# 操作系统实验日志

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学号 | 201808010718 | 姓名 | 肖鹏 | 专业年级班级 | 智能1802 |
| 实验日期 | 2020.10.15 | 实验项目 | 第4天：C语言与画面显示的练习 | | |

## 一、实验主要内容

1、辅助画面显示的tools函数实现

**内容：**

1、*void write\_mem8(int addr, int data)*

该函数功能是向指定地址写入传入的参数data中低8-bits数据；

2、*void init\_palette(void)*

初始化画板，将16中颜色的RGB值保存在一个16 by 3的数组中，调用set\_palette按这16种颜色设定画板；

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

该函数中涉及到对IO设备（此处应该是显示器）的数据写入，而且在开始对端口写入之前需要屏蔽中断，防止其他中断干扰进程执行，在程序最后需要恢复被屏蔽的中断。这里用到对IO端口写入的函数是io\_out8(int port, int data)，在for循环中依次对16个颜色RGB值写入设备端口0x03c9，每次*rgb+=3*意思是往后一个RGB单位。对于端口号0x03c8和0x03c9这两个值，书中的解释是这是固定的设备号码，不可更改，使用前需仔细调查，但可惜的是作者给出的参考网页已经失效，笔者几番搜索之后也无实质性的发现，便暂时搁置这一个点。

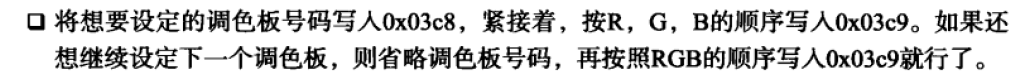
4、io系列功能函数

除了io\_out8(int port, int data)之外，还有一系列io的函数，由于代码简洁短小，放在截图中概况。

**重点总结：**

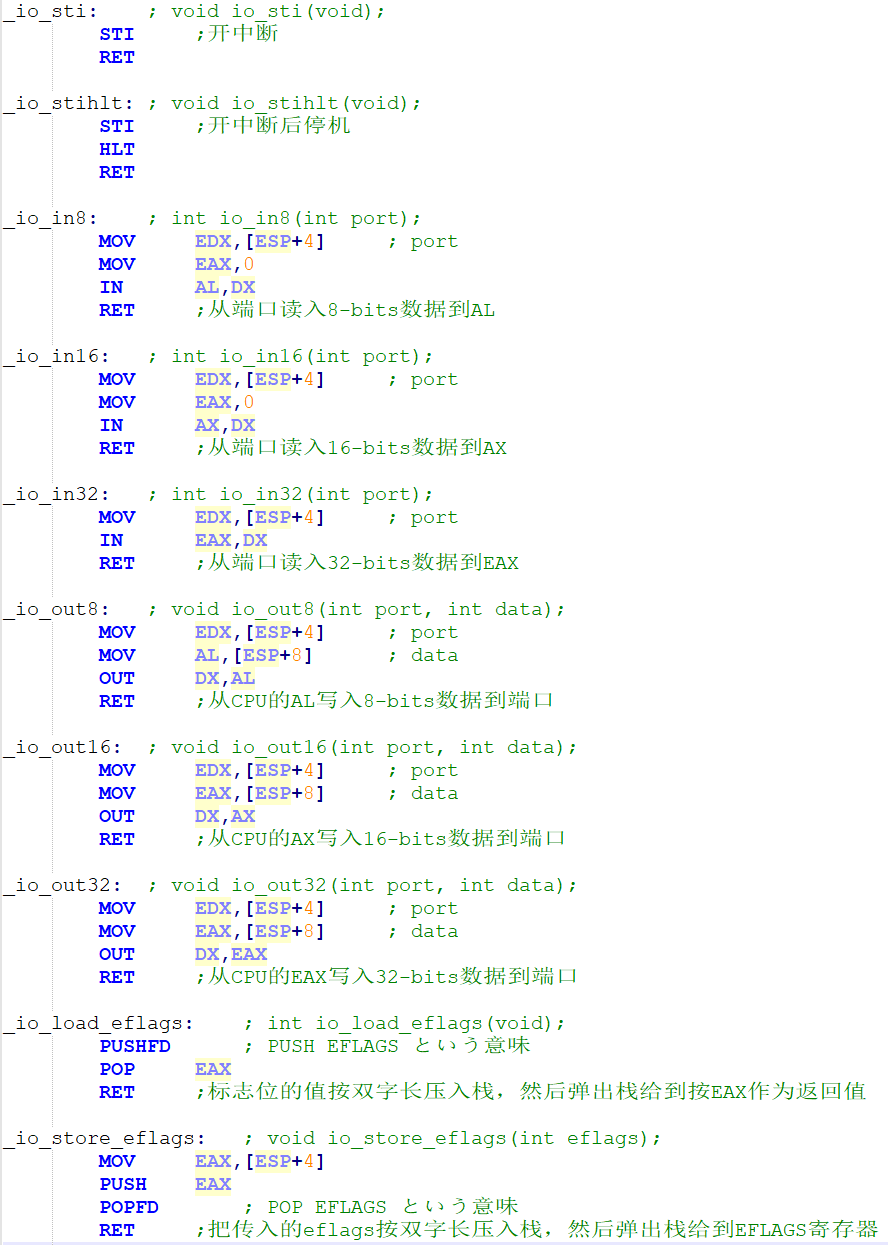
a）EFLAGS寄存器与EAX之间不能直接的互换数据，需要通过栈来作为中转；

b）上述函数中较不好理解的是set\_palette(int start, int end, unsigned char\* rgb)，主要疑惑是为什么在循环中重复向0x03c9写了16次？虽然书中没有对两个设备端口号做具体阐述，但是后面有一段使用描述，个人理解如下：



屏蔽中断后，向端口0x03c8写入将要设定的调色板号码，然后向端口0x03c9写入RGB值；之后我们需要继续设定下一个调色板，根据书中的说明，此时应该忽略调色板号码，再次将RGB值写入0x03c9端口即可。

**关键代码及注释：**



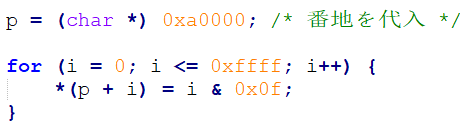
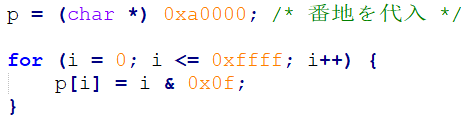
2、几种实现“条纹”的指针写法

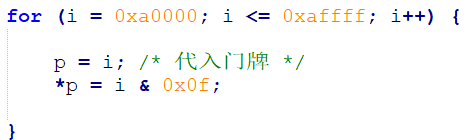
**内容：**

遍历地址空间，且每一个地址指向的值被写为Addr&0x0f，即16种颜色范围内选取一种。从而形成条纹的形状。

**重点总结：**由于我们对指针都比较熟悉，所以这一部分没有理解上的难点；

**关键代码及注释：**





3、绘制矩形和简易界面

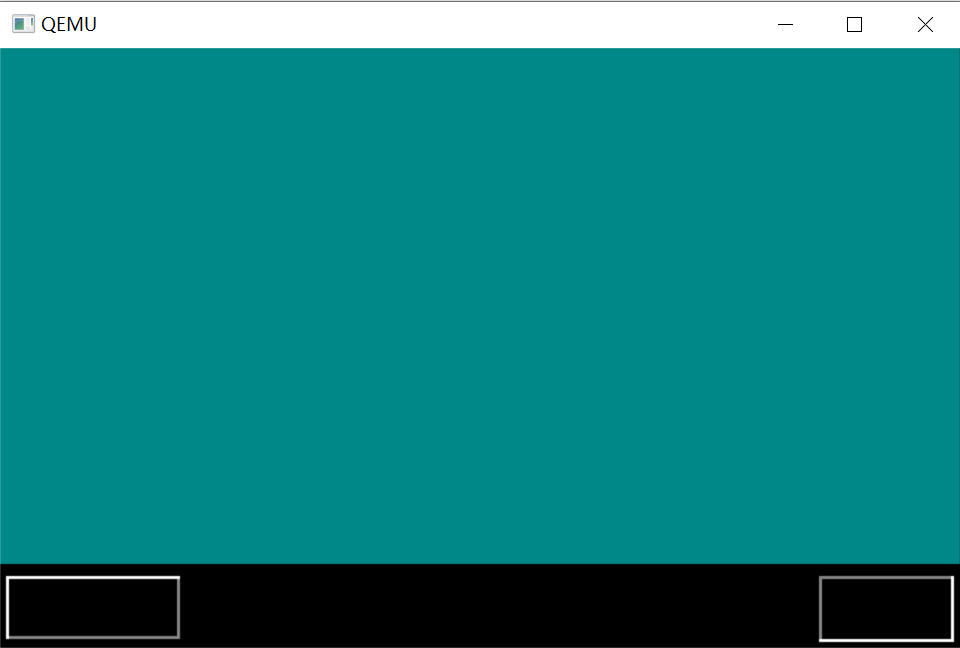
**内容：**

绘制矩形涉及到像素坐标，由于采用的是320\*200的8位模式，一共有6400个像素，以左上、右下对角为（0，0）、（319，199），坐标对于的VRAM计算为：

*ADDR=0xa0000+x+y\*320*

对于之前用到调色板的16种颜色，在这里用宏定义声明了一遍便于使用；

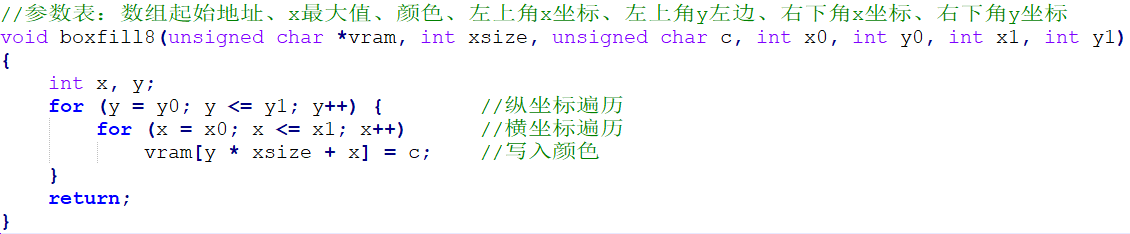
由于需要对矩形区域绘制，需要用到辅助函数boxfill8函数（其实就是双重循环）；在绘制简易界面的时候，就是分成了四个矩形：蓝色区域大矩形、任务栏、开始按钮边框、时间边框（为了便于看出，将任务栏删除，见下图），其中开始键、时间区的边框绘制又是由小矩形组成的（具体步骤见下面的代码注释）。

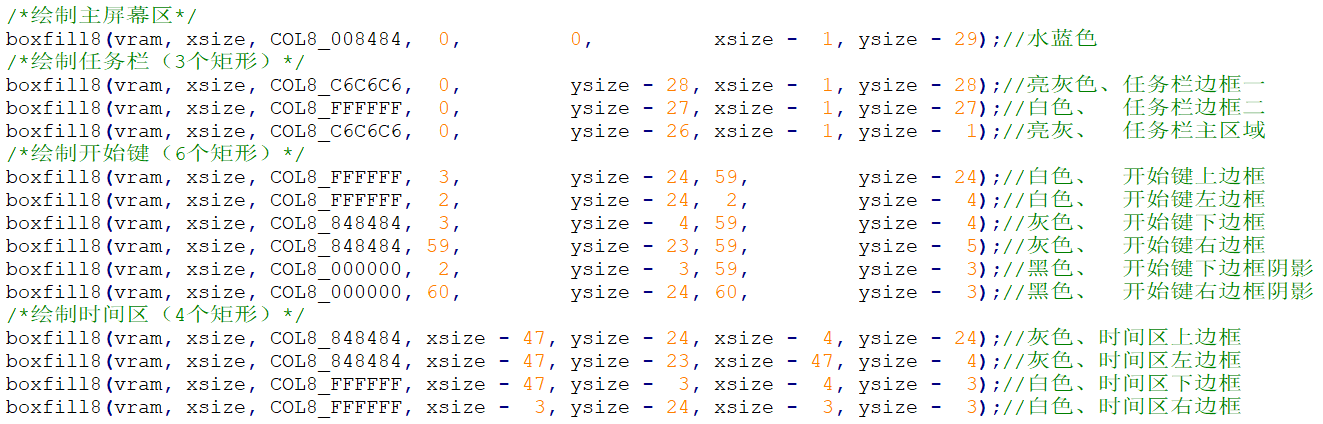


**重点总结：**

这一部分也比较简单，主要就是划分区域（见代码注释），利用boxfill8函数进行绘制。

**关键代码及注释：**





## 二、遇到的问题及解决方法

1、set\_palette函数中rgb[0]/4、rgb[1]/4、rgb[2]/4为什么要除以4?

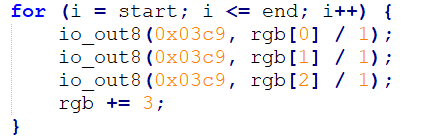
**猜想：**

实验使用的320x200 8-bits 的VAG模式只支持6-bits的R/G/B值，也就是对于RGB值，尽管有8-bits用来存储，但是只能靠低6位来调整颜色，所以需要将原有的8-bits数据舍弃两位，即除以4，把原来高6位转为低6位用来分辨颜色；

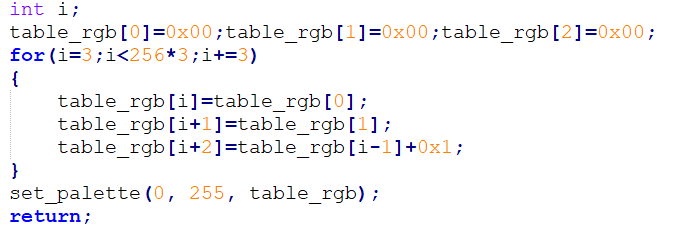
**验证：**

下面通过一些代码来验证我们的猜想

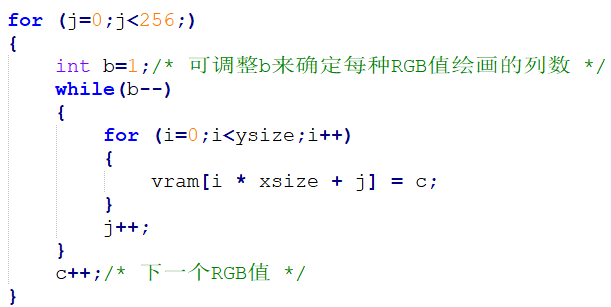
①首先将set\_palette中的/4去掉：



②生成256中颜色供显示，并画入调色板（以黑色为底色，B值变化0~256）：



③绘制256中颜色，在320x200的画面中，以列为绘制单位，每一种颜色占据一列，只画256列，剩余列不处理：

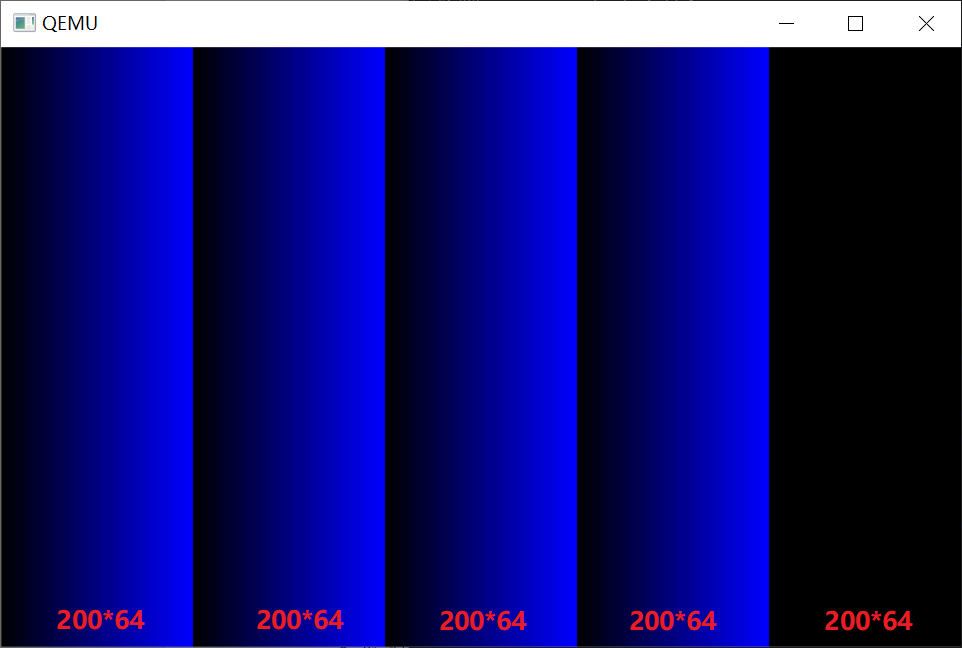


**预期结果：**

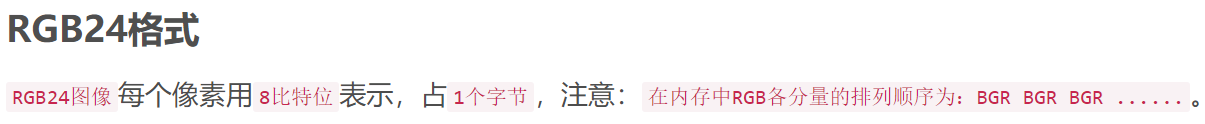
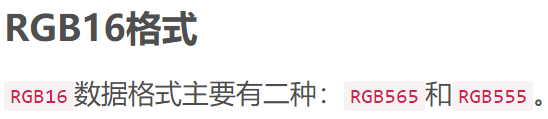
如果我们的理解没有错，那么应该出现的是，颜色以64列为周期变化：从“黑”->蓝（因为我们初始RGB是0x000000，最终RGB是0x0000ff）；在所绘制的256列中，应该回出现256/64=4个重复的色块，运行一下验证验证！

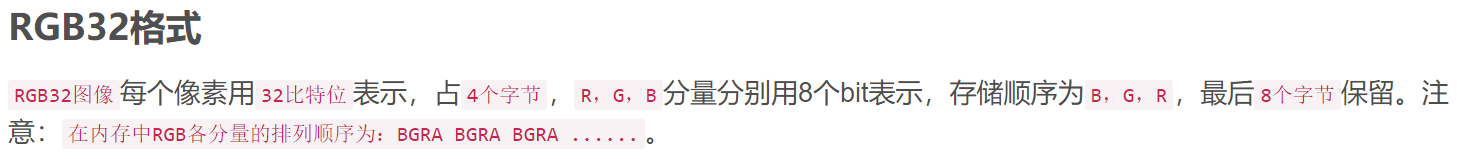
**实验结果：**

与我们猜想的相同，并没有出现256中RGB分别对照的颜色，而是以64个颜色周期显示；



**结论：**在网上搜了一下，常见的RGB显示格式一般是采用以下：





以RGB16中的RGB565为例，如果想写入这种格式的RGB值，需要以下操作：



但实验中这种格式应该是RGB18中的RGB666，所以需要取高6-bits，也就是（初始RGB值/4）；

## 三、程序设计创新点

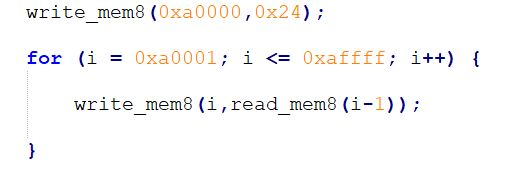
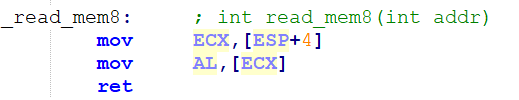
1、实现一个读取函数

**创新点：**

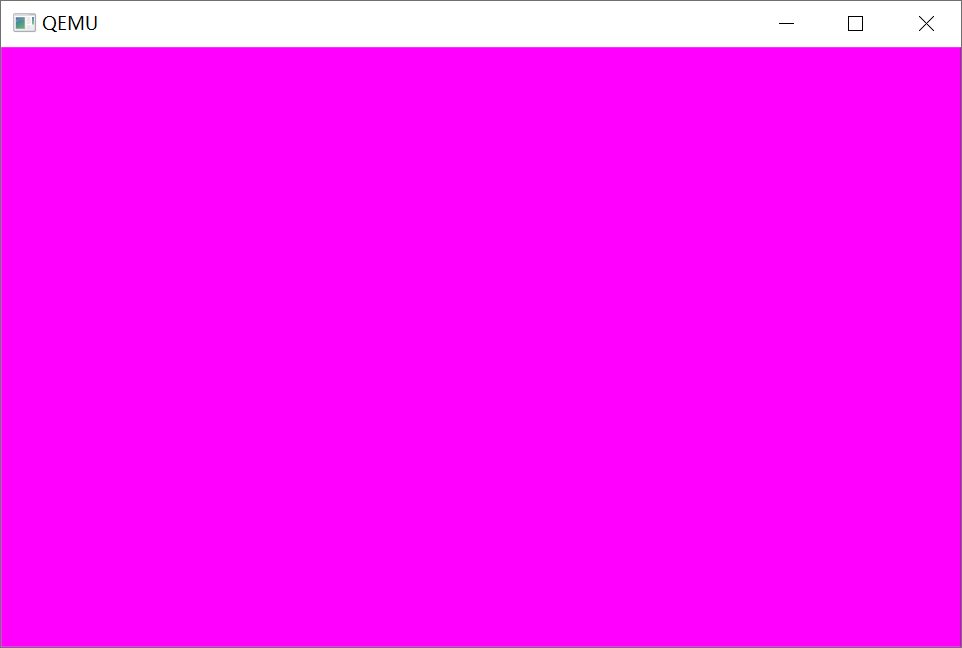
这是一个比较小的点，在实验的haribote01a文件夹中有一个write\_mem8的函数，实现了向指定内存写入的，然后自然而然地就想写一个从指定内存读数据的函数（做到后面才发现作者在后面的实验中实现了一系列的io函数……不过鉴于这次的素材比较少，就把这个写进来凑内容了，也当做是通过汇编声明c函数的初体验吧）

怎么验证汇编函数是否写对了呢？可以先向VRAM的初始地址写一个值，然后后面的所有上色操作都是靠read\_mem8函数来读取前一个颜色，相当于是把初始颜色复制了64000份，只要最终显示的只有一种颜色，那么函数验证就通过了。

**关键代码：**



**结果截图：**

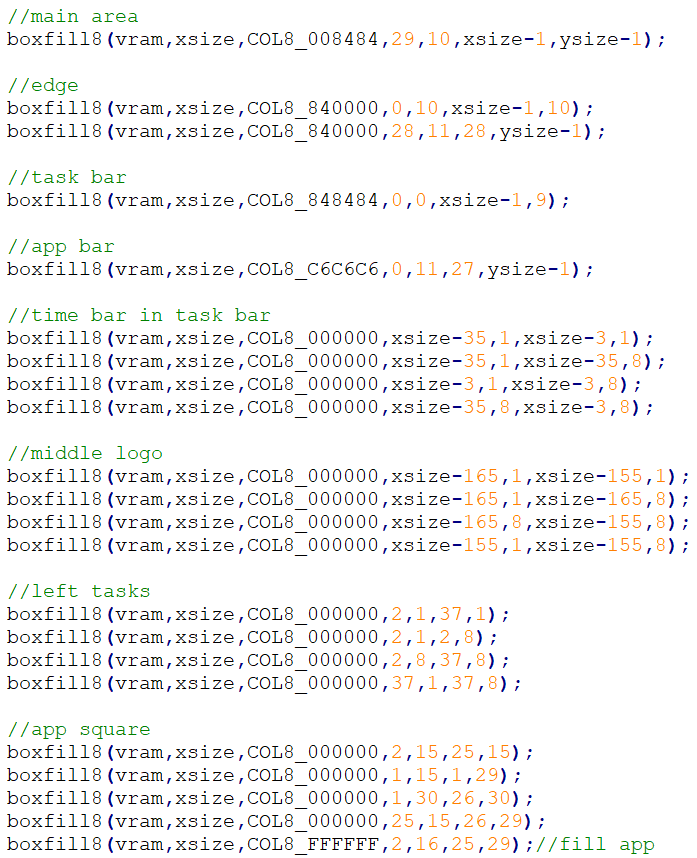


2、实现Ubuntu风格的简易界面

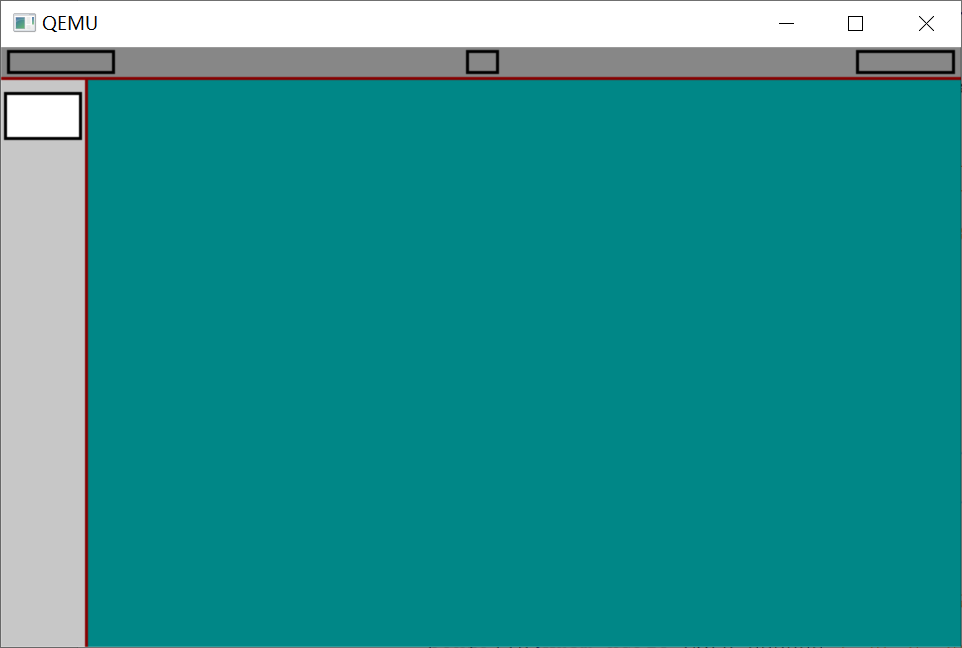
**创新点：**

本次实验中haribote01h中实现的简易操作界面，在测试之后觉得自己也可以尝试绘制一个界面，函数方面用boxfill8()就够了，其他的完全就是靠自己的“绘画”感觉，没有啥难度，就直接上效果图吧。

**关键代码：**



**结果截图：**



## 四、实验心得体会

这次实验整体来讲不是很难，主要是引入在VGA模式下的绘制图形功能，涉及到的函数都是一些比较基本的功能函数，没有难以理解的，最终的实现也只是一些矩形的拼凑实现一个简易的界面，所以内容比较少。