

河南工业大学

# 课程设计报告

## 一个小型图形界面操作系统

课程名称：\_\_\_\_操作系统原理\_\_\_\_

专业班级：\_\_\_\_软件 1305 班\_\_\_\_

姓 名：\_\_\_\_田劲锋\_\_\_\_

学 号：\_\_\_\_201316920311\_\_\_\_

指导教师：\_\_\_\_刘扬\_\_\_\_

完成时间：\_\_\_\_2015 年 7 月 3 日\_\_\_\_



## 软件工程 专业课程设计任务书

学生姓名	田劲锋	专业班级	软件 1305 班	学 号	201316920311
题 目	一个小型图形界面操作系统				
课题性质	其他		课题来源	自拟课题	
指导教师	刘扬		同组姓名	无	
主要内容	<p>操作系统是控制应用程序执行的程序，并充当应用程序和计算机硬件之间的接口。一个操作系统的主要功能有：</p> <ol style="list-style-type: none"> <li>1. 处理器管理</li> <li>2. 存储器管理</li> <li>3. 设备管理</li> <li>4. 文件管理</li> </ol> <p>现在的桌面操作系统多是多任务分时操作系统。</p>				
任务要求	<p>目标是完成一个基本可用的图形界面操作系统，包括如下基本模块：</p> <ol style="list-style-type: none"> <li>1. 进程：中断处理、多任务调度、系统保护</li> <li>2. 存储管理：内存分配、进程空间管理</li> <li>3. I/O 系统：鼠标、键盘和屏幕的控制</li> <li>4. 文件系统：文件与可执行程序的读取和加载</li> </ol> <p>系统提供命令行用户接口和图形化用户接口，允许使用 C 语言编写系统应用程序，可以从 FAT12 格式软盘启动。</p>				
参考文献	<p>川合秀实. 30天自制操作系统. 人民邮电出版社, 2012</p> <p>W. Stallings. 操作系统: 精髓与设计原理（第6版）. 机械工业出版社, 2010</p> <p>R. E. Bryant, 等. 深入理解计算机系统系统（第2版）. 机械工业出版社, 2010</p> <p>A.S.Tanenbaum, 等. 操作系统设计与实现. 电子工业出版社, 2007</p> <p>W. R. Stevens, 等. UNIX 环境高级编程（第3版）. 人民邮电出版社, 2014</p>				
审查意见	<p>指导教师签字：</p>          <p>教研室主任签字：2015 年 6 月 25 日</p>				

说明：本表由指导教师填写，由教研室主任审核后下达给选题学生，装订在设计（论文）首页



# 目录

<b>1</b>	<b>概述</b>	<b>3</b>
<b>2</b>	<b>设计</b>	<b>5</b>
2.1	引导程序 . . . . .	5
2.2	设备管理 . . . . .	9
2.2.1	中断处理 . . . . .	10
2.2.2	键盘 . . . . .	11
2.2.3	鼠标 . . . . .	12
2.2.4	屏幕 . . . . .	13
2.2.5	窗口管理器 . . . . .	15
2.3	进程管理 . . . . .	19
2.4	内存管理 . . . . .	22
2.5	文件管理 . . . . .	24
2.6	系统接口 . . . . .	25
2.6.1	终端 . . . . .	28
<b>3</b>	<b>总结</b>	<b>35</b>
	<b>参考文献</b>	<b>37</b>



# 1 概述

操作系统（Operating System）是控制应用程序执行的程序，并充当应用程序和计算机硬件之间的接口。它有以下三个目标：

- 方便：操作系统是计算机更易于使用。
- 有效：操作系统允许以更有效的方式使用计算机系统资源。
- 扩展能力：在构造操作系统时，应该允许在不妨碍服务的前提下有效地开发、测试和引进新的系统功能。

作为用户/计算机接口下的操作系统，提供了程序开发、程序运行、输入输出设备访问、文件访问控制、系统访问、错误检测和相应。作为资源管理器的操作系统，包括内核程序和当前正在使用的其他操作系统程序，统筹软硬件。作为扩展机的操作系统，能够不断发展。

操作系统是最复杂的软件之一，这反映在为了达到那些困难的甚至相互冲突的目标而带来的挑战上。操作系统开发中5个重要的理论[1]：

- 进程
- 内存管理
- 信息保护和安全
- 调度和资源管理
- 系统结构

该操作系统实现了基本的进程管理、内存管理、窗口和图层管理，以及简陋的文件管理。

下面的截图展示了操作系统的实际运行效果。

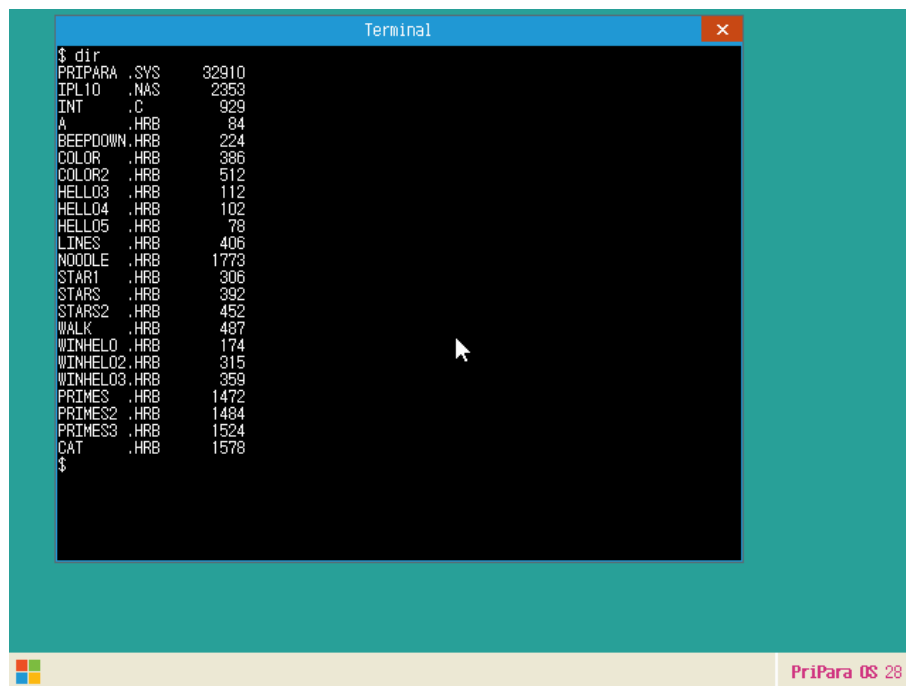


图 1

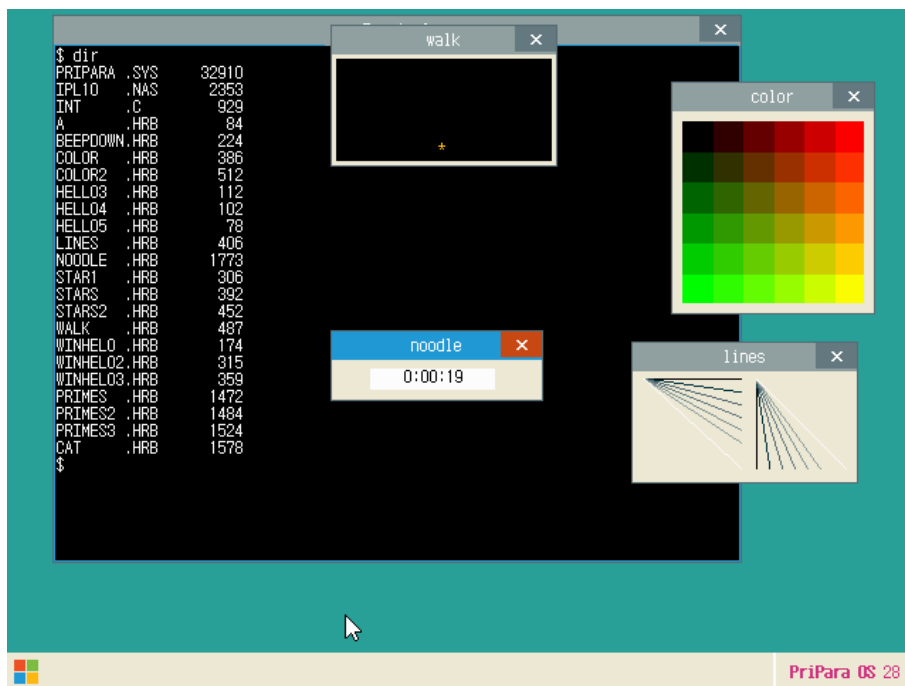


图 2

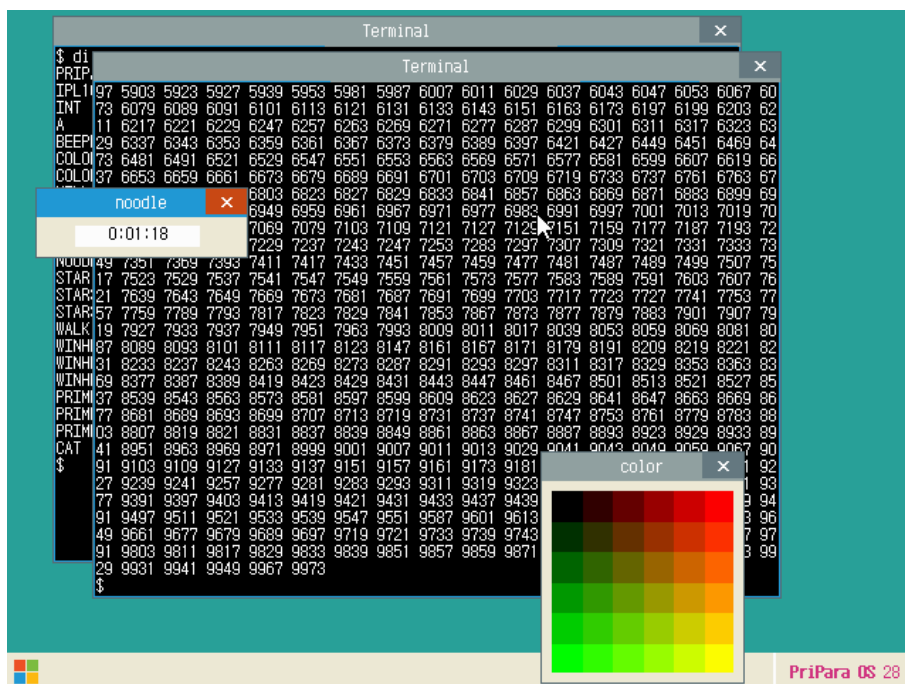


图 3



## 2 设计

该课程设计内容，主要是以川合秀实老师所著《30天自制操作系统》[2]一书中介绍的OSASK操作系统为基础的。

代码以C语言和汇编写成，其中汇编是nasm的一个方言NASK，而C语言则是ANSI C，使用gcc编译器可以编译。编译成为启动镜像文件的Makefile适用于Windows平台（可以移植到其他平台），在z\_tools目录中提供了所需要的编译程序和链接库。

### 2.1 引导程序

系统存放在一个1.44MB软盘中，其第一扇区为引导程序ipl10.bin，作用是将软盘中的前10个柱面读入内存中。

该系统只支持读取FAT12格式，所以首先是格式化的代码。

Listing 1: FAT12格式软盘格式化

```
10 DB 0x90
11 DB "PURIPARA" ; 启动区名 (8 字节)
12 DW 512 ; 每个扇区大小 (必须为 512 字节)
13 DB 1 ; 簇大小 (必须 1 扇区)
14 DW 1 ; FAT 起始位置 (一般从 1 开始)
15 DB 2 ; FAT 个数 (必须为 2)
16 DW 224 ; 根目录大小 (一般为 224 项)
17 DW 2880 ; 磁盘大小 (必须为 2880 扇区)
18 DB 0xf0 ; 磁盘类型 (必须为 0xf0)
19 DW 9 ; FAT 长度 (必须为 9 扇区)
20 DW 18 ; 磁道扇区数 (必须为 18)
21 DW 2 ; 磁头数 (必须是 2)
22 DD 0 ; 不使用的分区 (必须为 0)
23 DD 2880 ; 再次重写磁盘大小
24 DB 0, 0, 0x29 ; 意义不明的固定写法
25 DD 0xffffffff ; 卷标号码 (可能)
26 DB "PRIPARA-OS " ; 磁盘名 (11 字节)
27 DB "FAT12 " ; 磁盘格式 (8 字节)
28 RESB 18 ; 空出
```

下面分别列出了读取一个扇区、18个扇区、10个柱面的汇编代码：

Listing 2: 读取一个扇区

```
40 MOV AX, 0x0820
41 MOV ES, AX
42 MOV CH, 0 ; 柱面 0
43 MOV DH, 0 ; 磁头 0
44 MOV CL, 2 ; 扇区 2
49 retry:
50 MOV AH, 0x02 ; AH=0x02 : 读盘
51 MOV AL, 1 ; 1 个扇区
52 MOV BX, 0
53 MOV DL, 0x00 ; 驱动器 A:
54 INT 0x13 ; 调用磁盘 BIOS
55 JNC next ; 没有错误
56 ADD SI, 1 ; SI += 1
57 CMP SI, 5 ; 比较 SI 和 5
58 JAE error ; 如果 SI >= 5 跳到 error
59 MOV AH, 0x00
60 MOV DL, 0x00 ; 驱动器 A:
61 INT 0x13 ; 重置驱动器
62 JMP retry
85 error:
86 MOV SI, msg
102 msg:
103 DB 0x0a, 0x0a ; 两个换行
104 DB "load error"
```

```

105     DB 0x0a ; 换行
106     DB 0
107     RESB 0x7dfe-$ ; 填充0到0x7dfe
108     DB 0x55, 0xaa

```

Listing 3: 读取18个扇区

```

46 readloop:
47     MOV SI, 0 ; 记录失败次数
64 next:
65     MOV AX, ES ; 内存地址后移0x200
66     ADD AX, 0x0020
67     MOV ES, AX ; ES += 512 / 16
68     ADD CL, 1 ; CL += 1
69     CMP CL, 18 ; 比较CL和18
70     JBE readloop ; 如果CL <= 18跳到readloop

```

Listing 4: 读取10个柱面

```

71     MOV CL, 1
72     ADD DH, 1
73     CMP DH, 2
74     JB readloop ; 如果DH < 2跳到readloop
75     MOV DH, 0
76     ADD CH, 1
77     CMP CH, CYLS
78     JB readloop ; 如果CH < CYLS跳到readloop
82     MOV [0x0ff0], CH ; 告知IPL加载到了何处
83     JMP 0xc200
88 putloop:
89     MOV AL, [SI] ; 待显示字符
90     ADD SI, 1 ; SI++
91     CMP AL, 0
92     JE fin
93     MOV AH, 0x0e ; 显示一个字的指令
94     MOV BX, 15 ; 指定颜色, 并不管用
95     INT 0x10 ; 调用显卡BIOS
96     JMP putloop
98 fin:
99     HLT ; 停止CPU, 等待
100    JMP fin ; 无限循环

```

将磁盘上的内容读入到内存之后, 开始载入操作系统内核。我们让操作系统进入图形模式:

Listing 5: 启动信息

```

5  VBEMODE EQU 0x101
6  ; (VBE画面模式列表)
7  ; 0x100 : 640 x 400 x 8位色
8  ; 0x101 : 640 x 480 x 8位色
9  ; 0x103 : 800 x 600 x 8位色
10 ; 0x105 : 1024 x 768 x 8位色
11 ; 0x107 : 1280 x 1024 x 8位色
12
13 BOTPAK EQU 0x00280000 ; bootpack加载目的
14 DSKCAC EQU 0x00100000 ; 磁盘缓存
15 DSKCAC0 EQU 0x00008000 ; 磁盘缓存(实模式)
16
17 ; BOOT_INFO有关
18 CYLS EQU 0x0ff0 ; 设定启动区
19 LEDS EQU 0x0ff1
20 VMODE EQU 0x0ff2 ; 颜色位数
21 SCRNX EQU 0x0ff4 ; 水平分辨率
22 SCRNY EQU 0x0ff6 ; 垂直分辨率
23 VRAM EQU 0x0ff8 ; 图像缓冲区地址

```

对于支持VESA BIOS扩展的BIOS, 我们进入高分辨率模式(640 x 400 x 8位色):

Listing 6: 判断VBE并进入高分辨率模式

```

27 ; 判断是否存在VBE
28
29     MOV AX, 0x9000
30     MOV ES, AX
31     MOV DI, 0

```

```

32     MOV AX, 0x4f00
33     INT 0x10
34     CMP AX, 0x004f
35     JNE scrn320
36
37 ; 检查 VBE 版本 > 2.0
38
39     MOV AX, [ES:DI+4]
40     CMP AX, 0x0200
41     JB scrn320 ; if (AX < 0x0200) goto scrn320
42
43 ; 获取画面模式信息
44
45     MOV CX, VBEMODE
46     MOV AX, 0x4f01
47     INT 0x10
48     CMP AX, 0x004f
49     JNE scrn320
50
51 ; 确认画面模式信息
52
53     CMP BYTE [ES:DI+0x19], 8 ; 颜色数为 8
54     JNE scrn320
55     CMP BYTE [ES:DI+0x1b], 4 ; 调色板模式
56     JNE scrn320
57     MOV AX, [ES:DI+0x00] ; 模式属性能否加上 0x4000
58     AND AX, 0x0080
59     JZ scrn320 ; 如果不能
60
61 ; 切换画面模式
62
63     MOV BX, VBEMODE+0x4000
64     MOV AX, 0x4f02
65     INT 0x10
66     MOV BYTE [VMODE], 8 ; 记录画面模式
67     MOV AX, [ES:DI+0x12]
68     MOV [SCRNX], AX
69     MOV AX, [ES:DI+0x14]
70     MOV [SCRNY], AX
71     MOV EAX, [ES:DI+0x28]
72     MOV [VRAM], EAX
73     JMP keystatus

```

对于不支持 VBE 的主板，进入低分辨率模式：

Listing 7: 低分辨率模式

```

75 scrn320:
76     MOV AL, 0x13 ; VGA 320x200x8 位色
77     MOV AH, 0x00
78     INT 0x10
79     MOV BYTE [VMODE], 8 ; 记录画面模式
80     MOV WORD [SCRNX], 320
81     MOV WORD [SCRNY], 200
82     MOV DWORD [VRAM], 0x000a0000

```

获取键盘指示灯和屏蔽终端后，开始切换进入 32 位模式：

Listing 8: 进入 32 位模式转存数据

```

111 ; 切换到保护模式
112
113 [INSTRSET "i486p"] ; 使用 486 指令集
114
115     LGDT [GDTR0] ; 设置临时 GDT
116     MOV EAX, CR0
117     AND EAX, 0x7fffffff ; 将 bit31 置 0 (禁止分页)
118     OR EAX, 0x00000001 ; 将 bit0 置 1 (切换到保护模式)
119     MOV CR0, EAX
120     JMP pipelineflush ; 重置 CPU 流水线
121 pipelineflush:
122     MOV AX, 1*8 ; 32bit 可读写段
123     MOV DS, AX
124     MOV ES, AX
125     MOV FS, AX
126     MOV GS, AX

```

```

127     MOV SS, AX
128
129 ; 传送 bootpack
130
131     MOV ESI, bootpack ; 传送来源
132     MOV EDI, BOTPAK ; 传送目的
133     MOV ECX, 512*1024/4
134     CALL memcpy
135
136 ; 转存磁盘数据
137
138 ; 启动扇区
139
140     MOV ESI, 0x7c00 ; 传送来源
141     MOV EDI, DSKCAC ; 传送目的
142     MOV ECX, 512/4
143     CALL memcpy
144
145 ; 剩下的
146
147     MOV ESI, DSKCAC0+512; 传送来源
148     MOV EDI, DSKCAC+512 ; 传送目的
149     MOV ECX, 0
150     MOV CL, BYTE [CYLS]
151     IMUL ECX, 512*18*2/4 ; 柱面数变为字节数/4
152     SUB ECX, 512/4 ; 减去 IPL
153     CALL memcpy

```

然后调用主函数，正式启动操作系统：

Listing 9: 启动 bootpack

```

157 ; 启动 bootpack
158
159     MOV EBX, BOTPAK
160     MOV ECX, [EBX+16]
161     ADD ECX, 3 ; ECX += 3;
162     SHR ECX, 2 ; ECX /= 4;
163     JZ skip ; 没有要传送的东西
164     MOV ESI, [EBX+20] ; 传送来源
165     ADD ESI, EBX
166     MOV EDI, [EBX+12] ; 传送目的
167     CALL memcpy
168 skip:
169     MOV ESP, [EBX+12] ; 初始化栈
170     JMP DWORD 2*8:0x0000001b
171     ALIGNB 16
172 bootpack:

```

Listing 10: 启动信息结构体

```

9  typedef struct BOOTINFO { /* 0x0ff0-0x0fff */
10     char cyls; /* 启动区读盘终止处 */
11     char leds; /* 键盘灯状态 */
12     char vmode; /* 显卡模式 */
13     char reserve;
14     short scrnx, scrny; /* 分辨率 */
15     char* vram;
16 } bootinfo_t;

```

操作系统首先初始化中断描述符表、系统FIFO队列、鼠标键盘等：

Listing 11: 初始化设备

```

53     init_gdtidt();
54     init_pic();
55     io_sti(); /* IDT/PIC初始化后解除对CPU中断的禁止 */
56     fifo32_init(&fifo, 128, fifobuf, 0);
57     *((int*)0x0fec) = (int)&fifo;
58     init_pit();
59     init_keyboard(&fifo, 256);
60     enable_mouse(&fifo, 512, &mdec);
61     io_out8(PIC0_IMR, 0xf8); /* 允许PIC1、PIT和键盘(11111000) */
62     io_out8(PIC1_IMR, 0xef); /* 允许鼠标(11101111) */
63     fifo32_init(&keycmd, 32, keycmd_buf, 0);

```

然后初始化内存管理器:

Listing 12: 初始化内存管理器

```
65 unsigned int memtotal = mentest(0x00400000, 0xbfffffff);
66 memman_init(memman);
67 memman_free(memman, 0x00001000, 0x0009e000); /* 0x00001000 - 0x0009efff */
68 memman_free(memman, 0x00400000, memtotal - 0x00400000);
```

初始化调色板和桌面, 启动一个默认的终端窗口:

Listing 13: 初始化桌面

```
70 init_palette();
71 shtctl = shtctl_init(memman, binfo->vram, binfo->scrnx, binfo->scrny);
72 /* sht_back */
73 sht_back = sheet_alloc(shtctl);
74 buf_back = (unsigned char*)memman_alloc_4k(memman, binfo->scrnx * binfo->scrny);
75 /* sht_cons */
76 key_win = open_console(shtctl, memtotal);
```

初始化鼠标指针:

Listing 14: 初始化鼠标

```
89 /* sht_mouse */
90 sht_mouse = sheet_alloc(shtctl);
91 sheet_setbuf(sht_mouse, buf_mouse, CURSOR_X, CURSOR_Y, 99);
92 init_mouse_cursor8(buf_mouse, 99);
```

这时候系统就已经算是启动完成了。接下来进入一个无限循环, 该循环查询CPU中断事件, 并给与响应:

Listing 15: 主循环

```
104 for (;;) {
105     if (fifo32_status(&keycmd) > 0 && keycmd_wait < 0) {
106         /* 如果存在向键盘控制器发送的数据, 发送之 */
107     }
108     io_cli();
109     if (fifo32_status(&fifo) == 0) {
110         /* FIFO为空, 当存在搁置的绘图操作时立即执行 */
111     } else {
112         i = fifo32_get(&fifo);
113         io_sti();
114         if (key_win != 0 && key_win->flags == 0) { /* 窗口关闭 */
115         }
116         if (256 <= i && i <= 511) { /* 键盘 */
117         } else if (512 <= i && i <= 767) { /* 鼠标 */
118         }
119     }
120 }
121 }
```

## 2.2 设备管理

我们用多个先进先出 (FIFO) 队列来管理系统中的各种消息:

Listing 16: 队列结构体

```
48 typedef struct FIFO32 {
49     int* buf;
50     int p, q, size, free, flags;
51     struct TASK* task;
52 } fifo32;
```

Listing 17: FIFO队列

```
7 /* 初始化FIFO缓冲区 */
8 void fifo32_init(fifo32* q, int size, int* buf, struct TASK* task)
9 {
```

```

10     q->size = size;
11     q->buf = buf;
12     q->free = size; /* 空 */
13     q->flags = 0;
14     q->p = 0; /* 队尾 */
15     q->q = 0; /* 队首 */
16     q->task = task; /* 有数据写入时需要唤醒的任务 */
17     return;
18 }
19
20 /* 压入FIFO堆里 */
21 int fifo32_put(fifo32* q, int data)
22 {
23     if (q->free == 0) {
24         /* 溢出 */
25         q->flags |= FLAGS_OVERRUN;
26         return -1;
27     }
28     q->buf[q->p] = data;
29     q->p++;
30     if (q->p == q->size) {
31         q->p = 0;
32     }
33     q->free--;
34     if (q->task != 0) {
35         if (q->task->flags != 2) { /* 任务在休眠 */
36             task_run(q->task, -1, 0); /* 唤醒 */
37         }
38     }
39     return 0;
40 }
41
42 /* 弹出FIFO队列 */
43 int fifo32_get(fifo32* q)
44 {
45     int data;
46     if (q->free == q->size) {
47         /* 已空, 返回-1 */
48         return -1;
49     }
50     data = q->buf[q->q];
51     q->q++;
52     if (q->q == q->size) {
53         q->q = 0;
54     }
55     q->free++;
56     return data;
57 }
58
59 /* 队列长度 */
60 int fifo32_status(fifo32* q)
61 {
62     return q->size - q->free;
63 }

```

## 2.2.1 中断处理

Listing 18: GDT和IDT 结构体

```

103 typedef struct SEGMENT_DESCRIPTOR {
104     short limit_low, base_low;
105     char base_mid, access_right;
106     char limit_high, base_high;
107 } segment_descriptor;
108
109 typedef struct GATE_DESCRIPTOR {
110     short offset_low, selector;
111     char dw_count, access_right;
112     short offset_high;
113 } gate_descriptor;

```

首先是初始化GDT和IDT:

Listing 19: 初始化GDT和IDT

```

5 void init_gdtidt(void)
6 {
7     segment_descriptor* gdt = (segment_descriptor*)ADR_GDT;
8     gate_descriptor* idt = (gate_descriptor*)ADR_IDT;
9     int i;
10
11     /* GDT初始化 */
12     for (i = 0; i < 8192; i++) {
13         set_segmdesc(gdt + i, 0, 0, 0, 0);
14     }
15     set_segmdesc(gdt + 1, 0xffffffff, 0x00000000, AR_DATA32_RW);
16     set_segmdesc(gdt + 2, LIMIT_BOTPAK, ADR_BOTPAK, AR_CODE32_ER);
17     load_gdtr(LIMIT_GDT, ADR_GDT);
18
19     /* IDT初始化 */
20     for (i = 0; i < 256; i++) {
21         set_gatedesc(idt + i, 0, 0, 0, 0);
22     }
23     load_idtr(LIMIT_IDT, ADR_IDT);
24
25     /* IDT设置 */
26     set_gatedesc(idt + 0x0c, (int)asm_inthandler0c, 2 * 8, AR_INTGATE32);
27     set_gatedesc(idt + 0x0d, (int)asm_inthandler0d, 2 * 8, AR_INTGATE32);
28
29     set_gatedesc(idt + 0x20, (int)asm_inthandler20, 2 * 8, AR_INTGATE32);
30     set_gatedesc(idt + 0x21, (int)asm_inthandler21, 2 * 8, AR_INTGATE32);
31     set_gatedesc(idt + 0x27, (int)asm_inthandler27, 2 * 8, AR_INTGATE32);
32     set_gatedesc(idt + 0x2c, (int)asm_inthandler2c, 2 * 8, AR_INTGATE32);
33     set_gatedesc(idt + 0x40, (int)asm_hrb_api, 2 * 8, AR_INTGATE32 + 0x60);
34
35     return;
36 }

```

初始化PIC:

Listing 20: 初始化PIC

```

6 void init_pic(void)
7 {
8     io_out8(PIC0_IMR, 0xff); /* 禁止主PIC中断 */
9     io_out8(PIC1_IMR, 0xff); /* 禁止从PIC中断 */
10
11     io_out8(PIC0_ICW1, 0x11); /* 边缘触发模式 */
12     io_out8(PIC0_ICW2, 0x20); /* IRQ0-7由INT20-27接收 */
13     io_out8(PIC0_ICW3, 1 << 2); /* PIC1由IRQ2连接 */
14     io_out8(PIC0_ICW4, 0x01); /* 无缓冲区模式 */
15
16     io_out8(PIC1_ICW1, 0x11); /* 边缘触发模式 */
17     io_out8(PIC1_ICW2, 0x28); /* IRQ8-15由INT28-2f接收 */
18     io_out8(PIC1_ICW3, 2); /* PIC1由IRQ2连接 */
19     io_out8(PIC1_ICW4, 0x01); /* 无缓冲区模式 */
20
21     io_out8(PIC0_IMR, 0xfb); /* 11111011 PIC1以外全部禁止 */
22     io_out8(PIC1_IMR, 0xff); /* 11111111 禁止所有中断 */
23
24     return;
25 }

```

## 2.2.2 键盘

Listing 21: 键缓冲区结构体

```

132 struct KEYBUF {
133     unsigned char data[32];
134     int next_r, next_w, len;
135 };

```

Listing 22: PS/2 键盘中断

```

9 void inthandler21(int* esp)
10 {

```

```

11     int data;
12     io_out8(PIC0_OCW2, 0x61); /* 接收IRQ-01后通知PIC */
13     data = io_in8(PORT_KEYDAT);
14     fifo32_put(keyfifo, data + keydata0);
15     return;
16 }

```

Listing 23: 键盘初始化

```

36 void init_keyboard(fifo32* fifo, int data0)
37 {
38     /* 保存队列缓冲区信息到全局变量 */
39     keyfifo = fifo;
40     keydata0 = data0;
41     /* 初始化键盘控制电路 */
42     wait_KBC_sendready();
43     io_out8(PORT_KEYCMD, KEYCMD_WRITE_MODE);
44     wait_KBC_sendready();
45     io_out8(PORT_KEYDAT, KBC_MODE);
46     return;
47 }

```

### 2.2.3 鼠标

Listing 24: 鼠标设备结构体

```

164 typedef struct MOUSE_DEC {
165     unsigned char buf[3], phase;
166     int x, y, btn;
167 } mouse_dec;

```

Listing 25: PS/2 鼠标中断

```

9 void inthandler2c(int* esp)
10 {
11     int data;
12     io_out8(PIC1_OCW2, 0x64); /* 接收IRQ-12后通知PIC */
13     io_out8(PIC0_OCW2, 0x62); /* 接收IRQ-02后通知PIC */
14     data = io_in8(PORT_KEYDAT);
15     fifo32_put(mousefifo, data + mousedata0);
16     return;
17 }

```

因为鼠标中断是多个字节，所以需要特殊处理：

Listing 26: 鼠标中断处理

```

36 int mouse_decode(mouse_dec* mdec, unsigned char dat)
37 {
38     if (mdec->phase == 0) {
39         /* 等待鼠标的0xfa状态 */
40         if (dat == 0xfa) {
41             mdec->phase = 1;
42         }
43         return 0;
44     } else if (mdec->phase == 1) {
45         /* 等待鼠标的第1字节 */
46         if ((dat & 0xc8) == 0x08) {
47             /* 如果第1字节正确 */
48             mdec->buf[0] = dat;
49             mdec->phase = 2;
50         }
51         return 0;
52     } else if (mdec->phase == 2) {
53         /* 等待鼠标的第2字节 */
54         mdec->buf[1] = dat;
55         mdec->phase = 3;
56         return 0;
57     } else if (mdec->phase == 3) {
58         /* 等待鼠标的第3字节 */
59         mdec->buf[2] = dat;
60         mdec->phase = 1;

```



```

61     mdec->btn = mdec->buf[0] & 0x07;
62     mdec->x = mdec->buf[1];
63     mdec->y = mdec->buf[2];
64     if ((mdec->buf[0] & 0x10) != 0) {
65         mdec->x |= 0xffffffff00;
66     }
67     if ((mdec->buf[0] & 0x20) != 0) {
68         mdec->y |= 0xffffffff00;
69     }
70     mdec->y = -mdec->y; /* 鼠标的垂直方向与屏幕相反 */
71     return 1;
72 }
73 return -1;
74 }

```

## 2.2.4 屏幕

初始化一个 $6 \times 6 \times 6$ 的调色板:

Listing 27: 初始化调色板

```

7 void init_palette(void)
8 {
9     static unsigned char table_rgb[16 * 3] = {
10         0x00, 0x00, 0x00, /* base03 */
11         0x07, 0x36, 0x42, /* base02 */
12         0x58, 0x6e, 0x75, /* base01 */
13         0x65, 0x7b, 0x83, /* base00 */
14         0x83, 0x94, 0x96, /* base0 */
15         0x93, 0xa1, 0xa1, /* base1 */
16         0xee, 0xe8, 0xd5, /* base2 */
17         0xff, 0xff, 0xff, /* base3 */
18         0xfd, 0xb8, 0x13, /* yellow */
19         0xcb, 0x4b, 0x16, /* orange */
20         0xef, 0x50, 0x26, /* red */
21         0xd3, 0x36, 0x82, /* magenta */
22         0x26, 0x8b, 0xa2, /* violet */
23         0x23, 0x99, 0xd7, /* blue */
24         0x2a, 0xa1, 0x98, /* cyan */
25         0x7f, 0xbc, 0x43, /* green */
26     };
27     set_palette(0, 15, table_rgb);
28
29     unsigned char table2[6 * 6 * 6 * 3];
30     int r, g, b;
31     for (b = 0; b < 6; b++) {
32         for (g = 0; g < 6; g++) {
33             for (r = 0; r < 6; r++) {
34                 table2[(r + g * 6 + b * 6 * 6) * 3 + 0] = r * 51;
35                 table2[(r + g * 6 + b * 6 * 6) * 3 + 1] = g * 51;
36                 table2[(r + g * 6 + b * 6 * 6) * 3 + 2] = b * 51;
37             }
38         }
39     }
40     set_palette(16, 16 + 6 * 6 * 6 - 1, table2);
55     return;
56 }
57
58 void set_palette(int start, int end, unsigned char* rgb)
59 {
60     int i, eflags;
61     eflags = io_load_eflags(); /* 备份中断许可标志 */
62     io_cli(); /* 标志置0, 禁止中断 */
63     io_out8(0x03c8, start);
64     for (i = start; i <= end; i++) {
65         io_out8(0x03c9, rgb[0] / 4);
66         io_out8(0x03c9, rgb[1] / 4);
67         io_out8(0x03c9, rgb[2] / 4);
68         rgb += 3;
69     }
70     io_store_eflags(eflags); /* 复原中断许可标志 */
71     return;
72 }

```

初始化鼠标光标:

Listing 28: 初始化鼠标光标

```
140 void init_mouse_cursor8(char* mouse, char bc)
141 {
142     static char cursor[CURSOR_Y][CURSOR_X] = {
143         "          ",
144         "          ",
145         " 0         ",
146         " 00        ",
147         " 000       ",
148         " 0000      ",
149         " 00000     ",
150         " 000000    ",
151         " 0000000   ",
152         " 00000000  ",
153         " 000000000 ",
154         " 0000000000",
155         " 000000*****",
156         " 000*00     ",
157         " 00* 00*    ",
158         " 0* 00*     ",
159         "** 00*     ",
160         "      00*    ",
161         "      **     ",
162     }; /* 仿 Window 8 的鼠标指针 */
163     int x, y;
164
165     for (y = 0; y < CURSOR_Y; y++) {
166         for (x = 0; x < CURSOR_X; x++) {
167             if (cursor[y][x] == '*') {
168                 mouse[y * CURSOR_X + x] = base03;
169             }
170             if (cursor[y][x] == 'O') {
171                 mouse[y * CURSOR_X + x] = base3;
172             }
173             if (cursor[y][x] == ' ') {
174                 mouse[y * CURSOR_X + x] = bc;
175             }
176         }
177     }
178     return;
179 }
```

在屏幕上显示一个半角字符:

Listing 29: 显示字符

```
94 void putfont8(char* vram, int xsize, int x, int y, char c, char* font)
95 {
96     int i;
97     char *p, d /* data */;
98     for (i = 0; i < FNT_H; i++) {
99         p = vram + (y + i) * xsize + x;
100         d = font[i];
101         if ((d & 0x80) != 0) {
102             p[0] = c;
103         }
104         if ((d & 0x40) != 0) {
105             p[1] = c;
106         }
107         if ((d & 0x20) != 0) {
108             p[2] = c;
109         }
110         if ((d & 0x10) != 0) {
111             p[3] = c;
112         }
113         if ((d & 0x08) != 0) {
114             p[4] = c;
115         }
116         if ((d & 0x04) != 0) {
117             p[5] = c;
118         }
119         if ((d & 0x02) != 0) {
120             p[6] = c;
121         }
122     }
```

```

122         if ((d & 0x01) != 0) {
123             p[7] = c;
124         }
125     }
126     return;
127 }

```

显示字符串:

Listing 30: 显示字符串

```

129 void putfonts8_asc(char* vram, int xsize, int x, int y, char c, unsigned char* s)
130 {
131     extern char hankaku[256 * FNT_H + FNT_OFFSET];
132     char* start = hankaku + FNT_OFFSET;
133     for (; *s != 0x00; s++) {
134         putfont8(vram, xsize, x, y, c, start + *s * FNT_H);
135         x += FNT_W;
136     }
137     return;
138 }

```

## 2.2.5 窗口管理器

Listing 31: 图层管理器结构体

```

197 typedef struct SHEET {
198     unsigned char* buf;
199     int bsize, bysize, vx0, vy0, alpha, height, flags;
200     struct SHCTL* ctl;
201     struct TASK* task;
202 } sheet_t;
203
204 typedef struct SHCTL {
205     unsigned char *vram, *map;
206     int xsize, ysize, top;
207     sheet_t* sheets[MAX_SHEETS];
208     sheet_t sheets0[MAX_SHEETS];
209 } shctl_t;

```

Listing 32: 初始化图层管理器

```

7 shctl_t* shctl_init(memman_t* memman, unsigned char* vram, int xsize, int ysize)
8 {
9     shctl_t* ctl;
10     int i;
11     ctl = (shctl_t*)memman_alloc_4k(memman, sizeof(shctl_t));
12     if (ctl == 0) {
13         goto err;
14     }
15     ctl->map = (unsigned char*)memman_alloc_4k(memman, xsize * ysize);
16     if (ctl->map == 0) {
17         memman_free_4k(memman, (int)ctl, sizeof(shctl_t));
18         goto err;
19     }
20     ctl->vram = vram;
21     ctl->xsize = xsize;
22     ctl->ysize = ysize;
23     ctl->top = -1; /* 暂无图层 */
24     for (i = 0; i < MAX_SHEETS; i++) {
25         ctl->sheets0[i].flags = 0; /* 标记为未使用 */
26         ctl->sheets0[i].ctl = ctl; /* 记录所属 */
27     }
28 err:
29     return ctl;
30 }

```

Listing 33: 为新图层分配内存

```

32 sheet_t* sheet_alloc(shctl_t* ctl)
33 {
34     sheet_t* sht;

```

```

35     int i;
36     for (i = 0; i < MAX_SHEETS; i++) {
37         if (ctl->sheets0[i].flags == 0) {
38             sht = &ctl->sheets0[i];
39             sht->flags = SHEET_USE; /* 标记为使用中 */
40             sht->height = -1; /* 隐藏 */
41             sht->task = 0; /* 不使用自动关闭功能 */
42             return sht;
43         }
44     }
45     return 0; /* 所有图层都在使用中 */
46 }

```

重绘一个图层相对比较麻烦，需要处理透明色，以及一个小的优化：

Listing 34: 重绘图层

```

57 void sheet_refreshmap(shtctl_t* ctl, int vx0, int vy0, int vx1, int vy1, int h0)
58 {
59     int h, bx, by, vx, vy, bx0, by0, bx1, by1, sid4, *p;
60     unsigned char *buf, sid, *map = ctl->map;
61     sheet_t* sht;
62     if (vx0 < 0) {
63         vx0 = 0;
64     }
65     if (vy0 < 0) {
66         vy0 = 0;
67     }
68     if (vx1 > ctl->xsize) {
69         vx1 = ctl->xsize;
70     }
71     if (vy1 > ctl->ysize) {
72         vy1 = ctl->ysize;
73     }
74     for (h = h0; h <= ctl->top; h++) {
75         sht = ctl->sheets[h];
76         sid = sht - ctl->sheets0; /* 地址相减得到图层号 */
77         buf = sht->buf;
78         bx0 = vx0 - sht->vx0;
79         by0 = vy0 - sht->vy0;
80         bx1 = vx1 - sht->vx0;
81         by1 = vy1 - sht->vy0;
82         if (bx0 < 0) {
83             bx0 = 0;
84         }
85         if (by0 < 0) {
86             by0 = 0;
87         }
88         if (bx1 > sht->bysize) {
89             bx1 = sht->bysize;
90         }
91         if (by1 > sht->bysize) {
92             by1 = sht->bysize;
93         }
94         if (sht->alpha == -1) {
95             if ((sht->vx0 & 3) == 0 && (bx0 & 3) == 0 && (bx1 & 3) == 0) {
96                 /* 无透明色专用的高速版 (4字节型) */
97                 bx1 = (bx1 - bx0) / 4; /* MOV次数 */
98                 sid4 = sid | sid << 8 | sid << 16 | sid << 24;
99                 for (by = by0; by < by1; by++) {
100                     vy = sht->vy0 + by;
101                     vx = sht->vx0 + bx0;
102                     p = (int*)&map[vy * ctl->xsize + vx];
103                     for (bx = 0; bx < bx1; bx++) {
104                         p[bx] = sid4;
105                     }
106                 }
107             } else {
108                 /* 无透明色专用的高速版 (1字节型) */
109                 for (by = by0; by < by1; by++) {
110                     vy = sht->vy0 + by;
111                     for (bx = bx0; bx < bx1; bx++) {
112                         vx = sht->vx0 + bx;
113                         map[vy * ctl->xsize + vx] = sid;
114                     }
115                 }
116             }

```

```

117     } else {
118         /* 有透明色的一般版 */
119         for (by = by0; by < by1; by++) {
120             vy = sht->vy0 + by;
121             for (bx = bx0; bx < bx1; bx++) {
122                 vx = sht->vx0 + bx;
123                 if (buf[by * sht->bysize + bx] != sht->alpha) {
124                     map[vy * ctl->xsize + vx] = sid;
125                 }
126             }
127         }
128     }
129 }
130 return;
131 }
293 void sheet_refresh(sheet_t* sht, int bx0, int by0, int bx1, int by1)
294 {
295     if (sht->height >= 0) { /* 如果可视则刷新画面 */
296         sheet_refreshsub(sht->ctl, sht->vx0 + bx0, sht->vy0 + by0, sht->vx0 + bx1, sht->vy0 + by1, sht->height, s
297     }
298     return;
299 }

```

改变图层层次:

Listing 35: 改变图层层次

```

233 void sheet_updown(sheet_t* sht, int height)
234 {
235     shtctl_t* ctl = sht->ctl;
236     int h, old = sht->height; /* 备份层高 */
237
238     /* 修正层高 */
239     if (height > ctl->top + 1) {
240         height = ctl->top + 1;
241     }
242     if (height < -1) {
243         height = -1;
244     }
245     sht->height = height; /* 设置层高 */
246
247     /* 重新排列 sheets[] */
248     if (old > height) { /* 比以前低 */
249         if (height >= 0) {
250             /* 中间图层上升 */
251             for (h = old; h > height; h--) {
252                 ctl->sheets[h] = ctl->sheets[h - 1];
253                 ctl->sheets[h]->height = h;
254             }
255             ctl->sheets[height] = sht;
256             sheet_refreshmap(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, height + 1);
257             sheet_refreshsub(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, height + 1,
258         } else { /* 隐藏 */
259             if (ctl->top > old) {
260                 /* 上面图层下降 */
261                 for (h = old; h < ctl->top; h++) {
262                     ctl->sheets[h] = ctl->sheets[h + 1];
263                     ctl->sheets[h]->height = h;
264                 }
265             }
266             ctl->top--; /* 显示中的图层减少，最高层下降 */
267             sheet_refreshmap(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, 0);
268             sheet_refreshsub(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, 0, old - 1);
269         }
270     } else if (old < height) { /* 比以前高 */
271         if (old >= 0) {
272             /* 中间图层下降 */
273             for (h = old; h < height; h++) {
274                 ctl->sheets[h] = ctl->sheets[h + 1];
275                 ctl->sheets[h]->height = h;
276             }
277             ctl->sheets[height] = sht;
278         } else { /* 显示 */
279             /* 上面图层上升 */
280             for (h = ctl->top; h >= height; h--) {
281                 ctl->sheets[h + 1] = ctl->sheets[h];
282                 ctl->sheets[h + 1]->height = h + 1;

```

```

283     }
284     ctl->sheets[height] = sht;
285     ctl->top++; /* 显示中的图层增加，最高层上升 */
286 }
287 sheet_refreshmap(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, height);
288 sheet_refreshsub(ctl, sht->vx0, sht->vy0, sht->vx0 + sht->bysize, sht->vy0 + sht->bysize, height, height);
289 }
290 return;
291 }
292
293 void sheet_slide(sheet_t* sht, int vx0, int vy0)
294 {
295     shtctl_t* ctl = sht->ctl;
296     int old_vx0 = sht->vx0, old_vy0 = sht->vy0;
297     sht->vx0 = vx0;
298     sht->vy0 = vy0;
299     if (sht->height >= 0) { /* 如果可视则刷新画面 */
300         sheet_refreshmap(ctl, old_vx0, old_vy0, old_vx0 + sht->bysize, old_vy0 + sht->bysize, 0);
301         sheet_refreshmap(ctl, vx0, vy0, vx0 + sht->bysize, vy0 + sht->bysize, sht->height);
302         sheet_refreshsub(ctl, old_vx0, old_vy0, old_vx0 + sht->bysize, old_vy0 + sht->bysize, 0, sht->height - 1);
303         sheet_refreshsub(ctl, vx0, vy0, vx0 + sht->bysize, vy0 + sht->bysize, sht->height, sht->height);
304     }
305     return;
306 }
307
308 }

```

创建新窗口：

Listing 36: 创建新窗口

```

6 void make_window8(unsigned char* buf, int xsize, int ysize, char* title, char act)
7 {
8     boxfill8(buf, xsize, base01, 0, 0, xsize - 1, ysize - 1);
9     boxfill8(buf, xsize, base2, 1, 21, xsize - 2, ysize - 2);
10    make_wtitle8(buf, xsize, title, act);
11    return;
12 }
13
14 void make_wtitle8(unsigned char* buf, int xsize, char* title, char act)
15 {
16     static char closebtn[7][8] = {
17         "oo  oo",
18         " oo  oo ",
19         "  oooo  ",
20         "   oo   ",
21         "  oooo  ",
22         " oo  oo ",
23         "oo  oo",
24     }; /* 仿 Windows 8 关闭按钮 */
25     int x, y;
26     char c;
27     boxfill8(buf, xsize, act ? blue : base1, 1, 1, xsize - 2, 20);
28     boxfill8(buf, xsize, act ? orange : base00, xsize - 30, 1, xsize - 2, 18);
29     for (y = 0; y < 7; y++) {
30         for (x = 0; x < 8; x++) {
31             c = closebtn[y][x];
32             if (c == 'o') {
33                 buf[(7 + y) * xsize + (xsize - 19 + x)] = base3;
34             }
35         }
36     }
37     putfonts8_asc(buf, xsize, (xsize - strlen(title) * FNT_W) / 2, (22 - FNT_H) / 2, base3, title);
38     return;
39 }

```

改变活动窗口和不活动窗口的状态：

Listing 37: 改变窗口状态

```

57 void change_wtitle8(sheet_t* sht, char act)
58 {
59     int x, y, xsize = sht->bysize;
60     char c, tc_new, tbc_new, tc_old, tbc_old, *buf = sht->buf;
61     if (act != 0) {
62         tc_new = blue;
63         tbc_new = orange;
64         tc_old = base1;

```

```

65     tbc_old = base00;
66 } else {
67     tc_new = base1;
68     tbc_new = base00;
69     tc_old = blue;
70     tbc_old = orange;
71 }
72 for (y = 0; y <= 21; y++) {
73     for (x = 0; x < xsize; x++) {
74         c = buf[y * xsize + x];
75         if (c == tc_old) {
76             c = tc_new;
77         } else if (c == tbc_old) {
78             c = tbc_new;
79         }
80         buf[y * xsize + x] = c;
81     }
82 }
83 sheet_refresh(sht, 0, 0, xsize, 21);
84 return;
85 }

```

## 2.3 进程管理

Listing 38: 进程相关结构体

```

252 typedef struct TSS32 {
253     int backlink, esp0, ss0, esp1, ss1, esp2, ss2, cr3;
254     int eip, eflags, eax, ecx, edx, ebx, esp, ebp, esi, edi;
255     int es, cs, ss, ds, fs, gs;
256     int ldt, iomap;
257 } tss32;
258
259 typedef struct TASK {
260     int sel, flags; /* sel 存放 GDT 的编号 */
261     int level, priority; /* 优先级 */
262     fifo32 fifo;
263     tss32 tss;
264     struct SEGMENT_DESCRIPTOR ldt[2];
265     struct CONSOLE* cons;
266     int ds_base;
267     int cons_stack;
268     struct FILEHANDLE *fhandle;
269     int *fat;
270     char *cmdline;
271 } task_t;
272
273 typedef struct TASKLEVEL {
274     int running; /* 运行中任务数 */
275     int now; /* 当前运行中任务 */
276     task_t* tasks[MAX_TASKS_LV];
277 } tasklevel_t;
278
279 typedef struct TASKCTL {
280     int now_lv; /* 活动中的等级 */
281     char lv_change; /* 下次是否改变等级 */
282     tasklevel_t level[MAX_TASKLEVELS];
283     task_t tasks0[MAX_TASKS];
284 } taskctl_t;

```

Listing 39: 新建进程

```

8 task_t* task_now(void)
9 {
10     tasklevel_t* tl = &taskctl->level[taskctl->now_lv];
11     return tl->tasks[tl->now];
12 }
13
14 void task_add(task_t* task)
15 {
16     tasklevel_t* tl = &taskctl->level[task->level];
17     tl->tasks[tl->running] = task;
18     tl->running++;

```

```

19     task->flags = 2; /* 活动中 */
20     return;
21 }

```

Listing 40: 移除进程

```

23 void task_remove(struct TASK* task)
24 {
25     int i;
26     tasklevel_t* tl = &taskctl->level[task->level];
27
28     /* task在哪 */
29     for (i = 0; i < tl->running; i++) {
30         if (tl->tasks[i] == task) {
31             /* 在这 */
32             break;
33         }
34     }
35
36     tl->running--;
37     if (i < tl->now) {
38         tl->now--; /* 移动处理 */
39     }
40     if (tl->now >= tl->running) {
41         /* now修正 */
42         tl->now = 0;
43     }
44     task->flags = 1; /* 休眠中 */
45
46     /* 移动 */
47     for (; i < tl->running; i++) {
48         tl->tasks[i] = tl->tasks[i + 1];
49     }
50
51     return;
52 }

```

Listing 41: 切换进程

```

54 void task_switchsub(void)
55 {
56     int i;
57     /* 找最上层 */
58     for (i = 0; i < MAX_TASKLEVELS; i++) {
59         if (taskctl->level[i].running > 0) {
60             break; /* 找到了 */
61         }
62     }
63     taskctl->now_lv = i;
64     taskctl->lv_change = 0;
65     return;
66 }
161
162 void task_switch(void)
163 {
164     tasklevel_t* tl = &taskctl->level[taskctl->now_lv];
165     task_t *new_task, *now_task = tl->tasks[tl->now];
166     tl->now++;
167     if (tl->now == tl->running) {
168         tl->now = 0;
169     }
170     if (taskctl->lv_change != 0) {
171         task_switchsub();
172         tl = &taskctl->level[taskctl->now_lv];
173     }
174     new_task = tl->tasks[tl->now];
175     timer_settime(task_timer, new_task->priority);
176     if (new_task != now_task) {
177         farjmp(0, new_task->sel);
178     }
179     return;
180 }

```

Listing 42: 进程休眠

```

68 void task_idle(void)

```



```

69 {
70     for (;;) {
71         io_hlt();
72     }
73 }
181
182 void task_sleep(task_t* task)
183 {
184     task_t* now_task;
185     if (task->flags == 2) {
186         /* 活动中 */
187         now_task = task_now();
188         task_remove(task); /* flags变1 */
189         if (task == now_task) {
190             /* 如果是让自己休眠，则需要切换任务 */
191             task_switchsub();
192             now_task = task_now(); /* 设置后获取当前任务值 */
193             farjmp(0, now_task->sel);
194         }
195     }
196 }

```

Listing 43: 初始化进程控制块

```

75 task_t* task_init(memman_t* memman)
76 {
77     int i;
78     task_t *task, *idle;
79     segment_descriptor* gdt = (segment_descriptor*)ADR_GDT;
80     taskctl = (taskctl_t*)memman_alloc_4k(memman, sizeof(taskctl_t));
81     for (i = 0; i < MAX_TASKS; i++) {
82         taskctl->tasks0[i].flags = 0;
83         taskctl->tasks0[i].sel = (TASK_GDT0 + i) * 8;
84         taskctl->tasks0[i].tss.ldtr = (TASK_GDT0 + MAX_TASKS + i) * 8;
85         set_segmdesc(gdt + TASK_GDT0 + i, 103, (int)&taskctl->tasks0[i].tss, AR_TSS32);
86         set_segmdesc(gdt + TASK_GDT0 + MAX_TASKS + i, 15, (int)taskctl->tasks0[i].ldt, AR_LDT);
87     }
88     task = task_alloc();
89     task->flags = 2; /* 活动中标志 */
90     task->priority = 2; /* 0.02s */
91     task->level = 0; /* 最高等级 */
92     task_add(task);
93     task_switchsub(); /* 设置等级 */
94     load_tr(task->sel);
95     task_timer = timer_alloc();
96     timer_settime(task_timer, task->priority);
97
98     idle = task_alloc();
99     idle->tss.esp = memman_alloc_4k(memman, 64 * 1024) + 64 * 1024;
100     idle->tss.eip = (int)&task_idle;
101     idle->tss.es = 1 * 8;
102     idle->tss.cs = 2 * 8;
103     idle->tss.ss = 1 * 8;
104     idle->tss.ds = 1 * 8;
105     idle->tss.fs = 1 * 8;
106     idle->tss.gs = 1 * 8;
107     task_run(idle, MAX_TASKLEVELS - 1, 1);
108
109     return task;
110 }

```

Listing 44: 分配进程控制块内存

```

112 task_t* task_alloc(void)
113 {
114     int i;
115     task_t* task;
116     for (i = 0; i < MAX_TASKS; i++) {
117         if (taskctl->tasks0[i].flags == 0) {
118             task = &taskctl->tasks0[i];
119             task->flags = 1; /* 使用中标志 */
120             task->tss.eflags = 0x00000202; /* IF = 1; */
121             task->tss.eax = 0; /* 先置为0 */
122             task->tss.ecx = 0;
123             task->tss.edx = 0;
124             task->tss.ebx = 0;
125             task->tss.ebp = 0;

```

```

126         task->tss.esi = 0;
127         task->tss.edi = 0;
128         task->tss.es = 0;
129         task->tss.ds = 0;
130         task->tss.fs = 0;
131         task->tss.gs = 0;
132         task->tss.iomap = 0x40000000;
133         task->tss.ss0 = 0;
134         return task;
135     }
136 }
137 return 0; /* 全部使用中 */
138 }

```

Listing 45: 执行进程

```

140 void task_run(task_t* task, int level, int priority)
141 {
142     if (level < 0) {
143         level = task->level; /* 等级不变 */
144     }
145     if (priority > 0) {
146         task->priority = priority;
147     }
148
149     if (task->flags == 2 && task->level != level) { /* 活动中等级变更 */
150         task_remove(task); /* 执行后 flags 变 1, 可以执行下面的 */
151     }
152     if (task->flags != 2) {
153         /* 从休眠唤醒 */
154         task->level = level;
155         task_add(task);
156     }
157
158     taskctl->lv_change = 1; /* 下次任务切换时要检查等级 */
159     return;
160 }

```

## 2.4 内存管理

Listing 46: 内存块结构体

```

177 typedef struct FREEINFO { /* 空闲块 */
178     unsigned int addr, size;
179 } freeinfo_t;

```

Listing 47: 内存管理器结构体

```

181 typedef struct MEMMAN { /* 内存管理 */
182     int frees, maxfrees, lostsize, losts;
183     freeinfo_t free[MEMMAN_FREES];
184 } memman_t;

```

Listing 48: 初始化内存管理器

```

41 void memman_init(memman_t* man)
42 {
43     man->frees = 0; /* 空闲块数 */
44     man->maxfrees = 0; /* 用于观察可用状况 */
45     man->lostsize = 0; /* 释放失败的内存大小总和 */
46     man->losts = 0; /* 释放失败的次数 */
47     return;
48 }

```

Listing 49: 空闲内存总大小

```

51 unsigned int memman_total(memman_t* man)
52 {
53     unsigned int i, t = 0;
54     for (i = 0; i < man->frees; i++) {
55         t += man->free[i].size;
56     }
57     return t;
58 }

```

Listing 50: 分配内存块

```

61 unsigned int memman_alloc(memman_t* man, unsigned int size)
62 {
63     unsigned int i, a;
64     for (i = 0; i < man->frees; i++) {
65         if (man->free[i].size >= size) {
66             /* 找到了足够大的空闲块 */
67             a = man->free[i].addr;
68             man->free[i].addr += size;
69             man->free[i].size -= size;
70             if (man->free[i].size == 0) {
71                 /* 如果free[i]变成0就减掉一个空闲块 */
72                 man->frees--;
73                 for (; i < man->frees; i++) {
74                     man->free[i] = man->free[i + 1]; /* 结构体赋值 */
75                 }
76             }
77             return a;
78         }
79     }
80     return 0; /* 没有可用空间 */
81 }

```

Listing 51: 释放内存块

```

84 int memman_free(memman_t* man, unsigned int addr, unsigned int size)
85 {
86     int i, j;
87     /* 为便于合并内存, 将free[]按照addr顺序排列 */
88     /* 所以先决定应该放在哪里 */
89     for (i = 0; i < man->frees; i++) {
90         if (man->free[i].addr > addr) {
91             break;
92         }
93     }
94     /* free[i - 1].addr < addr < free[i].addr */
95     if (i > 0) {
96         /* 前面有空闲块 */
97         if (man->free[i - 1].addr + man->free[i - 1].size == addr) {
98             /* 与前面合并 */
99             man->free[i - 1].size += size;
100             if (i < man->frees) {
101                 /* 后面还有 */
102                 if (addr + size == man->free[i].addr) {
103                     /* 与后面合并 */
104                     man->free[i - 1].size += man->free[i].size;
105                     /* 移除man->free[i] */
106                     /* free[i]变0后合并到前面 */
107                     man->frees--;
108                     for (; i < man->frees; i++) {
109                         man->free[i] = man->free[i + 1]; /* 结构体赋值 */
110                     }
111                 }
112             }
113             return 0; /* 成功完成 */
114         }
115     }
116     /* 不能与前面的空闲块合并 */
117     if (i < man->frees) {
118         /* 后面还有 */
119         if (addr + size == man->free[i].addr) {
120             /* 与后面合并 */
121             man->free[i].addr = addr;
122             man->free[i].size += size;
123             return 0; /* 成功完成 */
124         }
125     }
126     /* 既不能与前面合并, 也不能与后面合并 */
127     if (man->frees < MEMMAN_FREES) {
128         /* free[i]之后的向后移动一些距离来腾出空间 */
129         for (j = man->frees; j > i; j--) {
130             man->free[j] = man->free[j - 1];
131         }
132         man->frees++;
133         if (man->maxfrees < man->frees) {
134             man->maxfrees = man->frees; /* 更新最大值 */
135         }
136     }
137 }

```

```

136     man->free[i].addr = addr;
137     man->free[i].size = size;
138     return 0; /* 成功完成 */
139 }
140 /* 不能往后移动 */
141 man->losts++;
142 man->lostsize += size;
143 return -1; /* 失败 */
144 }

```

## 2.5 文件管理

Listing 52: 文件结构体

```

346 typedef struct FILEINFO {
347     unsigned char name[8], ext[3], type;
348     char reserve[10];
349     unsigned short time, date, clustno;
350     unsigned int size;
351 } fileinfo;

```

Listing 53: 读入FAT表

```

6 void file_readfat(int* fat, unsigned char* img)
7 {
8     int i, j = 0;
9     for (i = 0; i < 2880; i += 2) {
10         fat[i + 0] = (img[j + 0] | img[j + 1] << 8) & 0xffff;
11         fat[i + 1] = (img[j + 1] >> 4 | img[j + 2] << 4) & 0xffff;
12         j += 3;
13     }
14     return;
15 }

```

Listing 54: 读入一个文件

```

17 void file_loadfile(int clustno, int size, char* buf, int* fat, char* img)
18 {
19     int i;
20     for (;;) {
21         if (size <= 512) {
22             for (i = 0; i < size; i++) {
23                 buf[i] = img[clustno * 512 + i];
24             }
25             break;
26         }
27         for (i = 0; i < 512; i++) {
28             buf[i] = img[clustno * 512 + i];
29         }
30         size -= 512;
31         buf += 512;
32         clustno = fat[clustno];
33     }
34     return;
35 }

```

Listing 55: 查找指定文件

```

37 fileinfo* file_search(char* name, fileinfo* finfo, int max)
38 {
39     int i, j;
40     char s[12];
41     for (j = 0; j < 11; j++) {
42         s[j] = '.';
43     }
44     j = 0;
45     for (i = 0; name[i] != 0; i++) {
46         if (j >= 11) {
47             return 0; /* 找不到 */
48         }
49         if (name[i] == '.' && j <= 8) {
50             j = 8;

```

```

51     } else {
52         s[j] = name[i];
53         if ('a' <= s[j] && s[j] <= 'z') {
54             /* 转成大写 */
55             s[j] -= 0x20;
56         }
57         j++;
58     }
59 }
60 for (i = 0; i < max; i) {
61     if (finfo[i].name[0] == 0x00) {
62         break;
63     }
64     if ((finfo[i].type & 0x18) == 0) {
65         for (j = 0; j < 11; j++) {
66             if (finfo[i].name[j] != s[j]) {
67                 goto next;
68             }
69         }
70         return finfo + i; /* 找到文件 */
71     }
72     next:
73     i++;
74 }
75 return 0; /* 未找到文件 */
76 }

```

## 2.6 系统接口

Listing 56: 系统接口声明

```

1 void api_putchar(int c);
2 void api_putstr0(char* s);
3 void api_putstr1(char* s, int l);
4 void api_end(void);
5 int api_openwin(char* buf, int xsiz, int ysiz, int col_inv, char* title);
6 void api_putstrwin(int win, int x, int y, int col, int len, char* str);
7 void api_boxfilwin(int win, int x0, int y0, int x1, int y1, int col);
8 void api_initmalloc(void);
9 char* api_malloc(int size);
10 void api_free(char* addr, int size);
11 void api_point(int win, int x, int y, int col);
12 void api_refreshwin(int win, int x0, int y0, int x1, int y1);
13 void api_linewin(int win, int x0, int y0, int x1, int y1, int col);
14 void api_closewin(int win);
15 int api_getkey(int mode);
16 int api_alloctimer(void);
17 void api_inittimer(int timer, int data);
18 void api_settimer(int timer, int time);
19 void api_freetimer(int timer);
20 void api_beep(int tone);
21 int api_fopen(char* fname);
22 void api_fclose(int fhandle);
23 void api_fseek(int fhandle, int offset, int mode);
24 int api_fsize(int fhandle, int mode);
25 int api_fread(char* buf, int maxsize, int fhandle);
26 int api_cmdline(char* buf, int maxsize);

```

Listing 57: 系统接口

```

395 int* hrb_api(int edi, int esi, int ebp, int esp, int ebx, int edx, int ecx, int eax)
396 {
397     task_t* task = task_now();
398     int ds_base = task->ds_base;
399     console* cons = task->cons;
400     shtctl_t* shtctl = (shtctl_t*)((int*)0x0fe4);
401     sheet_t* sht;
402     fifo32* sys_fifo = (fifo32*)((int*)0x0fec);
403     int* reg = &eax + 1; /* eax后面的地址 */
404     /* 强行政写通过PUSHAD保存的值 */
405     /* reg[0] : EDI,   reg[1] : ESI,   reg[2] : EBP,   reg[3] : ESP */
406     /* reg[4] : EBX,   reg[5] : EDX,   reg[6] : ECX,   reg[7] : EAX */
407     int i;

```

```

408     fileinfo* finfo;
409     filehandle* fh;
410     memman_t* memman = (memman_t*)MEMMAN_ADDR;
411
412     if (edx == 1) {
413         cons_putchar(cons, eax & 0xff, 1);
414     } else if (edx == 2) {
415         cons_putstr0(cons, (char*)ebx + ds_base);
416     } else if (edx == 3) {
417         cons_putstr1(cons, (char*)ebx + ds_base, ecx);
418     } else if (edx == 4) {
419         return &(task->tss.esp0);
420     } else if (edx == 5) {
421         sht = sheet_alloc(shtctl);
422         sht->task = task;
423         sht->flags |= 0x10;
424         sheet_setbuf(sht, (char*)ebx + ds_base, esi, edi, eax);
425         make_window8((char*)ebx + ds_base, esi, edi, (char*)ecx + ds_base, 0);
426         sheet_slide(sht, (shtctl->xsize - esi) / 2, (shtctl->ysize - edi) / 2);
427         sheet_updown(sht, shtctl->top);
428         reg[7] = (int)sht;
429     } else if (edx == 6) {
430         sht = (sheet_t*)(ebx & 0xfffffffffe);
431         putfonts8_asc(sht->buf, sht->bysize, esi, edi, eax, (char*)ebp + ds_base);
432         if ((ebx & 1) == 0) {
433             sheet_refresh(sht, esi, edi, esi + ecx * FNT_W, edi + FNT_H);
434         }
435     } else if (edx == 7) {
436         sht = (sheet_t*)(ebx & 0xfffffffffe);
437         boxfill8(sht->buf, sht->bysize, ebp, eax, ecx, esi, edi);
438         if ((ebx & 1) == 0) {
439             sheet_refresh(sht, eax, ecx, esi + 1, edi + 1);
440         }
441     } else if (edx == 8) {
442         memman_init((memman_t*)(ebx + ds_base));
443         ecx &= 0xfffffffff0; /* 以16字节为单位 */
444         memman_free((memman_t*)(ebx + ds_base), eax, ecx);
445     } else if (edx == 9) {
446         ecx = (ecx + 0x0f) & 0xfffffffff0; /* 以16为单位向上取整 */
447         reg[7] = memman_alloc((memman_t*)(ebx + ds_base), ecx);
448     } else if (edx == 10) {
449         ecx = (ecx + 0x0f) & 0xfffffffff0; /* 以16字节为单位向上取整 */
450         memman_free((memman_t*)(ebx + ds_base), eax, ecx);
451     } else if (edx == 11) {
452         sht = (sheet_t*)(ebx & 0xfffffffffe);
453         sht->buf[sht->bysize * edi + esi] = eax;
454         if ((ebx & 1) == 0) {
455             sheet_refresh(sht, esi, edi, esi + 1, edi + 1);
456         }
457     } else if (edx == 12) {
458         sht = (sheet_t*)ebx;
459         sheet_refresh(sht, eax, ecx, esi, edi);
460     } else if (edx == 13) {
461         sht = (sheet_t*)(ebx & 0xfffffffffe);
462         hrb_api_linewin(sht, eax, ecx, esi, edi, ebp);
463         if ((ebx & 1) == 0) {
464             sheet_refresh(sht, eax, ecx, esi + 1, edi + 1);
465         }
466     } else if (edx == 14) {
467         sheet_free((sheet_t*)ebx);
468     } else if (edx == 15) {
469         for (;;) {
470             io_cli();
471             if (fifo32_status(&task->fifo) == 0) {
472                 if (eax != 0) {
473                     task_sleep(task); /* FIFO为空时休眠并等待 */
474                 } else {
475                     io_sti();
476                     reg[7] = -1;
477                     return 0;
478                 }
479             }
480             i = fifo32_get(&task->fifo);
481             io_sti();
482             if (i <= 1) { /* 光标用计时器 */
483                 /* 应用程序运行时并不需要显示光标 */
484                 timer_init(cons->timer, &task->fifo, 1); /* 置为1 */

```

```

485         timer_settime(cons->timer, 50);
486     }
487     if (i == 2) { /* 光标显示 */
488         cons->cur_c = base3;
489     }
490     if (i == 3) { /* 光标隐藏 */
491         cons->cur_c = -1;
492     }
493     if (i == 4) { /* 只关闭终端窗口 */
494         timer_cancel(cons->timer);
495         io_cli();
496         fifo32_put(sys_fifo, cons->sht - shtctl->sheets0 + 2024); /* 2024 ■ 2279 */
497         cons->sht = 0;
498         io_sti();
499     }
500     if (256 <= i) { /* 通过任务A接收的键盘数据 */
501         reg[7] = i - 256;
502         return 0;
503     }
504 }
505 } else if (edx == 16) {
506     reg[7] = (int)timer_alloc();
507     ((timer_t*)reg[7])->flags2 = 1; /* 允许自动取消 */
508 } else if (edx == 17) {
509     timer_init((timer_t*)ebx, &task->fifo, eax + 256);
510 } else if (edx == 18) {
511     timer_settime((timer_t*)ebx, eax);
512 } else if (edx == 19) {
513     timer_free((timer_t*)ebx);
514 } else if (edx == 20) {
515     if (eax == 0) {
516         i = io_in8(0x61);
517         io_out8(0x61, i & 0x0d);
518     } else {
519         i = 1193180000 / eax;
520         io_out8(0x43, 0xb6);
521         io_out8(0x42, i & 0xff);
522         io_out8(0x42, i >> 8);
523         i = io_in8(0x61);
524         io_out8(0x61, (i | 0x03) & 0x0f);
525     }
526 } else if (edx == 21) {
527     for (i = 0; i < 8; i++) {
528         if (task->fhandle[i].buf == 0) {
529             break;
530         }
531     }
532     fh = &task->fhandle[i];
533     reg[7] = 0;
534     if (i < 8) {
535         finfo = file_search((char*)ebx + ds_base,
536                             (fileinfo*)(ADR_DISKIMG + 0x002600), 224);
537         if (finfo != 0) {
538             reg[7] = (int)fh;
539             fh->buf = (char*)memman_alloc_4k(memman, finfo->size);
540             fh->size = finfo->size;
541             fh->pos = 0;
542             file_loadfile(finfo->clustno, finfo->size, fh->buf, task->fat, (char*)(ADR_DISKIMG + 0x003e00));
543         }
544     }
545 } else if (edx == 22) {
546     fh = (filehandle*)eax;
547     memman_free_4k(memman, (int)fh->buf, fh->size);
548     fh->buf = 0;
549 } else if (edx == 23) {
550     fh = (filehandle*)eax;
551     if (ecx == 0) {
552         fh->pos = ebx;
553     } else if (ecx == 1) {
554         fh->pos += ebx;
555     } else if (ecx == 2) {
556         fh->pos = fh->size + ebx;
557     }
558     if (fh->pos < 0) {
559         fh->pos = 0;
560     }
561     if (fh->pos > fh->size) {

```

```

562         fh->pos = fh->size;
563     }
564 } else if (edx == 24) {
565     fh = (filehandle*)eax;
566     if (ecx == 0) {
567         reg[7] = fh->size;
568     } else if (ecx == 1) {
569         reg[7] = fh->pos;
570     } else if (ecx == 2) {
571         reg[7] = fh->pos - fh->size;
572     }
573 } else if (edx == 25) {
574     fh = (filehandle*)eax;
575     for (i = 0; i < ecx; i++) {
576         if (fh->pos == fh->size) {
577             break;
578         }
579         *((char*)ebx + ds_base + i) = fh->buf[fh->pos];
580         fh->pos++;
581     }
582     reg[7] = i;
583 } else if (edx == 26) {
584     i = 0;
585     for (;;) {
586         *((char *) ebx + ds_base + i) = task->cmdline[i];
587         if (task->cmdline[i] == 0) {
588             break;
589         }
590         if (i >= ecx) {
591             break;
592         }
593         i++;
594     }
595     reg[7] = i;
596 }
597 return 0;
598 }

```

## 2.6.1 终端

Listing 58: 终端结构体

```

316 typedef struct CONSOLE {
317     sheet_t* sht;
318     int cur_x, cur_y, cur_c;
319     timer_t* timer;
320 } console;

```

Listing 59: 启动一个终端

```

7 void console_task(sheet_t* sheet, unsigned int memtotal)
8 {
9     task_t* task = task_now();
10    memman_t* memman = (memman_t*)MEMMAN_ADDR;
11    int i, *fat = (int *)memman_alloc_4k(memman, 4 * 2880);
12    console cons;
13    char cmdline[CONS_COLN];
14    filehandle fhandle[8];
15
16    cons.sht = sheet;
17    cons.cur_x = CONS_LEFT;
18    cons.cur_y = CONS_TOP;
19    cons.cur_c = -1;
20    task->cons = &cons;
21    task->cmdline = cmdline;
22
23    if (cons.sht != 0) {
24        cons.timer = timer_alloc();
25        timer_init(cons.timer, &task->fifo, 1);
26        timer_settime(cons.timer, 50);
27    }
28    file_readfat(fat, (unsigned char*)(ADR_DISKIMG + 0x000200));
29    for (i = 0; i < 8; i++) {
30        fhandle[i].buf = 0; /* 未使用 */

```



```

31     }
32     task->fhandle = fhandle;
33     task->fat = fat;
34
35     /* 命令提示符 */
36     cons_putchar(&cons, '$', 1);
37     cons_putchar(&cons, ' ', 1);
38
39     for (;;) {
40         io_cli();
41         if (fifo32_status(&task->fifo) == 0) {
42             task_sleep(task);
43             io_sti();
44         } else {
45             i = fifo32_get(&task->fifo);
46             io_sti();
47             if (i <= 1 && cons.sht != 0) { /* 光标闪烁 */
48                 if (i != 0) {
49                     timer_init(&cons.timer, &task->fifo, 0);
50                     if (cons.cur_c >= 0) {
51                         cons.cur_c = base3;
52                     }
53                 } else {
54                     timer_init(&cons.timer, &task->fifo, 1);
55                     if (cons.cur_c >= 0) {
56                         cons.cur_c = base03;
57                     }
58                 }
59                 timer_settime(&cons.timer, 50);
60             }
61             if (i == 2) { /* 光标ON */
62                 cons.cur_c = base3;
63             }
64             if (i == 3) { /* 光标OFF */
65                 if (cons.sht != 0) {
66                     boxfill8(cons.sht->buf, cons.sht->bysize, base03, cons.cur_x, cons.cur_y, cons.cur_x + FNT_W
67                 )
68                 }
69                 cons.cur_c = -1;
70             }
71             if (i == 4) { /* 关闭窗口 */
72                 cmd_exit(&cons, fat);
73             }
74             if (256 <= i && i <= 511) { /* 键盘数据 (从任务A) */
75                 if (i == 8 + 256) {
76                     /* 退格键 */
77                     if (cons.cur_x > CONS_LEFT + FNT_W * 2) {
78                         /* 擦除光标, 前移一位 */
79                         cons_putchar(&cons, ' ', 0);
80                         cons.cur_x -= FNT_W;
81                     }
82                 } else if (i == 10 + 256) {
83                     /* Enter */
84                     /* 擦除光标, 换行 */
85                     cons_putchar(&cons, ' ', 0);
86                     cmdline[(cons.cur_x - CONS_LEFT) / FNT_W - 2] = 0;
87                     cons_newline(&cons);
88                     cons_runcmd(cmdline, &cons, fat, memtotal); /* 执行命令 */
89                     if (cons.sht == 0) {
90                         cmd_exit(&cons, fat);
91                     }
92                     /* 命令提示符 */
93                     cons_putchar(&cons, '$', 1);
94                     cons_putchar(&cons, ' ', 1);
95                 } else {
96                     /* 一般字符 */
97                     if (cons.cur_x < CONS_LEFT + CONS_COLW - FNT_W) {
98                         /* 显示字符, 后移一位 */
99                         cmdline[(cons.cur_x - CONS_LEFT) / FNT_W - 2] = i - 256;
100                         cons_putchar(&cons, i - 256, 1);
101                     }
102                 }
103             }
104             /* 重新显示光标 */
105             if (cons.sht != 0) {
106                 if (cons.cur_c >= 0) {
107                     boxfill8(cons.sht->buf, cons.sht->bysize, cons.cur_c, cons.cur_x, cons.cur_y, cons.cur_x + FNT_W

```

```

108         sheet_refresh(cons.sht, cons.cur_x, cons.cur_y, cons.cur_x + FNT_W, cons.cur_y + FNT_H);
109     }
110 }
111 }
112 }

```

Listing 60: 向终端中写出字符

```

114 void cons_putchar(console* cons, int chr, char move)
115 {
116     char s[2];
117     s[0] = chr;
118     s[1] = 0;
119     if (s[0] == 0x09) { /* Tab */
120         for (;;) {
121             if (cons->sht != 0) {
122                 putfonts8_asc_sht(cons->sht, cons->cur_x, cons->cur_y, base3, base03, " ", 1);
123             }
124             cons->cur_x += FNT_W;
125             if (cons->cur_x == CONS_LEFT + CONS_COLW) {
126                 cons_newline(cons);
127             }
128             if ((cons->cur_x - CONS_LEFT) % (4 * FNT_W) == 0) {
129                 break;
130             }
131         }
132     } else if (s[0] == 0x0a) { /* 换行 */
133         cons_newline(cons);
134     } else if (s[0] == 0x0d) { /* 回车 */
135         cons->cur_x = CONS_LEFT;
136     } else { /* 一般字符 */
137         if (cons->sht != 0) {
138             putfonts8_asc_sht(cons->sht, cons->cur_x, cons->cur_y, base3, base03, s, 1);
139         }
140         if (move != 0) {
141             /* move为0の時不后移光标 */
142             cons->cur_x += FNT_W;
143             if (cons->cur_x == CONS_LEFT + CONS_COLW) {
144                 cons_newline(cons);
145             }
146         }
147     }
148     return;
149 }

```

Listing 61: 终端中换行

```

151 void cons_newline(console* cons)
152 {
153     int x, y;
154     sheet_t* sheet = cons->sht;
155     if (cons->cur_y < CONS_TOP + CONS_LINH - FNT_H) {
156         cons->cur_y += FNT_H; /* 换行 */
157     } else {
158         /* 滚动 */
159         if (sheet != 0) {
160             for (y = CONS_TOP; y < CONS_TOP + CONS_LINH - FNT_H; y++) {
161                 for (x = CONS_LEFT; x < CONS_LEFT + CONS_COLW; x++) {
162                     sheet->buf[x + y * sheet->bysize] = sheet->buf[x + (y + FNT_H) * sheet->bysize];
163                 }
164             }
165             for (y = CONS_TOP + CONS_LINH - FNT_H; y < CONS_TOP + CONS_LINH; y++) {
166                 for (x = CONS_LEFT; x < CONS_LEFT + CONS_COLW; x++) {
167                     sheet->buf[x + y * sheet->bysize] = base03;
168                 }
169             }
170             sheet_refresh(sheet, CONS_LEFT, CONS_TOP, CONS_LEFT + CONS_COLW, CONS_TOP + CONS_LINH);
171         }
172     }
173     cons->cur_x = CONS_LEFT;
174     return;
175 }

```

Listing 62: 向终端中写出字符串

```

177 void cons_putstr0(console* cons, char* s)

```

```

178 {
179     for (; *s != 0; s++) {
180         cons_putchar(cons, *s, 1);
181     }
182     return;
183 }
184
185 void cons_putstr1(console* cons, char* s, int l)
186 {
187     int i;
188     for (i = 0; i < l; i++) {
189         cons_putchar(cons, s[i], 1);
190     }
191     return;
192 }

```

Listing 63: 运行命令

```

194 void cons_runcmd(char* cmdline, console* cons, int* fat, unsigned int memtotal)
195 {
196     char s[CONS_COLN];
197     if (strcmp(cmdline, "mem") == 0) {
198         cmd_mem(cons, memtotal);
199     } else if (strcmp(cmdline, "clear") == 0 || strcmp(cmdline, "cls") == 0) {
200         cmd_cls(cons);
201     } else if (strcmp(cmdline, "ls -l") == 0 || strcmp(cmdline, "dir") == 0) {
202         cmd_dir(cons);
203     } else if (strcmp(cmdline, "exit") == 0) {
204         cmd_exit(cons, fat);
205     } else if (strncmp(cmdline, "start ", 6) == 0) {
206         cmd_start(cons, cmdline, memtotal);
207     } else if (strncmp(cmdline, "open ", 5) == 0) {
208         cmd_open(cons, cmdline, memtotal);
209     } else if (cmdline[0] != 0) {
210         if (cmd_app(cons, fat, cmdline) == 0) {
211             /* 不是有效命令，也不是空行 */
212             cmdline[8] = 0;
213             sprintf(s, "Command '%s' not found.\n", cmdline);
214             cons_putstr0(cons, s);
215         }
216     }
217     return;
218 }

```

Listing 64: 清屏

```

229 void cmd_cls(console* cons)
230 {
231     int x, y;
232     sheet_t* sheet = cons->sht;
233     for (y = CONS_TOP; y < CONS_TOP + CONS_LINH; y++) {
234         for (x = CONS_LEFT; x < CONS_LEFT + CONS_COLW; x++) {
235             sheet->buf[x + y * sheet->bysize] = base03;
236         }
237     }
238     sheet_refresh(sheet, CONS_LEFT, CONS_TOP, CONS_LEFT + CONS_COLW, CONS_TOP + CONS_LINH);
239     cons->cur_y = CONS_TOP;
240     return;
241 }

```

Listing 65: 列举目录

```

243 void cmd_dir(console* cons)
244 {
245     fileinfo* finfo = (fileinfo*)(ADR_DISKIMG + 0x002600);
246     int i, j;
247     char s[CONS_COLN];
248     for (i = 0; i < 224; i++) {
249         if (finfo[i].name[0] == 0x00) {
250             break;
251         }
252         if (finfo[i].name[0] != 0xe5) {
253             if ((finfo[i].type & 0x18) == 0) {
254                 sprintf(s, "filename.ext  %7d\n", finfo[i].size);
255                 for (j = 0; j < 8; j++) {
256                     s[j] = finfo[i].name[j];

```

```

257         }
258         s[9] = finfo[i].ext[0];
259         s[10] = finfo[i].ext[1];
260         s[11] = finfo[i].ext[2];
261         cons_putstr0(cons, s);
262     }
263 }
264 }
265 return;
266 }

```

Listing 66: 退出

```

268 void cmd_exit(console* cons, int* fat)
269 {
270     memman_t* memman = (memman_t*)MEMMAN_ADDR;
271     task_t* task = task_now();
272     shtctl_t* shtctl = (shtctl_t*)((int*)0x0fe4);
273     fifo32* fifo = (fifo32*)((int*)0x0fec);
274     timer_cancel(cons->timer);
275     memman_free_4k(memman, (int)fat, 4 * 2880);
276     io_cli();
277     if (cons->sht != 0) {
278         fifo32_put(fifo, cons->sht - shtctl->sheets0 + 768); /* 768 ■ 1023 */
279     } else {
280         fifo32_put(fifo, task - taskctl->tasks0 + 1024); /* 1024 ■ 2023 */
281     }
282     io_sti();
283     for (;;) {
284         task_sleep(task);
285     }
286 }

```

Listing 67: 执行程序

```

288 void cmd_start(console* cons, char* cmdline, int memtotal)
289 {
290     shtctl_t* shtctl = (shtctl_t*)((int*)0x0fe4);
291     sheet_t* sht = open_console(shtctl, memtotal);
292     fifo32* fifo = &sht->task->fifo;
293     int i;
294     sheet_slide(sht, 32, 4);
295     sheet_updown(sht, shtctl->top);
296     /* 将键入的命令复制到新命令行窗口 */
297     for (i = 6; cmdline[i] != 0; i++) {
298         fifo32_put(fifo, cmdline[i] + 256);
299     }
300     fifo32_put(fifo, 10 + 256); /* Enter */
301     cons_newline(cons);
302     return;
303 }
304
305 void cmd_open(console* cons, char* cmdline, int memtotal)
306 {
307     task_t* task = open_constask(0, memtotal);
308     fifo32* fifo = &task->fifo;
309     int i;
310     /* 将键入的命令复制到新命令行窗口 */
311     for (i = 5; cmdline[i] != 0; i++) {
312         fifo32_put(fifo, cmdline[i] + 256);
313     }
314     fifo32_put(fifo, 10 + 256); /* Enter */
315     cons_newline(cons);
316     return;
317 }
318
319 int cmd_app(console* cons, int* fat, char* cmdline)
320 {
321     memman_t* memman = (memman_t*)MEMMAN_ADDR;
322     fileinfo* finfo;
323     char name[18], *p, *q;
324     task_t* task = task_now();
325     int i, segsiz, datsiz, esp, dathrb;
326     shtctl_t* shtctl;
327     sheet_t* sht;
328
329     /* 根据命令行生成文件名 */

```

```

330     for (i = 0; i < 13; i++) {
331         if (cmdline[i] <= ' ') {
332             break;
333         }
334         name[i] = cmdline[i];
335     }
336     name[i] = 0; /* 先截断字符串 */
337
338     /* 找文件 */
339     finfo = file_search(name, (fileinfo*)(ADR_DISKIMG + 0x002600), 224);
340     if (finfo == 0 && name[i - 1] != '.') {
341         /* 找不到就加上后缀名再找一遍 */
342         name[i] = '.';
343         name[i + 1] = 'H';
344         name[i + 2] = 'R';
345         name[i + 3] = 'B';
346         name[i + 4] = 0;
347         finfo = file_search(name, (fileinfo*)(ADR_DISKIMG + 0x002600), 224);
348     }
349
350     if (finfo != 0) {
351         /* 找到文件 */
352         p = (char*)memman_alloc_4k(memman, finfo->size);
353         file_loadfile(finfo->clustno, finfo->size, p, fat, (char*)(ADR_DISKIMG + 0x003e00));
354         if (finfo->size >= 36 && strcmp(p + 4, "Hari", 4) == 0 && *p == 0x00) {
355             segsiz = *((int*)(p + 0x0000));
356             esp = *((int*)(p + 0x000c));
357             datsiz = *((int*)(p + 0x0010));
358             dathrb = *((int*)(p + 0x0014));
359             q = (char*)memman_alloc_4k(memman, segsiz);
360             task->ds_base = (int)q;
361             set_segmdesc(task->ldt + 0, finfo->size - 1, (int)p, AR_CODE32_ER + 0x60);
362             set_segmdesc(task->ldt + 1, segsiz - 1, (int)q, AR_DATA32_RW + 0x60);
363             for (i = 0; i < datsiz; i++) {
364                 q[esp + i] = p[dathrb + i];
365             }
366             start_app(0x1b, 0 * 8 + 4, esp, 1 * 8 + 4, &(task->tss.esp0));
367             shtctl = (shtctl_t)*((int*)0x0fe4);
368             for (i = 0; i < MAX_SHEETS; i++) {
369                 sht = &(shtctl->sheets0[i]);
370                 if ((sht->flags & 0x11) == 0x11 && sht->task == task) {
371                     /* 找到应用程序遗留的窗口 */
372                     sheet_free(sht); /* 关闭之 */
373                 }
374             }
375             for (i = 0; i < 8; i++) { /* 关闭所有打开的文件 */
376                 if (task->fhandle[i].buf != 0) {
377                     memman_free_4k(memman, (int)task->fhandle[i].buf, task->fhandle[i].size);
378                     task->fhandle[i].buf = 0;
379                 }
380             }
381             timer_cancelall(&task->fifo);
382             memman_free_4k(memman, (int)q, segsiz);
383         } else {
384             cons_putstr0(cons, ".hrb file format error.\n");
385         }
386         memman_free_4k(memman, (int)p, finfo->size);
387         cons_newline(cons);
388         return 1;
389     }
390     /* 未找到文件 */
391     return 0;
392 }

```



### 3 总结

这个项目我是从2015年4月20日开始，基本代码完成于5月19日。最后到小学期，完成的这个文档。整个过程是很有意思的，也成为了我在GitHub上提交最频繁的一段记录。当然，还是一直在参考书[2]中的内容再做。书的内容是中文的，但是给出的随书源代码是日文注释，不过正好会日语，所以对源代码的理解和重写也就轻松了许多。许多函数名和变量名，实际上也都是日语的一些东西，看懂了是挺有意思的，看得出作者的用心。比如Haribote这个词就是「張りぼて」，意思是“纸糊的戏剧用小道具”。也就是说，这个操作系统，就像纸糊的一样，只是个玩具，看起来还不错，其实是空的。所以，我们也可以看到，这个操作系统，真的就像玩具一样，可以玩玩，但并不能实用。比如它不能真正地读写文件，没有虚拟内存，不能连接网络等。

在现代操作系统中，进程、存储、I/O、文件、网络，显然是必要的五个元素。这里呢，实际上只是对进程的实现，是相对比较丰富的。从代码中我们可以看到，这个操作系统，是分时多任务的。也就是说，我们为每个进程分配一个时间片，然后切换上下文来运行在队列中的进程。这里我们也加了一个优先级，来调度不同等级的任务。查找算法，也都是非常简单的线性算法。因为任务很少，也没有明显的性能问题。

对于存储器的管理，实际上更简单了。我们维护一个空闲内存块的链表，然后分配和释放都在这个链表上进行。算法也是线性的。

I/O存取，其实这里就是对系统中断的处理了。主要是鼠标和键盘中断，以及屏幕的绘制。

屏幕绘制，专门写了一个窗口和图层管理器。这个算是代码量最大的一部分了，所以看起来这个操作系统代码很多，大部分都不是核心。写屏幕很有意思，因为能够看到花花绿绿的结果，很开心，有成就感。但实际上这些东西并不是操作系统的核心。

当然完成了这些东西，提供了一些API供应用程序使用，精力也就比较有限了。所以文件系统没有实现，只是简单地在汇编阶段把软盘（而且只支持FAT12格式的软盘）内容全部读到内存中，再去操作内存。同样，网络也没有实现，下学期学《计算机网络》的时候，倒不妨写一个TCP/IP玩玩。

在此期间我又读了读MINIX[6]的源代码，这是一个分模块的操作系统实现，也从中学到了很多东西。不过还没有去读Linux的内核源代码，这是计划了。

其实这个学期对我影响最深的是CS:APP——《深入理解计算机系统》[4]。花了两个月的时间读完了。这本书从头到尾把计算机硬件到软件讲了一遍，是很好的计算机入门书。而且也正式从这本书中学到了汇编，对C的理解更深了一些。不过学习的教学环境还没有这种课程，我对于这本书的理解也只是浅尝辄止，后面的作业其实没有做多少。参考文献列在最后，L<sup>A</sup>T<sub>E</sub>X用来排版以及BibT<sub>E</sub>X用来列参考文献，倒是挺方便的。

操作系统原理性的东西，其实在上个世纪80年代已经全部搞明白了。作为一个科班出身的程序员，能写OS也不是什么稀奇事了。但是，玩具好做，真正能够把操作系统上的API设计得好，能够构建出一系列应用程序，是一项非常浩大的工程。从内核构建到虚拟机，再到用

户界面和应用程序，只能说只有微软完成了这个完整地流程。就算是Mac OS X和iOS，也是基于UNIX实现改过来的，说白了不过加个壳。而Linux至今在桌面领域一团糟，衍生而来的Android虽然在移动领域二分天下，不过也不是Google独立开发出来的，还面临着Java的版权问题。所以这么一个庞大的生态系统，可以说难于上青天。当然现在上天倒是很随意的事情了。

话说回来，这个操作系统，还只能是玩具。作为课程设计，这个可能有点太大了，但我觉得这一轮下来也是值得的。这回课程设计，能够借机完成一个操作系统，也是我一直以来的愿望。不过其实对这个玩具并不满意，除了可能GUI设计的扁平化了一些，看起来更像现代操作系统以外，真是没有一点可爱的地方。本来还想实现汉字显示，结果折腾了半个月没弄好，还好一直用git做版本控制，就回滚到能用的版本提交了这个版本。如果有时间，我倒是想写一个UNIX兼容（POSIX）的内核，而不是现在这个自有API的东西。当然这是后话了。

更多的探索还在后面，这个课程设计的完成也只是更大目标的开始。要学的东西还有很多，所以就去更多地学习和实践了！





## 参考文献

- [1] D. P., B. J., D. J., 等. Operating Systems. What Can Be Automated? 1980
- [2] 川合秀实. 30天自制操作系统. 人民邮电出版社, 2012
- [3] W. Stallings. 操作系统: 精髓与设计原理. 6 版. 机械工业出版社, 2010
- [4] R. E. Bryant, D. R. O'Hallaron. 深入理解计算机系统. 2 版. 机械工业出版社, 2010
- [5] W. R. Stevens, S. A. Rago. UNIX环境高级编程. 3 版. 人民邮电出版社, 2014
- [6] A. S. Tanenbaum, A. S. Woodhull. 操作系统设计与实现. 电子工业出版社, 2007
- [7] 邓建松, 彭冉冉, 陈长松. L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>科技排版指南. 北京: 科学出版社, 2001
- [8] B. W. Kernighan, D. M. Ritchie, (编辑) C 程序设计语言. 2 版. 北京: 机械工业出版社, 2004
- [9] D. E. Knuth. The Art Of Computer Programming. Pearson Education, 1968–2011
- [10] 高德纳. 计算机程序设计艺术. 北京: 国防工业出版社, 1992–2010