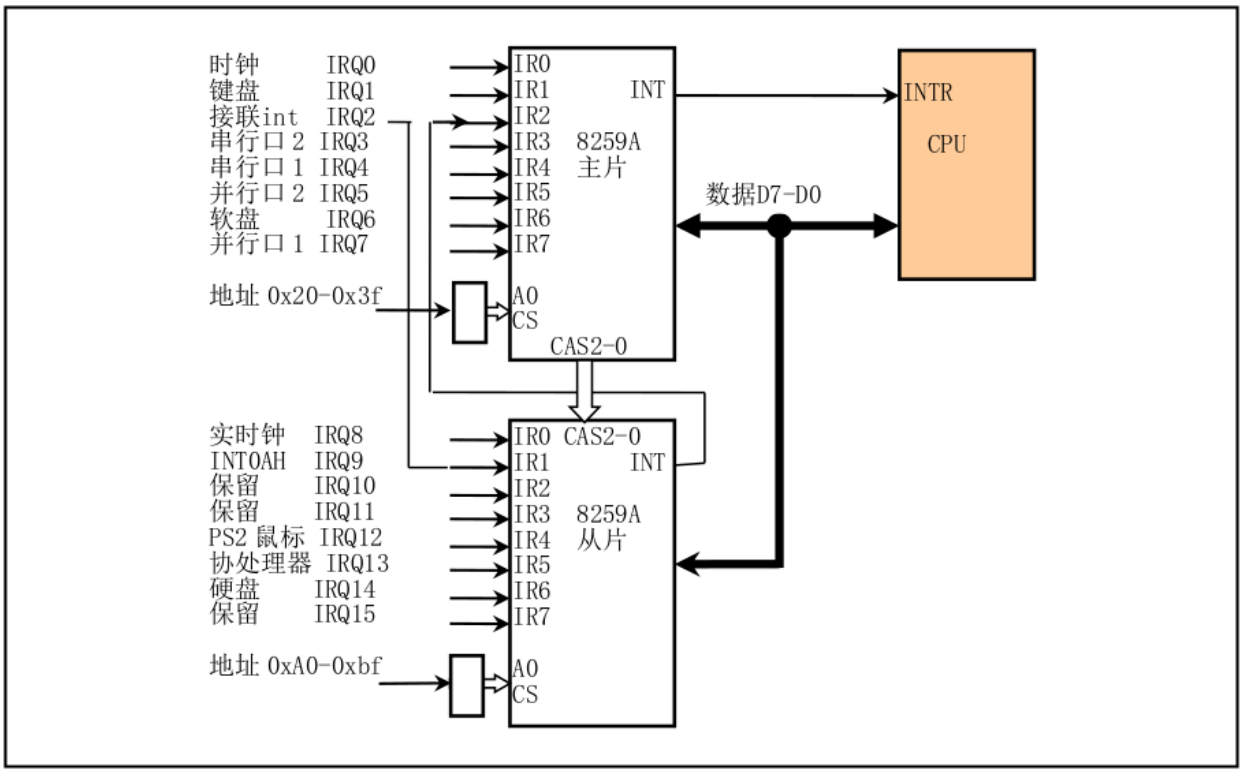

图 5-18 PC/AT 微机级联 8259 控制系统

3. **代码数据段类型**

表 4-3 代码段和数据段描述符类型

类型（TYPE）字段					描述符类型	说明
十进制	位 11	位 10	位 9	位 8		
		E	W	A		
0	0	0	0	0	数据	只读
1	0	0	0	1	数据	只读，已访问
2	0	0	1	0	数据	可读/写
3	0	0	1	1	数据	可读/写，已访问
4	0	1	0	0	数据	向下扩展，只读
5	0	1	0	1	数据	向下扩展，只读，已访问
6	0	1	1	0	数据	向下扩展，可读/写
7	0	1	1	1	数据	向下扩展，可读/写，已访问
		C	R	A		
8	1	0	0	0	代码	仅执行
9	1	0	0	1	代码	仅执行，已访问
10	1	0	1	0	代码	执行/可读
11	1	0	1	1	代码	执行/可读，已访问
12	1	1	0	0	代码	一致性段，仅执行
13	1	1	0	1	代码	一致性段，仅执行，已访问
14	1	1	1	0	代码	一致性段，执行/可读
15	1	1	1	1	代码	一致性段，执行/可读，已访问

4. **系统门段类型**

表 4-4 系统段和门描述符类型

类型（TYPE）字段					说明	
十进制	位 11	位 10	位 9	位 8		
0	0	0	0	0	Reserved	保留
1	0	0	0	1	16-Bit TSS (Available)	16 位 TSS（可用）
2	0	0	1	0	LDT	LDT
3	0	0	1	1	16-Bit TSS (Busy)	16 位 TSS（忙）
4	0	1	0	0	16-Bit Call Gate	16 位调用门
5	0	1	0	1	Task Gate	任务门
6	0	1	1	0	16-Bit Interrupt Gate	16 位中断门
7	0	1	1	1	16-Bit Trap Gate	16 位陷阱门
8	1	0	0	0	Reserved	保留
9	1	0	0	1	32-Bit TSS (Available)	32 位 TSS（可用）
10	1	0	1	0	Reserved	保留
11	1	0	1	1	32-Bit TSS (Busy)	32 位 TSS（忙）
12	1	1	0	0	32-Bit Call gate	32 位调用门
13	1	1	0	1	Reserved	保留
14	1	1	1	0	32-Bit Interrupt Gate	32 位中断门

5. **文件系统引用关系**

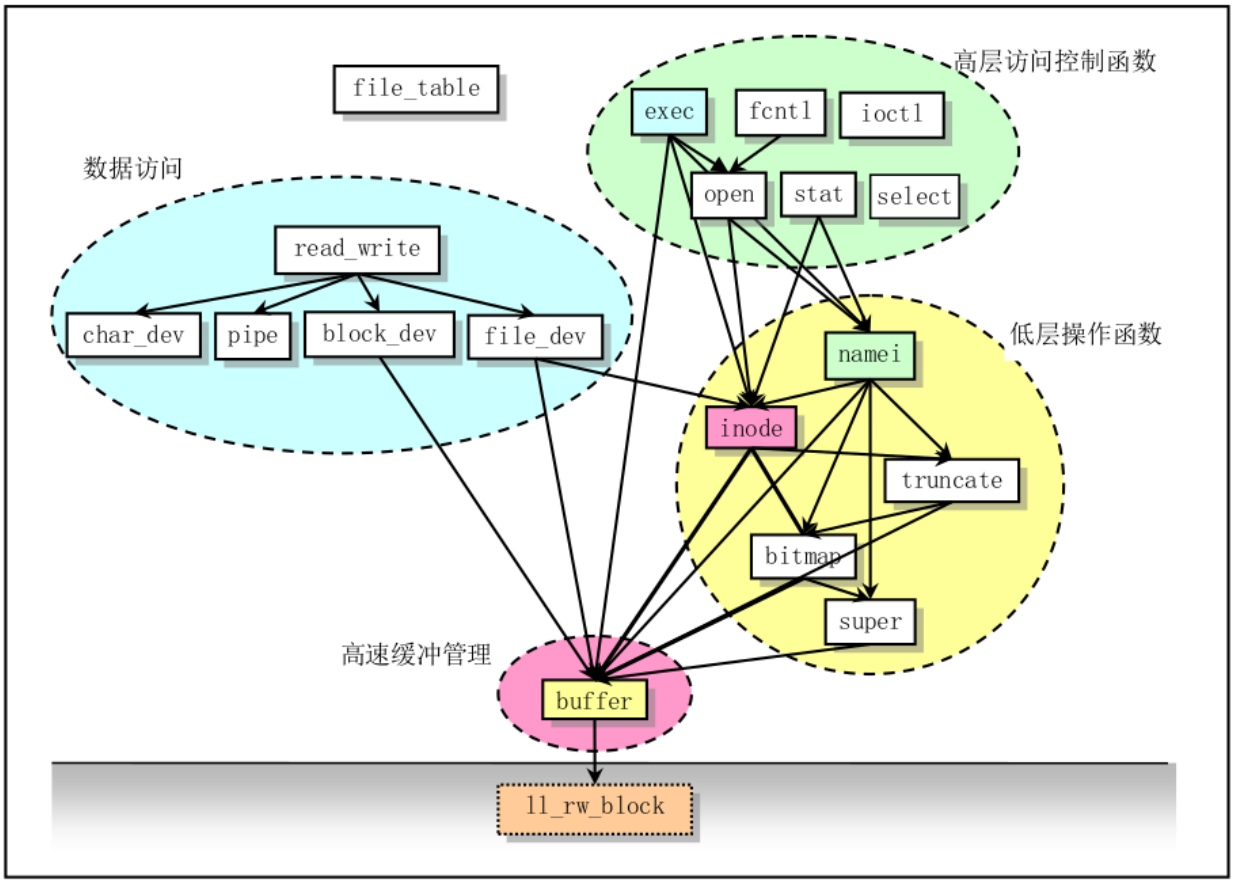

图 5-29 fs 目录中各程序中函数之间的引用关系。

2 引导启动程序

2.1 bootsect.s

1. 修改软驱的参数表
2. **read_it** 一次读一个磁道

2.2 setup.s

1. 获取系统数据
2. 进入保护模式

2.3 head.s

1. 启动分页
2. 进入 main 任务0

3 init

1. 安装根文件系统
2. 显示系统信息
3. 执行资源配置文件
4. 执行用户登录 shell

4 内存管理

利用页目录和页表结构管理内存

- page.s: 页异常处理
- memory.c: 页面管理
- swap.c: 页面交换

4.1 地址变换

- 线性地址到逻辑地址

```
PDE = addr >> 22
PTE = (addr >> 12) & 0x3ff
offset = addr & 0xfff
```

- 线性地址到物理地址

```
phyaddr = *( (*((laddr>>20)&0xffc)&0xfffff000)
           + ((laddr>>10) & 0xffc) )
        + (laddr & 0xfff)
```

4.2 swap.c

- **init_swapping()**
  1. blk_size 何时被设置?
  2. `blk_size[MAJOR(SWAP_DEV)][MINOR(SWAP_DEV)]` 获取交换设备块数（1k), 进而除 4 获取页数（4k).
  3. `read_swap_page(0, swap_bitmap)` 把交换设备页面 0 读入`swap_bitmap`中。
  4. `bit(swap_bitmap,i)` 测试，0 占用，1 空闲。
- **get_swap_page(),swap_out(), swap_in(),read_swap_page(),write_swap_page()**
  1. 本质上仍然是底层设备的读写，采取位映射的方式进行管理。
  2. `swap_nr<<1` 页表项存在位 P=0