河南工業大學

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 6 月 27 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	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： 1. 处理器管理 2. 存储器管理 3. 设备管理 4. 文件管理 现在的桌面操作系统多是多任务分时操作系统。				
任务要求	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： 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 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。				
参考文献	川合秀实. 30天自制操作系统. 人民邮电出版社, 2012 W. Stallings. 操作系统: 精髓与设计原理（第6版）. 机械工业出版社, 2010 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 A.S.Tanenbaum, 等. 操作系统设计与实现. 电子工业出版社, 2007 W. R. Stevens, 等. UNIX环境高级编程（第3版）. 人民邮电出版社, 2014				
审查意见	指导教师签字： 教研室主任签字：　　2015 年 6 月 25 日				

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

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。它有下面三个目标：

- 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

```
scrn320:
  MOV AL, 0x13 ; VGA 320x200x8位色
  MOV AH, 0x00
  INT 0x10
  MOV BYTE [VMODE], 8 ; 记录画面模式
  MOV WORD [SCRNX], 320
  MOV WORD [SCRNY], 200
  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

```
; 切换到保护模式

[INSTRSET "i486p"] ; 使用486指令集

  LGDT [GDTR0] ; 设置临时GDT
  MOV EAX, CR0
  AND EAX, 0x7fffffff ; 将bit31置0（禁止分页）
  OR EAX, 0x00000001 ; 将bit0置1（切换到保护模式）
  MOV CR0, EAX
  JMP pipelineflush ; 重置CPU流水线
pipelineflush:
  MOV AX, 1*8 ; 32bit可读写段
  MOV DS, AX
  MOV ES, AX
  MOV FS, AX
  MOV GS, AX
  MOV SS, AX

; 传送bootpack

  MOV ESI, bootpack ; 传送来源
  MOV EDI, BOTPAK ; 传送目的
  MOV ECX, 512*1024/4
  CALL memcpy

; 转存磁盘数据

; 启动扇区

  MOV ESI, 0x7c00 ; 传送来源
  MOV EDI, DSKCAC ; 传送目的
  MOV ECX, 512/4
  CALL memcpy

; 剩下的

  MOV ESI, DSKCAC0+512; 传送来源
  MOV EDI, DSKCAC+512 ; 传送目的
  MOV ECX, 0
  MOV CL, BYTE [CYLS]
  IMUL ECX, 512*18*2/4 ; 柱面数变为字节数/4
  SUB ECX, 512/4 ; 减去IPL
  CALL memcpy
```

然后调用主函数，正式启动操作系统：

Listing 9: 启动bootpack

```
; 启动bootpack

  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);
77      /* sht_back */
78      sht_back = sheet_alloc(shtctl);
79      buf_back = (unsigned char*)memman_alloc_4k(memman, binfo->scrnx * binfo->scrny);
83      /* sht_cons */
84      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);
93      int mx = (binfo->scrnx - CURSOR_X) / 2; /* 计算画面中央坐标 */
94      int my = (binfo->scrny - CURSOR_Y) / 2;
```

接下来进入一个无限循环，该循环查询CPU中断事件，并给与相应：

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 */
110         }
111         io_cli();
112         if (fifo32_status(&fifo) == 0) {
113             /* FIFO为空，当存在搁置的绘图操作时立即执行 */
126         } else {
127             i = fifo32_get(&fifo);
128             io_sti();
129             if (key_win != 0 && key_win->flags == 0) { /* 窗口关闭 */
136             }
137             if (256 <= i && i <= 511) { /* 键盘 */
239             } else if (512 <= i && i <= 767) { /* 鼠标 */
330             }
331         }
332     }
```

2.2 设备管理

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., et al. Oprerating 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 edn. 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 edn. 机械工业出版社, 2010

[5] W. R. Stevens, S. A. Rago. UNIX环境高级编程. 3 edn. 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. LaTeX 2ε科技排版指南. 北京: 科学出版社, 2001

[8] B. W. Kernighan, D. M. Ritchie, (Editors) C 程序设计语言. 2 edn. 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
```

```
    ├── winhelo2
    │   ├── Makefile
    │   └── winhelo2.c
    ├── winhelo3
    │   ├── Makefile
    │   └── winhelo3.c
├── lib
    ├── Makefile
    ├── alloca.nas
    ├── api001.nas
    ├── api002.nas
    ├── api003.nas
    ├── api004.nas
    ├── api005.nas
    ├── api006.nas
    ├── api007.nas
    ├── api008.nas
    ├── api009.nas
    ├── api010.nas
    ├── api011.nas
    ├── api012.nas
    ├── api013.nas
    ├── api014.nas
    ├── api015.nas
    ├── api016.nas
    ├── api017.nas
    ├── api018.nas
    ├── api019.nas
    ├── api020.nas
    ├── api021.nas
    ├── api022.nas
    ├── api023.nas
    ├── api024.nas
    ├── api025.nas
    └── api026.nas
└── sys
    ├── Makefile
    ├── ZpixEX2-12.fnt
    ├── asmhead.nas
    ├── bootpack.c
    ├── bootpack.h
    ├── console.c
    ├── dsctbl.c
    ├── fifo.c
    ├── file.c
    ├── graphic.c
    ├── int.c
    ├── ipl10.nas
    ├── keyboard.c
    ├── memory.c
    ├── mouse.c
    ├── mtask.c
    ├── naskfunc.nas
    ├── sheet.c
    ├── timer.c
    ├── unifont-7.0.06.hex
    └── window.c

23 directories, 95 files
```

版权所有 (c) 2015 田劲锋

保留所有权利

这份授权条款，在使用者符合以下三条件的情形下，授予使用者使用及再散播本

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。

Listing 15: sys/ipl10.nas

```
; pripara-ipl

CYLS EQU 10 ; 常量定义

  ORG 0x7c00 ; 程序装载地址

; FAT12软盘格式化

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

; 程序主体

entry:
  MOV AX, 0 ; 初始化寄存器
  MOV SS, AX
  MOV SP, 0x7c00
  MOV DS, AX

; 读取磁盘

  MOV AX, 0x0820
  MOV ES, AX
  MOV CH, 0 ; 柱面0
  MOV DH, 0 ; 磁头0
  MOV CL, 2 ; 扇区2
```

```
45
46  readloop:
47    MOV SI, 0 ; 记录失败次数
48
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
63
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
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
79
80  ; 读取完成后执行haribote.sys
81
82    MOV [0x0ff0], CH ; 告知IPL加载到了何处
83    JMP 0xc200
84
85  error:
86    MOV SI, msg
87
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
97
98  fin:
99    HLT ; 停止CPU，等待
100   JMP fin ; 无限循环
101
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 16: sys/asmhead.nas

```
; pripara-os boot asm

[INSTRSET "i486p"]

VBEMODE EQU 0x101
; （VBE画面模式列表）
; 0x100 : 640 x 400 x 8位色
; 0x101 : 640 x 480 x 8位色
; 0x103 : 800 x 600 x 8位色
; 0x105 : 1024 x 768 x 8位色
; 0x107 : 1280 x 1024 x 8位色

BOTPAK EQU 0x00280000 ; bootpack加载目的
DSKCAC EQU 0x00100000 ; 磁盘缓存
DSKCAC0 EQU 0x00008000 ; 磁盘缓存（实模式）

; BOOT_INFO有关
CYLS EQU 0x0ff0 ; 设定启动区
LEDS EQU 0x0ff1
VMODE EQU 0x0ff2 ; 颜色位数
SCRNX EQU 0x0ff4 ; 水平分辨率
SCRNY EQU 0x0ff6 ; 垂直分辨率
VRAM EQU 0x0ff8 ; 图像缓冲区地址

  ORG 0xc200 ; 程序被装载的位置

; 判断是否存在VBE

  MOV AX, 0x9000
  MOV ES, AX
  MOV DI, 0
  MOV AX, 0x4f00
  INT 0x10
  CMP AX, 0x004f
  JNE scrn320

; 检查VBE版本>2.0

  MOV AX, [ES:DI+4]
  CMP AX, 0x0200
  JB scrn320 ; if (AX < 0x0200) goto scrn320

; 获取画面模式信息

  MOV CX, VBEMODE
  MOV AX, 0x4f01
  INT 0x10
  CMP AX, 0x004f
  JNE scrn320

; 确认画面模式信息

  CMP BYTE [ES:DI+0x19], 8 ; 颜色数为8
  JNE scrn320
  CMP BYTE [ES:DI+0x1b], 4 ; 调色板模式
  JNE scrn320
  MOV AX, [ES:DI+0x00] ; 模式属性能否加上0x4000
  AND AX, 0x0080
  JZ scrn320 ; 如果不能

; 切换画面模式

  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
74
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
83
84  ; 用BIOS获取键盘上LED指示灯的状态
85
86  keystatus:
87    MOV AH, 0x02
88    INT 0x16 ; keyboard BIOS
89    MOV [LEDS], AL
90
91  ; 屏蔽PIC中断
92  ; 根据AT兼容机的规格，要初始化PIC必须在CLI前执行，
93  ; 否则有时会挂起。随后初始化PIC
94
95    MOV AL, 0xff
96    OUT 0x21, AL
97    NOP ; 有些机器不能连续执行OUT命令
98    OUT 0xa1, AL
99    CLI ; 禁止CPU级别的中断
100
101 ; 为了让CPU能访问1MB以上的内存空间，设置A20GATE
102
103   CALL waitkbdout
104   MOV AL, 0xd1
105   OUT 0x64, AL
106   CALL waitkbdout
107   MOV AL, 0xdf ; enable A20
108   OUT 0x60, AL
109   CALL waitkbdout
110
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
154
155 ; asmhead的任务完成了，剩下的交给bootpack
156
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
172 waitkbdout:
173   IN AL, 0x64
174   AND AL, 0x02
175   IN AL, 0x60 ; 空读
176   JNZ waitkbdout ; AND结果不为0时跳转到waitkbdout
177   RET
178
179 memcpy:
180   MOV EAX, [ESI]
181   ADD ESI, 4
182   MOV [EDI], EAX
183   ADD EDI, 4
184   SUB ECX, 1
185   JNZ memcpy ; 相减不为0时跳转到memcpy
186   RET
187 ; 如果你没忘记地址大小前缀，还可以使用字符串命令memcpy
188
189   ALIGNB 16
190 GDT0:
191   RESB 8 ; 空区域
```

```
  DW 0xffff,0x0000,0x9200,0x00cf ; 可读写的32位段
  DW 0xffff,0x0000,0x9a28,0x0047 ; 可执行的32位段（bootpack用）

  DW 0
GDTR0:
  DW 8*3-1
  DD GDT0

  ALIGNB 16
bootpack:
```

Listing 17: sys/naskfunc.nas

```
; naskfunc

[FORMAT "WCOFF"] ; 目标文件格式
[INSTRSET "i486p"] ; 指定486模式
[BITS 32] ; 32位模式
[FILE "naskfunc.nas"] ; 源文件名

GLOBAL _io_hlt, _io_cli, _io_sti, _io_stihlt
GLOBAL _io_in8, _io_in16, _io_in32
GLOBAL _io_out8, _io_out16, _io_out32
GLOBAL _io_load_eflags, _io_store_eflags
GLOBAL _load_gdtr, _load_idtr
GLOBAL _load_cr0, _store_cr0
GLOBAL _load_tr
GLOBAL _asm_inthandler20, _asm_inthandler21
GLOBAL _asm_inthandler27, _asm_inthandler2c
GLOBAL _asm_inthandler0c, _asm_inthandler0d
GLOBAL _asm_end_app, _memtest_sub
GLOBAL _farjmp, _farcall
GLOBAL _asm_hrb_api, _start_app
EXTERN _inthandler20, _inthandler21
EXTERN _inthandler27, _inthandler2c
EXTERN _inthandler0c, _inthandler0d
EXTERN _hrb_api

[SECTION .text]

_io_hlt: ; void io_hlt(void);
  HLT
  RET

_io_cli: ; void io_cli(void);
  CLI
  RET

_io_sti: ; void io_sti(void);
  STI
  RET

_io_stihlt: ; void io_stihlt(void);
  STI
  HLT
  RET

_io_in8: ; int io_in8(int port);
  MOV EDX, [ESP+4] ; port
  MOV EAX, 0
  IN AL, DX
  RET

_io_in16: ; int io_in16(int port);
```

```
52    MOV EDX, [ESP+4] ; port
53    MOV EAX, 0
54    IN AX, DX
55    RET
56
57  _io_in32: ; int io_in32(int port);
58    MOV EDX, [ESP+4] ; port
59    IN EAX, DX
60    RET
61
62  _io_out8: ; void io_out8(int port, int data);
63    MOV EDX, [ESP+4] ; port
64    MOV AL, [ESP+8] ; data
65    OUT DX, AL
66    RET
67
68  _io_out16: ; void io_out16(int port, int data);
69    MOV EDX, [ESP+4] ; port
70    MOV EAX, [ESP+8] ; data
71    OUT DX, AX
72    RET
73
74  _io_out32: ; void io_out32(int port, int data);
75    MOV EDX, [ESP+4] ; port
76    MOV EAX, [ESP+8] ; data
77    OUT DX, EAX
78    RET
79
80  _io_load_eflags: ; int io_load_eflags(void);
81    PUSHFD ; PUSH EFLAGS 的意思
82    POP EAX
83    RET
84
85  _io_store_eflags: ; void io_store_eflags(int eflags);
86    MOV EAX, [ESP+4]
87    PUSH EAX
88    POPFD ; POP EFLAGS 的意思
89    RET
90
91  _load_gdtr: ; void load_gdtr(int limit, int addr);
92    MOV AX, [ESP+4] ; limit
93    MOV [ESP+6], AX
94    LGDT [ESP+6]
95    RET
96
97  _load_idtr: ; void load_idtr(int limit, int addr);
98    MOV AX, [ESP+4] ; limit
99    MOV [ESP+6], AX
100   LIDT [ESP+6]
101   RET
102
103 _load_cr0: ; int load_cr0(void);
104   MOV EAX, CR0
105   RET
106
107 _store_cr0: ; void store_cr0(int cr0);
108   MOV EAX, [ESP+4]
109   MOV CR0, EAX
110   RET
111
112 _load_tr: ; void load_tr(int tr);
113   LTR [ESP+4] ; tr
114   RET
115
```

```
_asm_inthandler20:
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX, ESP
  PUSH EAX
  MOV AX, SS
  MOV DS, AX
  MOV ES, AX
  CALL _inthandler20
  POP EAX
  POPAD
  POP DS
  POP ES
  IRETD

_asm_inthandler21:
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX, ESP
  PUSH EAX
  MOV AX, SS
  MOV DS, AX
  MOV ES, AX
  CALL _inthandler21
  POP EAX
  POPAD
  POP DS
  POP ES
  IRETD

_asm_inthandler27:
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX, ESP
  PUSH EAX
  MOV AX, SS
  MOV DS, AX
  MOV ES, AX
  CALL _inthandler27
  POP EAX
  POPAD
  POP DS
  POP ES
  IRETD

_asm_inthandler2c:
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX, ESP
  PUSH EAX
  MOV AX, SS
  MOV DS, AX
  MOV ES, AX
  CALL _inthandler2c
  POP EAX
  POPAD
  POP DS
  POP ES
  IRETD

```

```
_asm_inthandler0d:
  STI
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX, ESP
  PUSH EAX
  MOV AX, SS
  MOV DS, AX
  MOV ES, AX
  CALL _inthandler0d
  CMP EAX, 0 ; 这里不同
  JNE _asm_end_app ; 这里不同
  POP EAX
  POPAD
  POP DS
  POP ES
  ADD ESP, 4 ; INT 0x0d 需要
  IRETD

_asm_inthandler0c:
  STI
  PUSH ES
  PUSH DS
  PUSHAD
  MOV EAX,ESP
  PUSH EAX
  MOV AX,SS
  MOV DS,AX
  MOV ES,AX
  CALL _inthandler0c
  CMP EAX,0
  JNE _asm_end_app
  POP EAX
  POPAD
  POP DS
  POP ES
  ADD ESP,4 ; INT 0x0c 需要
  IRETD

_memtest_sub: ; unsigned int memtest_sub(unsigned int start, unsigned int end)
  PUSH EDI ; （待用的EBX, ESI, EDI）
  PUSH ESI
  PUSH EBX
  MOV ESI, 0xaa55aa55 ; pat0 = 0xaa55aa55;
  MOV EDI, 0x55aa55aa ; pat1 = 0x55aa55aa;
  MOV EAX, [ESP+12+4] ; i = start;
mts_loop:
  MOV EBX, EAX
  ADD EBX, 0xffc ; p = i + 0xffc;
  MOV EDX, [EBX] ; old = *p;
  MOV [EBX], ESI ; *p = pat0;
  XOR DWORD [EBX], 0xffffffff ; *p ^= 0xffffffff;
  CMP EDI, [EBX] ; if (*p != pat1) goto fin;
  JNE mts_fin
  XOR DWORD [EBX], 0xffffffff ; *p ^= 0xffffffff;
  CMP ESI, [EBX] ; if (*p != pat0) goto fin;
  JNE mts_fin
  MOV [EBX], EDX ; *p = old;
  ADD EAX, 0x1000 ; i += 0x1000;
  CMP EAX, [ESP+12+8] ; if (i <= end) goto mts_loop;
  JBE mts_loop
  POP EBX
  POP ESI
```

```
244   POP EDI
245   RET
246 mts_fin:
247   MOV [EBX], EDX ; *p = old;
248   POP EBX
249   POP ESI
250   POP EDI
251   RET
252
253 _farjmp: ; void farjmp(int eip, int cs);
254   JMP FAR [ESP+4] ; eip, cs
255   RET
256
257 _farcall: ; void farcall(int eip, int cs);
258   CALL FAR [ESP+4] ; eip, cs
259   RET
260
261 _asm_hrb_api:
262   ; 方便起见一开始就禁止中断
263   PUSH DS
264   PUSH ES
265   PUSHAD ; 保存用PUSH
266   PUSHAD ; hrb_api用PUSH
267   MOV AX, SS
268   MOV DS, AX ; 系统用段地址存入DS和ES
269   MOV ES, AX
270   CALL _hrb_api
271   CMP EAX, 0 ; EAX不为0时程序结束
272   JNE _asm_end_app
273   ADD ESP, 32
274   POPAD
275   POP ES
276   POP DS
277   IRETD
278 _asm_end_app:
279   ; EAX为tss.esp0的地址
280   MOV ESP, [EAX]
281   MOV DWORD [EAX+4], 0
282   POPAD
283   RET ; 返回到cmd_app
284
285 _start_app: ; void start_app(int eip, int cs, int esp, int ds, int *tss_esp0);
286   PUSHAD ; 备份32位寄存器
287   MOV EAX, [ESP+36] ; 程序用EIP
288   MOV ECX, [ESP+40] ; 程序用CS
289   MOV EDX, [ESP+44] ; 程序用ESP
290   MOV EBX, [ESP+48] ; 程序用DS/SS
291   MOV EBP, [ESP+52] ; tss.esp0的地址
292   MOV [EBP ], ESP ; 系统用ESP
293   MOV [EBP+4], SS ; OS用のSSを保存
294   MOV ES, BX
295   MOV DS, BX
296   MOV FS, BX
297   MOV GS, BX
298   ; 调整栈以免跳转到程序
299   OR ECX, 3 ; 段号|3
300   OR EBX, 3 ; 段号|3
301   PUSH EBX ; 程序的SS
302   PUSH EDX ; 程序的ESP
303   PUSH ECX ; 程序的CS
304   PUSH EAX ; 程序的EIP
305   RETF
306   ; 程序终止后不会回到这里
```

Listing 18: sys/bootpack.h

```
#ifndef BOOTPACK_H
#define BOOTPACK_H

#define SYSNAME "PriPara OS"
#define SYSVERS "28"
#define SYSNAMEVER SYSNAME " " SYSVERS

/* asmhead.nas */
typedef struct BOOTINFO { /* 0x0ff0-0x0fff */
    char cyls; /* 启动区读盘终止处 */
    char leds; /* 键盘灯状态 */
    char vmode; /* 显卡模式 */
    char reserve;
    short scrnx, scrny; /* 分辨率 */
    char* vram;
} bootinfo_t;

#define ADR_BOOTINFO 0x00000ff0
#define ADR_DISKIMG 0x00100000

/* naskfunc.nas */
void io_hlt(void);
void io_cli(void);
void io_sti(void);
void io_stihlt(void);
int io_in8(int port);
void io_out8(int port, int data);
int io_load_eflags(void);
void io_store_eflags(int eflags);
void load_gdtr(int limit, int addr);
void load_idtr(int limit, int addr);
int load_cr0(void);
void store_cr0(int cr0);
void load_tr(int tr);
void asm_inthandler0c(void);
void asm_inthandler0d(void);
void asm_inthandler20(void);
void asm_inthandler21(void);
void asm_inthandler27(void);
void asm_inthandler2c(void);
unsigned int memtest_sub(unsigned int start, unsigned int end);
void farjmp(int eip, int cs);
void farcall(int eip, int cs);
void asm_hrb_api(void);
void start_app(int eip, int cs, int esp, int ds, int* tss_esp0);

/* fifo.c */
typedef struct FIFO32 {
    int* buf;
    int p, q, size, free, flags;
    struct TASK* task;
} fifo32;

void fifo32_init(fifo32* q, int size, int* buf, struct TASK*);
int fifo32_put(fifo32* fifo, int data);
int fifo32_get(fifo32* fifo);
int fifo32_status(fifo32* fifo);

/* graphic.c */
void init_palette(void);
void set_palette(int start, int end, unsigned char* rgb);
void boxfill8(unsigned char* vram, int X, unsigned char c,
    int x0, int y0, int x1, int y1);
```

```
void boxsize8(unsigned char* vram, int X, unsigned char c,
    int x0, int y0, int width, int height);
void init_screen8(char* vram, int x, int y);
void putfont8(char* vram, int xsize, int x, int y, char c, char* font);
void putfonts8_asc(char* vram, int xsize, int x, int y, char c, unsigned char* s);
void init_mouse_cursor8(char* mouse, char bc);
void putblock8_8(char* vram, int vxsize, int pxsize,
    int pysize, int px0, int py0, char* buf, int bxsize);

/* 16位色 */
#define base03 0
#define base02 1
#define base01 2
#define base00 3
#define base0 4
#define base1 5
#define base2 6
#define base3 7
#define yellow 8
#define orange 9
#define red 10
#define magenta 11
#define violet 12
#define blue 13
#define cyan 14
#define green 15

#define BGM cyan

/* 字体 */
#define FNT_H 12
#define FNT_W 6 // FNT_H / 2
#define FNT_OFFSET 726 // 60 * FNT_H

/* 鼠标指针 */
#define CURSOR_X 12
#define CURSOR_Y 19

/* dsctbl.c */
typedef struct SEGMENT_DESCRIPTOR {
    short limit_low, base_low;
    char base_mid, access_right;
    char limit_high, base_high;
} segment_descriptor;

typedef struct GATE_DESCRIPTOR {
    short offset_low, selector;
    char dw_count, access_right;
    short offset_high;
} gate_descriptor;

void init_gdtidt(void);
void set_segmdesc(segment_descriptor* sd, unsigned int limit, int base, int ar);
void set_gatedesc(gate_descriptor* gd, int offset, int selector, int ar);

#define ADR_IDT 0x0026f800
#define LIMIT_IDT 0x000007ff
#define ADR_GDT 0x00270000
#define LIMIT_GDT 0x0000ffff
#define ADR_BOTPAK 0x00280000
#define LIMIT_BOTPAK 0x0007ffff
#define AR_DATA32_RW 0x4092
#define AR_CODE32_ER 0x409a
#define AR_LDT 0x0082
```

```
#define AR_TSS32 0x0089
#define AR_INTGATE32 0x008e

/* int.c */
struct KEYBUF {
    unsigned char data[32];
    int next_r, next_w, len;
};

void init_pic(void);
void inthandler21(int* esp);
void inthandler27(int* esp);
void inthandler2c(int* esp);

#define PIC0_ICW1 0x0020
#define PIC0_OCW2 0x0020
#define PIC0_IMR 0x0021
#define PIC0_ICW2 0x0021
#define PIC0_ICW3 0x0021
#define PIC0_ICW4 0x0021
#define PIC1_ICW1 0x00a0
#define PIC1_OCW2 0x00a0
#define PIC1_IMR 0x00a1
#define PIC1_ICW2 0x00a1
#define PIC1_ICW3 0x00a1
#define PIC1_ICW4 0x00a1

/* keyboard.c */
void inthandler21(int* esp);
void wait_KBC_sendready(void);
void init_keyboard(fifo32* fifo, int data0);

#define PORT_KEYDAT 0x0060
#define PORT_KEYCMD 0x0064

/* mouse.c */
typedef struct MOUSE_DEC {
    unsigned char buf[3], phase;
    int x, y, btn;
} mouse_dec;

void inthandler2c(int* esp);
void enable_mouse(fifo32* fifo, int data0, mouse_dec* mdec);
int mouse_decode(mouse_dec* mdec, unsigned char dat);

/* memory.c */
#define MEMMAN_FREES 4090 // 大约是32KB
#define MEMMAN_ADDR 0x003c0000

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

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

unsigned int memtest(unsigned int start, unsigned int end);
void memman_init(memman_t* man);
unsigned int memman_total(memman_t* man);
unsigned int memman_alloc(memman_t* man, unsigned int size);
int memman_free(memman_t* man, unsigned int addr, unsigned int size);
unsigned int memman_alloc_4k(memman_t* man, unsigned int size);
```

```
192 int memman_free_4k(memman_t* man, unsigned int addr, unsigned int size);
193
194 /* sheet.c */
195 #define MAX_SHEETS 256
196
197 typedef struct SHEET {
198     unsigned char* buf;
199     int bxsize, bysize, vx0, vy0, alpha, height, flags;
200     struct SHTCTL* ctl;
201     struct TASK* task;
202 } sheet_t;
203
204 typedef struct SHTCTL {
205     unsigned char *vram, *map;
206     int xsize, ysize, top;
207     sheet_t* sheets[MAX_SHEETS];
208     sheet_t sheets0[MAX_SHEETS];
209 } shtctl_t;
210
211 shtctl_t* shtctl_init(memman_t* memman, unsigned char* vram, int xsize, int ysize);
212 struct SHEET* sheet_alloc(shtctl_t* ctl);
213 void sheet_setbuf(sheet_t* sht, unsigned char* buf, int xsize, int ysize, int col_inv);
214 void sheet_updown(sheet_t* sht, int height);
215 void sheet_refresh(sheet_t* sht, int bx0, int by0, int bx1, int by1);
216 void sheet_slide(sheet_t* sht, int vx0, int vy0);
217 void sheet_free(sheet_t* sht);
218
219 /* timer.c */
220 #define MAX_TIMER 512
221
222 typedef struct TIMER {
223     struct TIMER* next;
224     unsigned int timeout;
225     char flags, flags2;
226     fifo32* fifo;
227     int data;
228 } timer_t;
229
230 typedef struct TIMERCTL {
231     unsigned int count, next, using;
232     timer_t *t0, timers0[MAX_TIMER];
233 } timerctl_t;
234
235 extern timerctl_t timerctl;
236
237 void init_pit(void);
238 timer_t* timer_alloc(void);
239 void timer_free(timer_t* timer);
240 void timer_init(timer_t* timer, fifo32* fifo, int data);
241 void timer_settime(timer_t* timer, unsigned int timeout);
242 void inthandler20(int* esp);
243 int timer_cancel(timer_t* timer);
244 void timer_cancelall(fifo32* fifo);
245
246 /* mtask.c */
247 #define MAX_TASKS 1024 /* 最大任务数 */
248 #define TASK_GDT0 3 /* 从GDT的哪里开始分配TSS */
249 #define MAX_TASKS_LV 100
250 #define MAX_TASKLEVELS 10
251
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 ldtr, 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;
285 
286 extern timer_t* task_timer;
287 extern taskctl_t* taskctl;
288 
289 task_t* task_now(void);
290 task_t* task_init(memman_t* memman);
291 task_t* task_alloc(void);
292 void task_run(task_t* task, int level, int priority);
293 void task_switch(void);
294 void task_sleep(task_t* task);
295 
296 /* window.c */
297 #define WIN_TOP 28
298 #define WIN_LEFT 8
299 
300 void make_window8(unsigned char* buf, int xsize, int ysize, char* title, char act);
301 void make_wtitle8(unsigned char* buf, int xsize, char* title, char act);
302 void make_textbox8(sheet_t* sht, int x0, int y0, int sx, int sy, int c);
303 void putfonts8_asc_sht(sheet_t* sht, int x, int y, int c, int b, char* s, int l);
304 void change_wtitle8(sheet_t* sht, char act);
305 
306 /* console.c */
307 #define CONS_COLN 80 /* 列数（自定义） */
308 #define CONS_LINE 30 /* 行数（自定义） */
309 #define CONS_COLW (FNT_W * CONS_COLN) /* 列宽 */
310 #define CONS_LINH (FNT_H * CONS_LINE) /* 行高 */
311 #define CONS_LEFT 3 /* 左边宽度 */
312 #define CONS_TOP 23 /* 顶部高度 */
313 #define CONS_WINW (CONS_COLW + CONS_LEFT * 2) /* 窗口宽度 */
314 #define CONS_WINH (CONS_LINH + CONS_TOP + CONS_LEFT) /* 窗口高度 */
315 
316 typedef struct CONSOLE {
317     sheet_t* sht;
318     int cur_x, cur_y, cur_c;
319     timer_t* timer;
```

```
} console;
typedef struct FILEHANDLE {
    char *buf;
    int size;
    int pos;
} filehandle;
void console_task(sheet_t* sheet, unsigned int memtotal);
void cons_putchar(console* cons, int chr, char move);
void cons_newline(console* cons);
void cons_putstr0(console* cons, char* s);
void cons_putstr1(console* cons, char* s, int l);
void cons_runcmd(char* cmdline, console* cons, int* fat, unsigned int memtotal);
void cmd_mem(console* cons, unsigned int memtotal);
void cmd_cls(console* cons);
void cmd_dir(console* cons);
void cmd_exit(console* cons, int* fat);
void cmd_start(console* cons, char* cmdline, int memtotal);
void cmd_open(console* cons, char* cmdline, int memtotal);
int cmd_app(console* cons, int* fat, char* cmdline);
int* hrb_api(int edi, int esi, int ebp, int esp, int ebx, int edx, int ecx, int eax);
int* inthandler0c(int* esp);
int* inthandler0d(int* esp);
void asm_end_app(void);
void hrb_api_linewin(sheet_t* sht, int x0, int y0, int x1, int y1, int col);

/* file.c */
typedef struct FILEINFO {
    unsigned char name[8], ext[3], type;
    char reserve[10];
    unsigned short time, date, clustno;
    unsigned int size;
} fileinfo;

void file_readfat(int* fat, unsigned char* img);
void file_loadfile(int clustno, int size, char* buf, int* fat, char* img);
fileinfo* file_search(char* name, fileinfo* finfo, int max);

/* bootpack.c */
task_t* open_constask(sheet_t* sht, unsigned int memtotal);
sheet_t* open_console(shtctl_t* shtctl, unsigned int memtotal);

#endif
```