

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 2 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"> 1. 处理器管理 2. 存储器管理 3. 设备管理 4. 文件管理 <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"> 1. 进程：中断处理、多任务调度、系统保护 2. 存储管理：内存分配、进程空间管理 3. I/O 系统：鼠标、键盘和屏幕的控制 4. 文件系统：文件与可执行程序的读取和加载 <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

1 概述	3
1.1 进程	3
1.2 存储器管理	4
2 设计	5
2.1 引导程序	5
2.1.1 中断处理	9
2.2 设备管理	10
2.2.1 键盘	10
2.2.2 鼠标	10
2.2.3 屏幕	11
2.2.4 窗口管理器	13
2.3 进程管理	16
2.4 内存管理	16
2.5 文件管理	16
2.6 系统接口	16
2.7 应用程序	16
3 总结	17
参考文献	19
A 程序清单	21

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

1.1 进程

进程（process），是计算机中已运行程序的实体。进程为曾经是分时系统的基本运作单位。

一个计算机系统进程包括下列数据：

- 那个程序的可运行机器码的一个在存储器的映像。
- 分配到的存储器。存储器的内容包括可运行代码、特定于进程的数据、调用堆栈、堆栈。
- 分配给该进程的资源的操作系统描述符，诸如文件描述符、数据源和数据终端。
- 安全特性，诸如进程拥有者和进程的权限集。
- 处理器状态，诸如寄存器内容、物理存储器定址等。当进程正在运行时，状态通常存储在寄存器，其他情况在存储器。

进程在运行时，状态（state）会改变。所谓状态，就是指进程目前的动作：

- 创建（new）：进程新产生中。
- 运行（running）：正在运行。
- 等待（waiting）：等待某事发生，例如等待用户输入完成。亦称“阻塞”（blocked）。
- 就绪（ready）：排班中，等待CPU。
- 完成（finish）：完成运行。

1.2 存储器管理

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
200     ALIGNB 16
201 bootpack:

```

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 10: 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 11: 初始化内存管理器

```

65     unsigned int memtotal = memtest(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 12: 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 13: 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 14: 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  }

```

2.1.1 中断处理

首先是初始化GDT和IDT:

Listing 15: 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);
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);
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 16: 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 设备管理

2.2.1 键盘

Listing 17: 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 18: 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 }

```

2.2.2 鼠标

Listing 19: 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 20: 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 |= 0xfffff00;
66         }
67         if ((mdec->buf[0] & 0x20) != 0) {
68             mdec->y |= 0xfffff00;
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 }

```

2.2.3 屏幕

初始化一个 $6 \times 6 \times 6$ 的调色板：

Listing 21: 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);
41 return;
42 }
43
44 void set_palette(int start, int end, unsigned char* rgb)
45 {
46     int i, eflags;
47     eflags = io_load_eflags(); /* 备份中断许可标志 */
48     io_cli(); /* 标志置0, 禁止中断 */
49     io_out8(0x03c8, start);
50     for (i = start; i <= end; i++) {
51         io_out8(0x03c9, rgb[0] / 4);
52         io_out8(0x03c9, rgb[1] / 4);
53         io_out8(0x03c9, rgb[2] / 4);
54         rgb += 3;
55     }
56     io_store_eflags(eflags); /* 复原中断许可标志 */
57     return;
58 }

```

初始化鼠标光标:

Listing 22: 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         " *****00000000*",
154         " **0000000000*",
155         " ***000000****",
156         " ****000*00*   ",
157         " *****00*00*",
158         " **0*0*00*     ",
159         " ***000*00*    ",
160         " ****000*00*   ",
161         " *****00*    ",
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] == '0') {

```



```

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 23: 初始化鼠标光标

```

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

```

显示字符串：

Listing 24: 初始化鼠标光标

```

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 }

```

2.2.4 窗口管理器

Listing 25: 初始化图层管理器

```

7 shtctl_t* shtctl_init(memman_t* memman, unsigned char* vram, int xsize, int ysize)
8 {
9     shtctl_t* ctl;
10     int i;
11     ctl = (shtctl_t*)memman_alloc_4k(memman, sizeof(shtctl_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(shtctl_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 26: 为新图层分配内存

```

32 sheet_t* sheet_alloc(shtctl_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 27: 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 }

```

改变图层层次:

Listing 28: 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 }

```

2.3 进程管理

2.4 内存管理

2.5 文件管理

2.6 系统接口

2.7 应用程序

3 总结

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^AT_EX 2_ε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

A 程序清单

```
.
├── Makefile
├── Makefile.rule
├── apilib.h
├── app
│   ├── Makefile
│   ├── Makefile.rule
│   ├── a
│   │   ├── Makefile
│   │   └── a.c
│   ├── app_make.txt
│   ├── beepdown
│   │   ├── Makefile
│   │   └── beepdown.c
│   ├── cat
│   │   ├── Makefile
│   │   └── cat.c
│   ├── color
│   │   ├── Makefile
│   │   └── color.c
│   ├── color2
│   │   ├── Makefile
│   │   └── color2.c
│   ├── hello3
│   │   ├── Makefile
│   │   └── hello3.c
│   ├── hello4
│   │   ├── Makefile
│   │   └── hello4.c
│   ├── hello5
│   │   ├── Makefile
│   │   └── hello5.nas
│   ├── lines
│   │   ├── Makefile
│   │   └── lines.c
│   ├── noodle
│   │   ├── Makefile
│   │   └── noodle.c
│   ├── primes
│   │   ├── Makefile
│   │   └── primes.c
│   ├── primes2
│   │   ├── Makefile
│   │   └── primes2.c
│   ├── primes3
│   │   ├── Makefile
│   │   └── primes3.c
│   ├── star1
│   │   ├── Makefile
│   │   └── star1.c
│   ├── stars
│   │   ├── Makefile
│   │   └── stars.c
│   ├── stars2
│   │   ├── Makefile
│   │   └── stars2.c
│   ├── walk
│   │   ├── Makefile
│   │   └── walk.c
│   ├── winhelo
│   │   ├── Makefile
│   │   └── winhelo.c
```


软件包装原始码及二进制可执行形式的权利，无论此包装是否经改作皆然：

- * 对于本软件源代码的再散播，必须保留上述的版权宣告、此三条件表列，以及下述的免责声明。
- * 对于本套件二进制可执行形式的再散播，必须连带以文件以及／或者其他附于散播包装中的媒介方式，重制上述之版权宣告、此三条件表列，以及下述的免责声明。
- * 未获事前取得书面许可，不得使用伯克利加州大学或本软件贡献者之名称，来为本软件之衍生物做任何表示支持、认可或推广、促销之行为。

免责声明：本软件是由作者及本软件之贡献者以现状提供，本软件包装不负任何明示或默示之担保责任，包括但不限于就适售性以及特定目的的适用性为默示性担保。作者及本软件之贡献者，无论任何条件、无论成因或任何责任主义、无论此责任为因合约关系、无过失责任主义或因非违约之侵权（包括过失或其他原因等）而起，对于任何因使用本软件包装所产生的任何直接性、间接性、偶发性、特殊性、惩罚性或任何结果的损害（包括但不限于替代商品或劳务之购用、使用损失、资料损失、利益损失、业务中断等等），不负任何责任，即在该种使用已获事前告知可能会造成此类损害的情形下亦然。