**《30天自制操作系统》读书笔记(4) 绘图**

暑假果然是滋生懒散的温床. (╯‵□′)╯︵┻━┻

好久不动都忘记之前做到哪里了, 上次好像做到了C语言的引入, 这一节所做的东西都相当轻松, 将会绘制出操作系统的基本界面.

* **绘图的原理**

按照书中所说, 将值写入到显存中就能在屏幕上显示相应的像素, 在asmhead.nas 中有这一段:

CYLS EQU 0x0ff0 ; 设定启动区

LEDS EQU 0x0ff1

VMODE EQU 0x0ff2 ; 关于颜色数目的信息，颜色的位数

SCRNX EQU 0x0ff4 ; 分辨率 X(Screen X)

SCRNY EQU 0x0ff6 ; 分辨率 Y(Screen Y)

VRAM EQU 0x0ff8 ; 图像缓冲区的起始地址

ORG 0xc200

MOV AL,0x13 ; VGA 显卡

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

调用十号中断来指定显卡模式, 后面的几个MOV指令用来储存和显示有关的关键信息(为什么不直接用常量呢?), 在bootpack.c中有

struct BOOTINFO

{

char cyls, leds, vmode, reserve;

short scrnx, scrny;

char \*vram;

};

struct BOOTINFO \*binfo = (struct BOOTINFO \*) 0x0ff0;

这里就将结构体指针指向了这些信息, 可以知道VRAM的地址是0xa0000想指定地址写入数据, 当然是用指针, 将VRAM声明为一个数组即可.

作者顺便指出了数组的操作array[index] 仅是一个语法糖, a[i] 等价于\*(a + i), 当然也等价于\*(i + a), 所以很神奇的, a[i] 和 i[a] 在C语言里是等效的.

* **设定调色板**

作者替我们指定了300x200 8位的颜色模式, 如何将 0xffffff 的颜色值映射0~0xfff的空间里? 一一对应显然是不可能的, 事实上8位色彩模式中, 0到0xfff这255中颜色是由我们自己指定的, 譬如说0可以对应红色0xff0000, 这种对应关系显然被储存起来, 这就是**调色板(palette)**,作者给出了16中颜色的对应关系, 其中14号色被我改了, 调色板的数据结构如下(就一个unsigned char数组):

static unsigned char table\_rgb[16 \* 3] =

{

0x00, 0x00, 0x00, /\* 0:黑色\*/

0xff, 0x00, 0x00, /\* 1:亮红\*/

0x00, 0xff, 0x00, /\* 2:亮绿\*/

0xff, 0xff, 0x00, /\* 3:亮黄\*/

0x00, 0x00, 0xff, /\* 4:亮蓝\*/

0xff, 0x00, 0xff, /\* 5:亮紫\*/

0x00, 0xff, 0xff, /\* 6:浅亮蓝\*/

0xff, 0xff, 0xff, /\* 7:白色\*/

0xc6, 0xc6, 0xc6, /\* 8:亮灰\*/

0x84, 0x00, 0x00, /\* 9:暗红\*/

0x00, 0x84, 0x00, /\* 10:暗绿\*/

0x84, 0x84, 0x00, /\* 11:暗黄\*/

0x00, 0x00, 0x84, /\* 12:暗青\*/

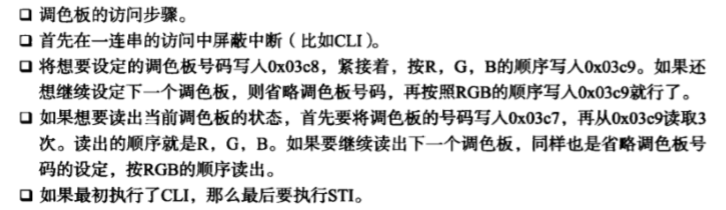
0x84, 0x00, 0x84, /\* 13:暗紫\*/

0x5b, 0x9b, 0xd5, /\* 14:浅灰蓝5B,9B,D5\*/

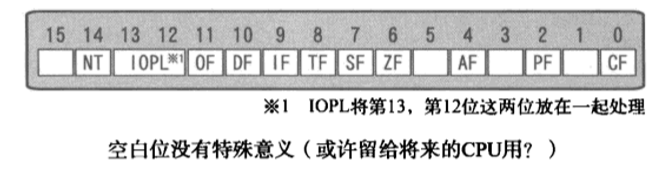
0x84, 0x84, 0x84 /\* 15:暗灰\*/

};

如何让我们设置调色板生效? 作者给出了如下步骤:



这里的0x03c9 是设备端口号, 屏蔽和恢复中断使用CLI和STI两个指令, 在naskfunc.nas里面封装成了函数io\_cli(); 和 io\_stl(), CLI(Clear interrupt flag) 将中断标志(interrupt flag)置零, STI(Set interrupt flag)则是将中断标志置一. 设置调色板的时候需要屏蔽中断, 但不知道当前的状态是什么, 所以在CLI之前要保存中断标志的状态. 中断标志保存在32位寄存器EFLAGS上, 由16位的FLAGS扩展而来, 下图应该是FLAGS 的示意图:中断标志位于第9位. 0 位是进位标志, 2是奇偶标志, 4 是辅助进位标志, 6是零标志, 7是符号标志, 8,10额…忘了, 11是溢出标志…使用指令PUSHFD和POPFD储存和恢复整个EFLAGS的状态, 封装为io\_load\_eflags()函数和io\_store\_eflags(eflags)函数.



对端口的读写要使用IN 指令和 OUT指令, 注意这里的IN是读取而OUT是写入.

结合以上的知识, 设置调色板的函数如下:

void set\_palette(int start, int end, unsigned char \*rgb)

{

int i, eflags;

eflags = io\_load\_eflags(); /\*记录中断标志的值 \*/

io\_cli(); /\*禁止中断 \*/

io\_out8(0x03c8, start);

for (i = start; i <= end; i++)

{

io\_out8(0x03c9, rgb[0] / 4); // 这里不知道为什么/4

io\_out8(0x03c9, rgb[1] / 4);

io\_out8(0x03c9, rgb[2] / 4);

rgb += 3;

}

io\_store\_eflags(eflags); /\* 恢复中断标志\*/

return;

}

有个疑惑便是rgb[0] 为什么要除以四再写入, 如果去掉这一句颜色会变得很灰…有谁知道的还望告知…(/4不就是往左移动两位?)

* **绘制界面**

我们写了一个叫做boxfill8的函数用来绘制矩形, 并用它来构建我们的基本界面:

二维屏幕和一维的地址换算关系如下: addr = 0xa0000 + x + y \* 320

void boxfill8(unsigned char \*vram, int xsize, unsigned char c, int x0, int y0, int x1, int y1);

参数的意思分别是: 显存地址, 换算参数(屏幕宽度), 颜色, 矩形的四个点.

使用的以下的代码画出界面:

void init\_screen(char \*vram, int x, int y)

{

boxfill8(vram, x, COL8\_008484, 0, 0, x - 1, y - 29); //绘制桌面填充灰蓝色

boxfill8(vram, x, COL8\_C6C6C6, 0, y - 28, x - 1, y - 28); //第一条阴影

boxfill8(vram, x, COL8\_FFFFFF, 0, y - 27, x - 1, y - 27); //绘制第二条阴影

boxfill8(vram, x, COL8\_C6C6C6, 0, y - 26, x - 1, y - 1); //绘制任务栏主体

boxfill8(vram, x, COL8\_FFFFFF, 3, y - 24, 59, y - 24); //绘制"开始"按钮 周围六条直线

boxfill8(vram, x, COL8\_FFFFFF, 2, y - 24, 2, y - 4);

boxfill8(vram, x, COL8\_848484, 3, y - 4, 59, y - 4);

boxfill8(vram, x, COL8\_848484, 59, y - 23, 59, y - 5);

boxfill8(vram, x, COL8\_000000, 2, y - 3, 59, y - 3);

boxfill8(vram, x, COL8\_000000, 60, y - 24, 60, y - 3);

boxfill8(vram, x, COL8\_848484, x - 47, y - 24, x - 4, y - 24);//绘制状态栏 四条直线

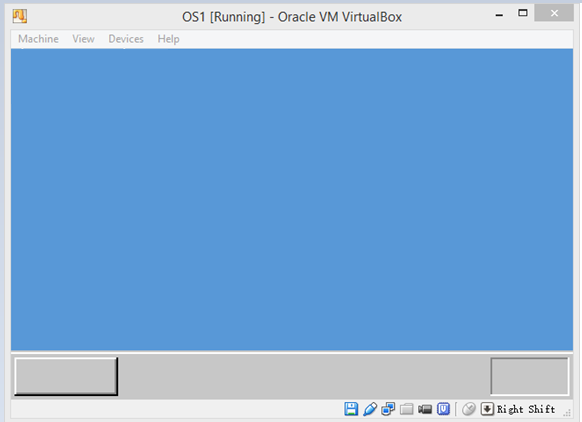
boxfill8(vram, x, COL8\_848484, x - 47, y - 23, x - 47, y - 4);

boxfill8(vram, x, COL8\_FFFFFF, x - 47, y - 3, x - 4, y - 3);

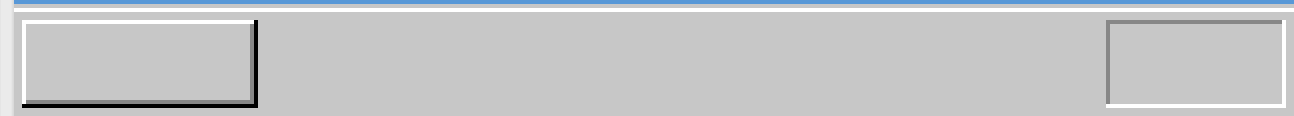
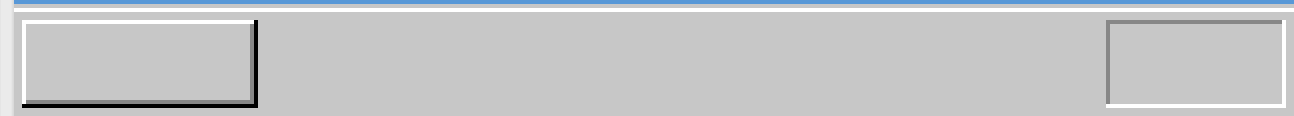
boxfill8(vram, x, COL8\_FFFFFF, x - 3, y - 24, x - 3, y - 3);

}

得到的界面如下:



屏幕下方左右两个”按钮”放大看:

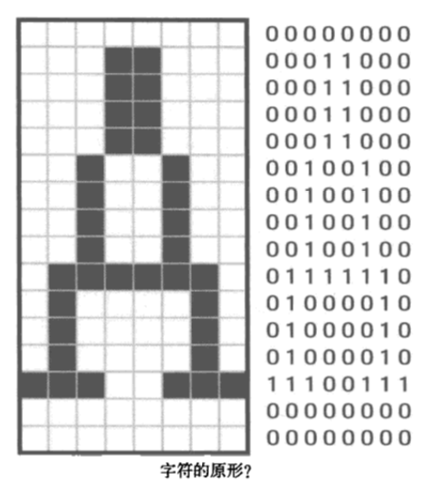
其实就是在原来的底色上画多了几条直线, 左边的是6条, 右边的是4条.

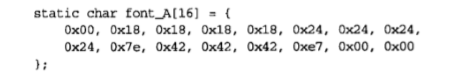
看情况, 这将会是我们的操作系统的桌面.

* **显示字符**

使用了asmhead.nas 后我们进入了32位模式(似乎是保护模式), 不可以使用int 0x13中断了, 因此要显示字符只能自己画.

作者使用8x16 的矩阵来表示一个字符, 不知道真正的字体文件\*.ttf是否用这样的方式储存…



换算成16进制是: 

这样的表示方法难以调整, 所以作者使用了一个文本文档来描画字体, 在这个文本文档里面8是: char 0x38

........

...\*\*...

..\*..\*..

.\*....\*.

.\*....\*.

.\*....\*.

..\*..\*..

...\*\*...

..\*..\*..

.\*....\*.

.\*....\*.

.\*....\*.

..\*..\*..

...\*\*...

........

........

然后伟大的作者又搬出了自己的”编译器”, makefont.exe, (╯‵□′)╯︵┻━┻), 神烦的”别人家的编译器”, 这也意味这Makefile又要改动了…如何画出字符和字符串呢?

写一个很厉害的putfont8函数:

void putfont8(char \*vram, int xsize, int x, int y, char c, char \*font)

{

int i;

char \*p, d /\* data \*/;

for (i = 0; i < 16; i++) {

p = vram + (y + i) \* xsize + x;

d = font[i];

if ((d & 0x80) != 0) { p[0] = c; }

if ((d & 0x40) != 0) { p[1] = c; }

if ((d & 0x20) != 0) { p[2] = c; }

if ((d & 0x10) != 0) { p[3] = c; }

if ((d & 0x08) != 0) { p[4] = c; }

if ((d & 0x04) != 0) { p[5] = c; }

if ((d & 0x02) != 0) { p[6] = c; }

if ((d & 0x01) != 0) { p[7] = c; }

}

return;

}

让特地的数和字体数据相与,判断特定位是否为1, 很厉害的做法.

显示字符串的函数:

void putfonts8\_asc(char \*vram, int xsize, int x, int y, char c, unsigned char \*s)

{

extern char hankaku[4096];

for (; \*s != 0x00; s++) {

putfont8(vram, xsize, x, y, c, hankaku + \*s \* 16);

x += 8;

}

return;

}

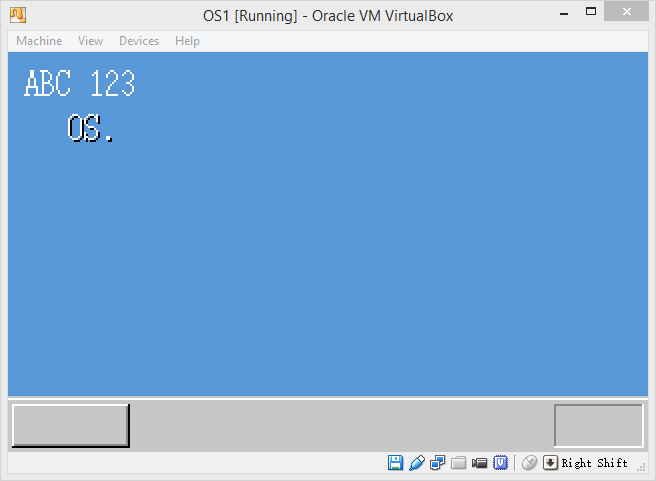
在主函数里调用

putfonts8\_asc(binfo->vram, binfo->scrnx, 8, 8, COL8\_FFFFFF, "ABC 123");

putfonts8\_asc(binfo->vram, binfo->scrnx, 31, 31, COL8\_000000, "OS.");

putfonts8\_asc(binfo->vram, binfo->scrnx, 30, 30, COL8\_FFFFFF, "OS.");

结果如下:



* **显示变量值**

为了显示变量值, 作者使用了名为GO的编译器里的sprintf函数,(这货是GO语言的GO吗?), Sprintf函数就是将指定内容格式化输出到字符流中, 然后我们再将这个字符显示出来, 达到显示变量内容的目的.

添加头文件stdio.h, 主函数加入以下语句:

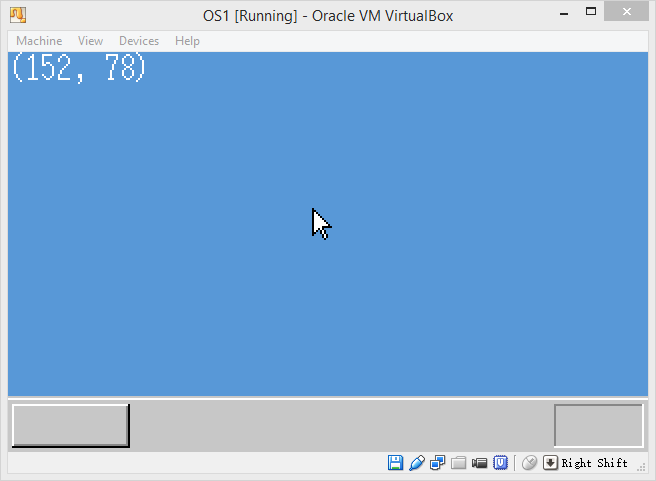
int mx = (binfo->scrnx - 16) / 2; //水平居中

int my = (binfo->scrny - 28 - 16) / 2; //排除任务栏垂直居中

sprintf(s, "(%d, %d)", mx, my);

putfonts8\_asc(binfo->vram, binfo->scrnx, 0, 0, COL8\_FFFFFF, s);

效果如下:



* **显示鼠标指针**

鼠标的显示和字符大同小异, 但是这里使用字符来储存而不是二进制, 因为需要填充色, 边缘色和背景色…作者画的鼠标实在难看, 我用的是@BIT\_祝威 写好的鼠标, 特此感谢!

static char cursor[16][16] = {

"\*...............",

"\*\*..............",

"\*O\*.............",

"\*OO\*............",

"\*OOO\*...........",

"\*OOOO\*..........",

"\*OOOOO\*.........",

"\*OOOOOO\*........",

"\*OOOOOOO\*.......",

"\*OOOO\*\*\*\*\*......",

"\*OO\*O\*..........",

"\*O\*.\*O\*.........",

"\*\*..\*O\*.........",

"\*....\*O\*........",

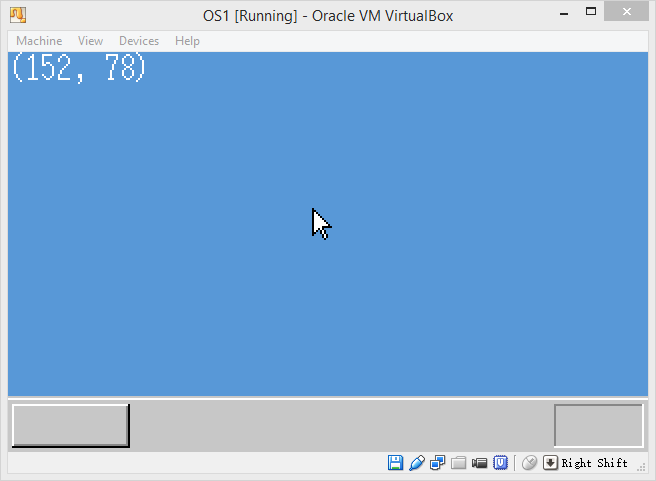
".....\*O\*........",

"......\*........."

};

事前将上面的图案处理成颜色信息cursor[], 再使用putblock8\_8函数将鼠标画出来, 效果

如下:



这次就做到这里…按书中的指示将代码分装成几个文件, 下面是当前的全部代码: