# 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 | 2019.11.25 | 实验项目 | 第10天：叠加处理 | | |

## 一、实验主要内容

**1、以4k为单位的内存管理**

**内容：**

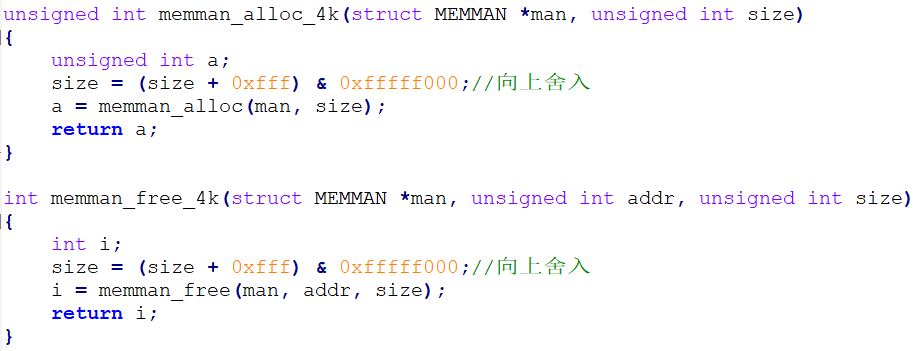
a）由于之前的memman\_alloc和memman\_free都是以1-byte为单位处理，但是在实 使用时，我们的每一次分配往往都远大于1-byte，经常用到的是对4k-byte整数倍的内 存进行处理，所以我们为了方便使用，同时也为了防止小内存的频繁分配、释放造成 对free列表的消耗，封装两个4k-byte的memman\_alloc和memman\_free函数。

b）这里在具体给一个内存请求分配多少个4k时做出了一个向上取正的操作；也就是 说就算一个请求只申请了4k+1-byte大小的内存，那么我们也照样给他4k+4k大小的内 存，虽然这样会造成浪费，但是是采用这种方式避免不了的。

**重点总结：**

a）这部分比较简单，所涉及的两个新函数也只是对size做了一个向上舍入，然后调用 已有的函数；其中向上舍入应该很好理解，书上花了大量笔墨把向上舍入和向下舍入 讲的很仔细。

**关键代码及注释：**

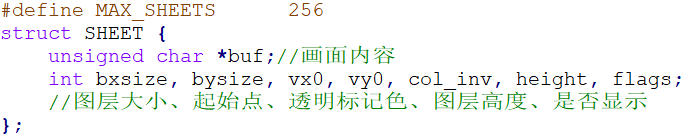


*（memman\_alloc和memman\_free）*

**2、图层设计**

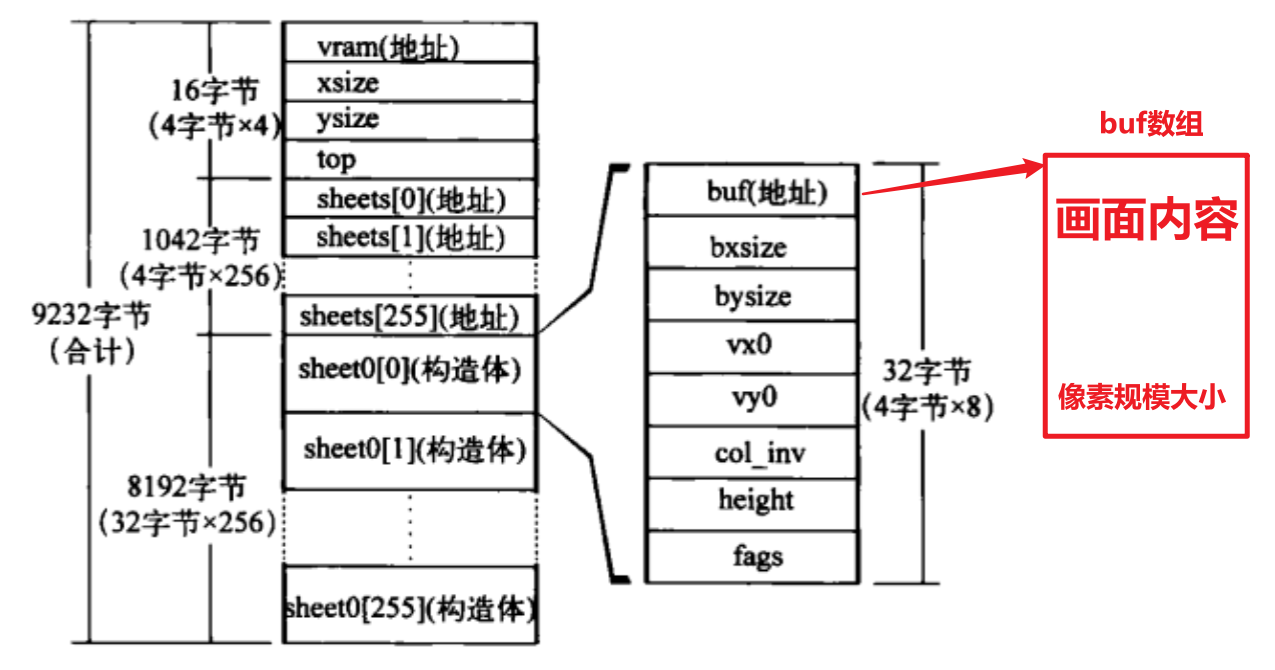
**内容：**

a）整体想法是，分别创建两个结构体——首先是图层结构体，用于记录该图层上的内 容信息、图层管理结构体，用于管理刚刚所创建的图层结构体数组。对于图层结构体， 其中需要包含图层的画面内容（这里并没有实际在每个图层实例中保存画面的具体内 容，而是保存了一个指向画面内容的一个字符数组指针），以及图层的大小、起始点、 透明标记、图层高度、是否显示；这里对后两者做简要解释：图层高度，比如我们在 windows中，打开多任务界面后，桌面就是高度最低的。所以高度最高的就是完整呈 现在面前的界面。绘制的顺序也是从低图层开始画，这意味着低图层可能被覆盖，具 体的不同覆盖情况之后会讨论。

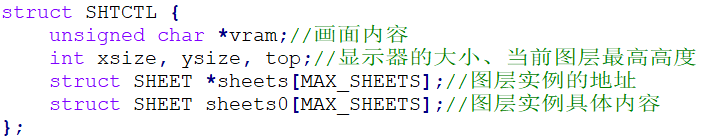


*（图层结构体）*

b）另一个结构体——图层控制结构体，我们只需要一个图层控制结构体实例即可控制 之前声明的256个图层实例。一个图层控制结构体实例，需要包含当前的画面内容（同 样是只存放一个指针），这个指针指向一个320\*200-byte大小的数组。



*（SHTCTL结构示意图）*



*（图层控制结构体）*

**重点总结：**

a）对指定图层的移动，可分为以下情况讨论注意事项：

**1. 上移图层**

**1.1 从不可显示区上移到可显示区**

**注意可显示的图层数增加**

**1.2 可显示区内移动**

**2. 下移图层**

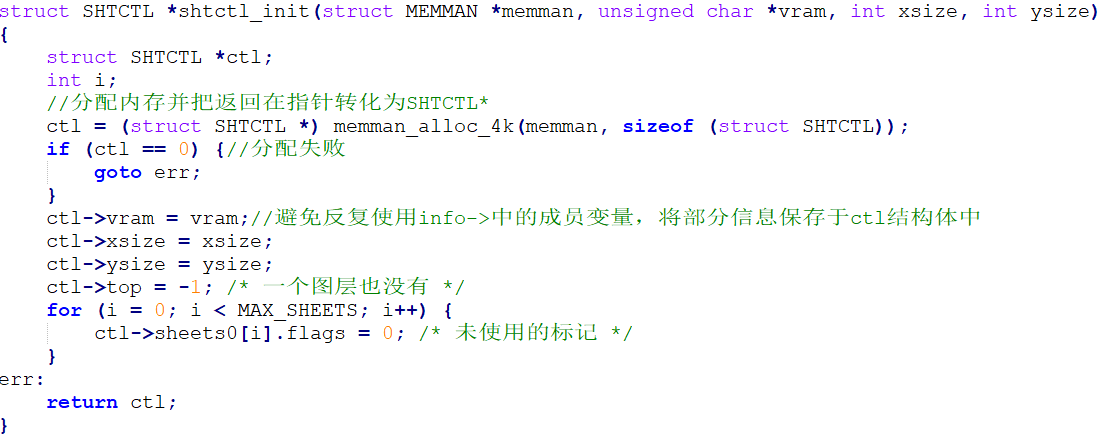
**2.1 从可显示区下移到不可显示区**

**注意可显示区的图层数减少**

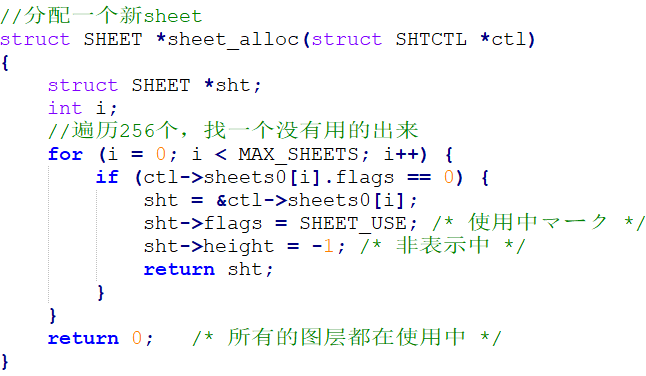
**2.2 可显示区内移动**

四种情况都需要对图层列表的重新排序

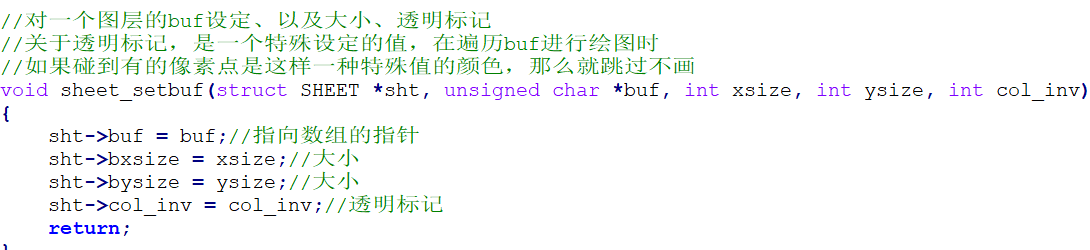
**关键代码及注释：**



*（初始化图层控制实例）*



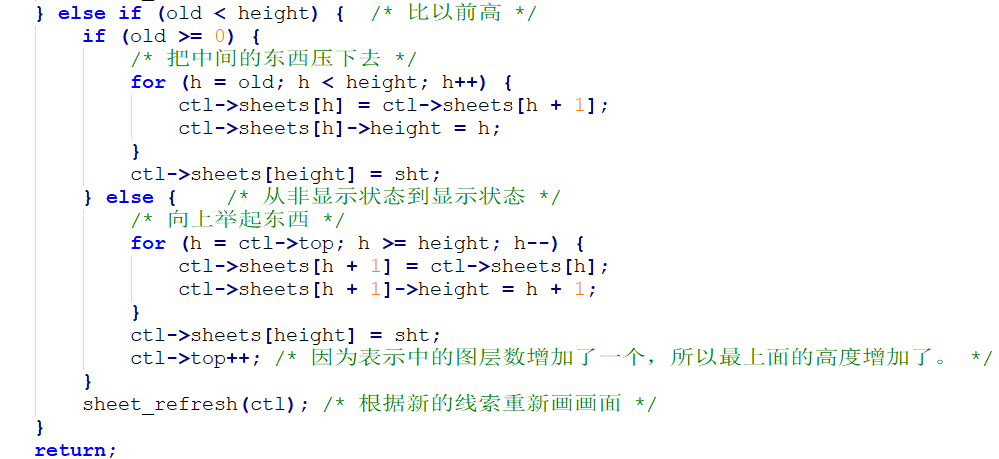
*（分配新图层）*



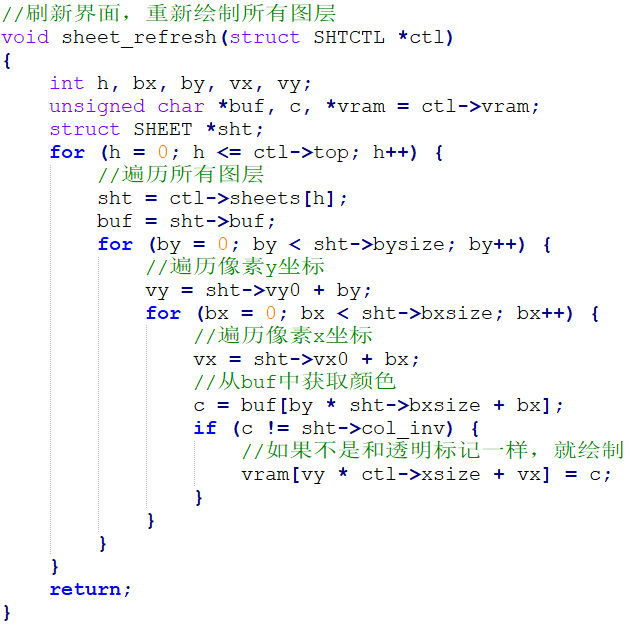
*（设定图层的内容缓冲信息）*



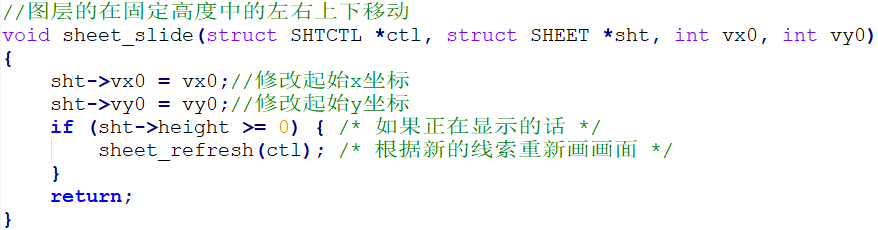
*（对指定图层的优先级移动1）*



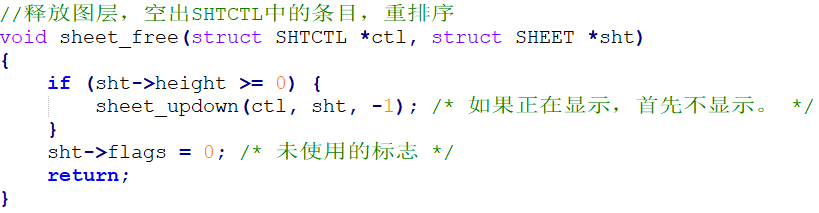
*（对指定图层的优先级移动2）*



*（刷新界面）*



*（图层的移动）*



*（图层的释放）*

**3、图层性能优化**

**内容：**

a）由于之前在绘制的时候，我们对每个图层，都做了320\*200规模大小的遍历，然而 对于像鼠标一样这种小的图层，这完全是一种浪费；所以我们对于绘制的函数增加了 一组限定坐标(x1,y1)、(x2,y2)，来表示只需要更新的区域范围。我们只需要增加一个 sheet\_refreshsub函数，然后用这个函数代替所有调用原先refresh将函数的地方。

b）前面提到的覆盖问题， 只要判断：

1. 限制范围的vx0与图层起始的sht->vx0，如果前者小于后者，则实际更新起始bx=图 层的起始sht->vx0；否则就是从vx0开始；

2. 对纵坐标也是相似处理；

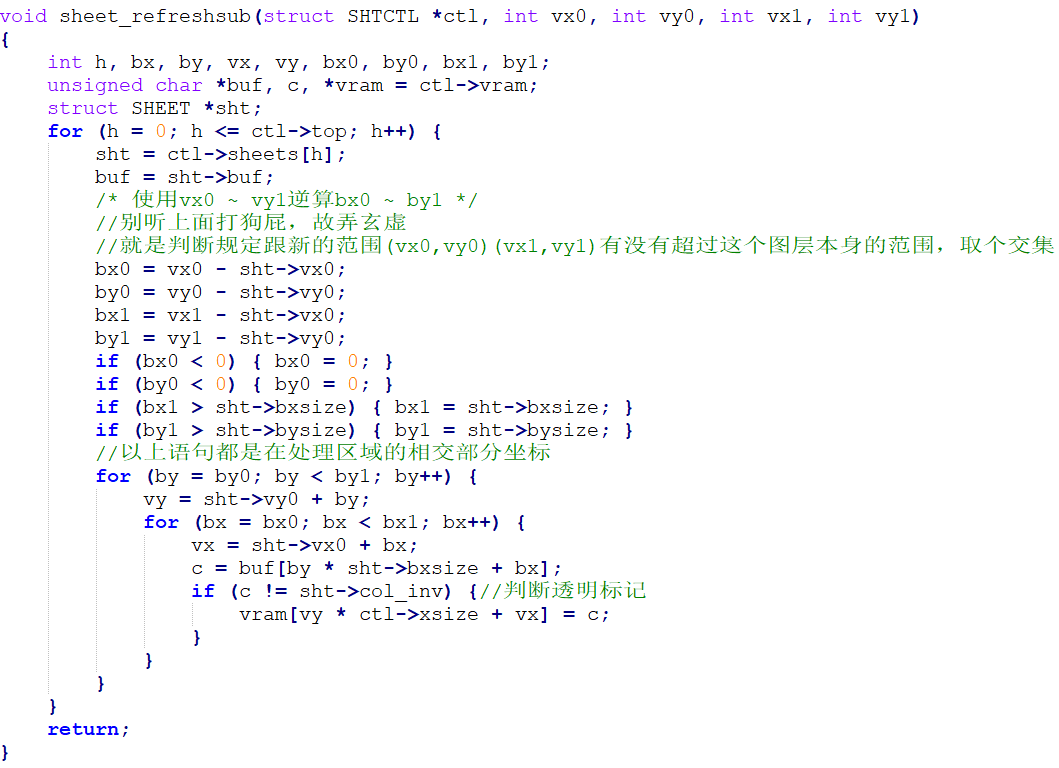
3. 这里没有显示的对限制范围的vx1、vy1与起始横坐标、纵坐标的判断，这是因为在 for循环中有隐式的判断，不成立则不进入循环。

**重点总结：**

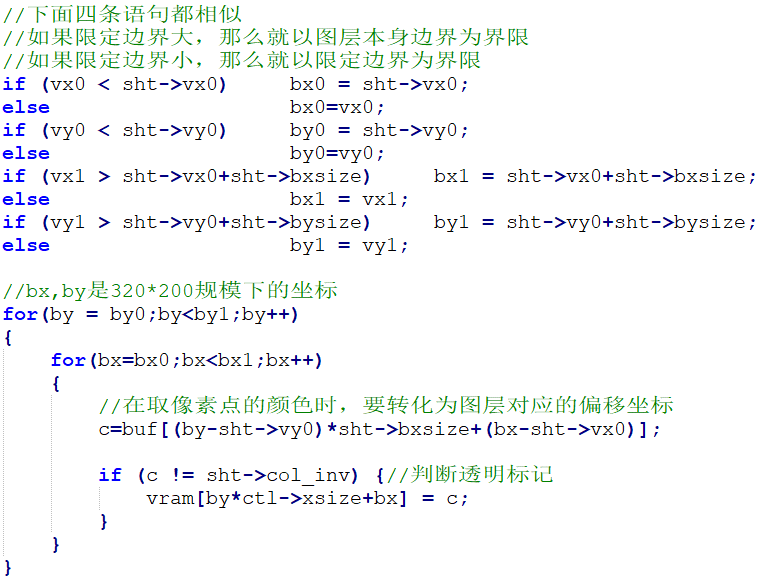
a）对覆盖问题，其实就是当限定区域和图层的规模不完全重合时的问题，需要对限定 区域针对图层做一些修改，详见代码注释。

b）对这个sheet\_refreshsub函数里面的这个区域处理写的代码很是困惑，感觉Day 10 的代码里面，明明简单易懂，却又要把代码写成让人一眼看不明白什么意思的地方有 好几处，这里最严重了。明明就是一个判断区域交集的地方，直接拿两个区域的横坐 标相比，然后在进行相应的最值获取即可，这里反反复复，又是先减去图层的大小， 再便利的时候又加上。**（详见“遇到的问题”）**

**关键代码及注释：**



*（更新后的刷新页面）*



*（简化代码）*

## 二、遇到的问题及解决方法

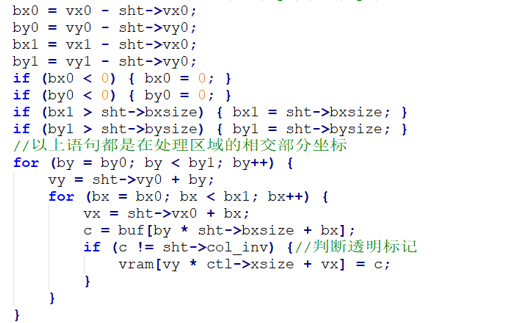
1、对判断覆盖问题的代码段感到困惑

**问题描述：**

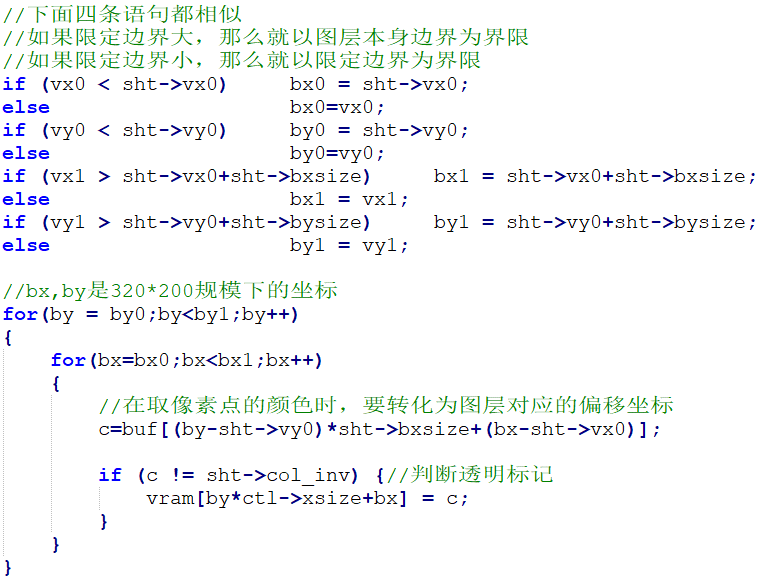
对于覆盖判断的拿一部分代码（如下），在一开始觉得很迷惑，为什么要先减去图层的 起始坐标，然后在遍历中又加上这一个值，这样不会很麻烦吗？然后我试图修改简化 这一部分代码，但是在修改的过程中，慢慢意识到，这一些操作是有原因的。

**问题解决：**

之所以要减去，是因为我们在绘图的时候用到了两个数组，一个是图层自带的储存图 层像素颜色信息的数组，其大小由图层的大小而定，如果将其看成二维的，那么它的 维度应该是[sht->bxsize, sht->bysize]，而绘图的时候，我们要用到的是info中的vram， 这可看作一个[320, 200]大小的二维数组，我们自然不能用同一组下标操作这两个数组， 即又在buf数组里取数据，又向vram中写数据，这有一点刻舟求剑的感觉，所以需要 我们把初始的坐标先减去sht->vx0/vy0，这样转化成buf数组的下标，我们可以用着一 组下标从buf取数据；但是要向ram中写数据的时候，需要用到另一组vx、vy。



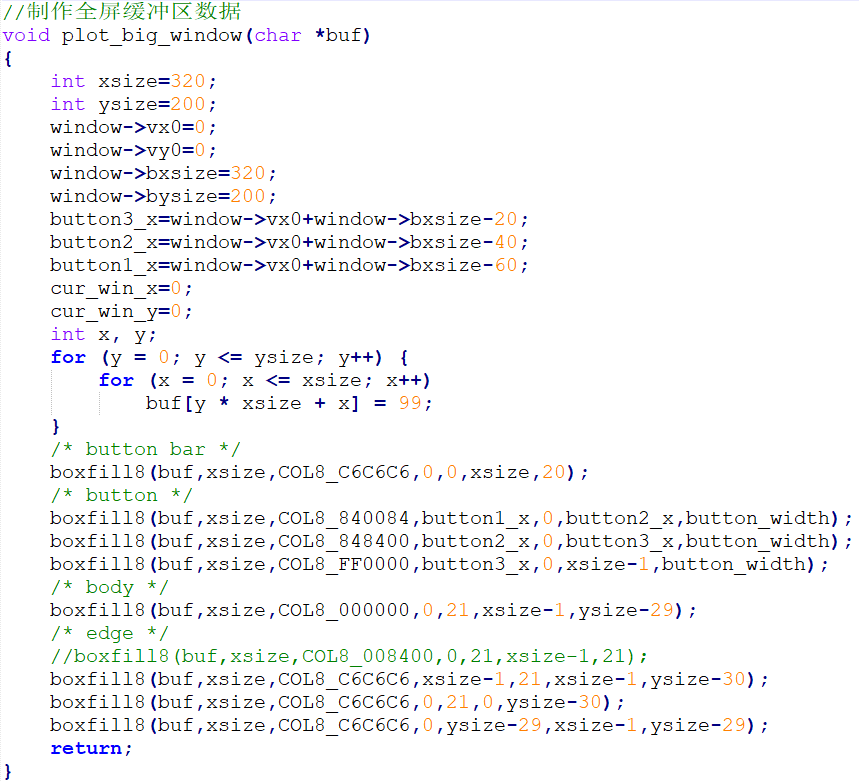
我按照以上理解，写出另一个版本的代码，即只用一组下标，但是在访问buf时作 修正，运行测试正常。



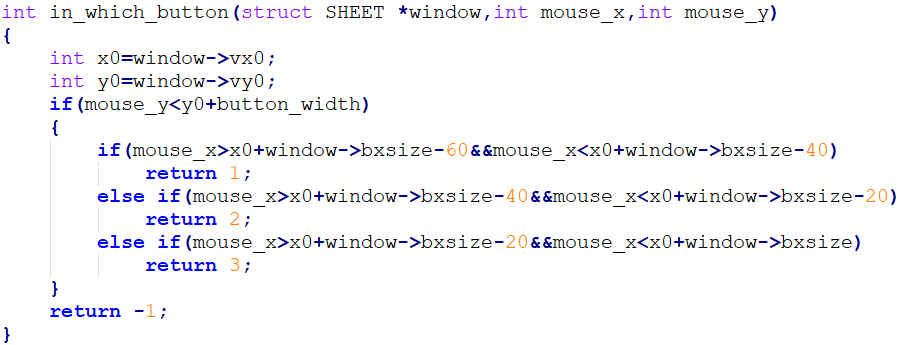
2、实现窗口的缩放以及关闭

问题描述：实现窗口操控，功能包括：①可以缩小至屏幕的1/4，可放大至全屏（不覆盖任务栏）；②可关闭窗口；③可拖动窗口；

代码实现：具体实现描述：设计一个窗口图层，特殊的，在三个小矩形中设置button，当检测到鼠标位于这三个button区域内、且按下了左键，则执行相应操作。从左往右三个矩形分别是：拖动Button（在此Button区域内，一直按着左键可以实现窗口随鼠标移动）、缩放Button、关闭Button（此操作不可逆，即一旦关闭，图层就不现实了，暂时没有实现任务栏点击可恢复功能）；



*（为绘制全屏准备画面像素信息，类似的对于1/4显示也有相应函数）*

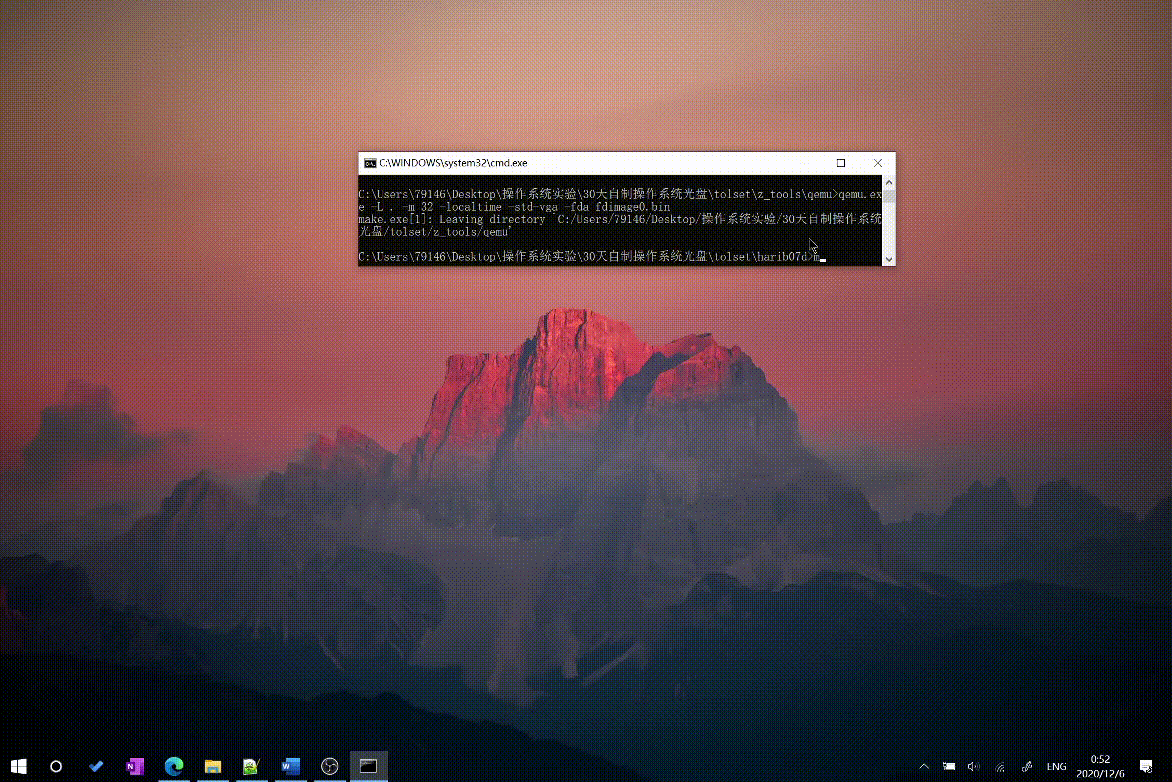


*（判定鼠标处于哪个Button中）*



*（判断Button后的分支操作）*

效果截图：



## 三、程序设计创新点

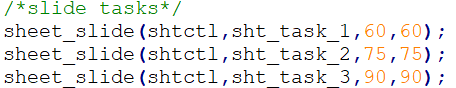
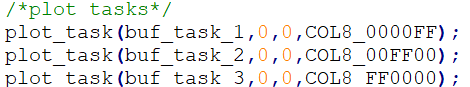
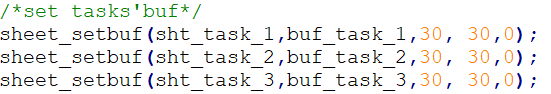
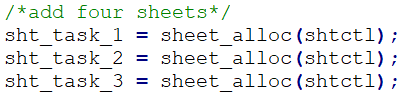
1、实现图层切换

**创新点：**

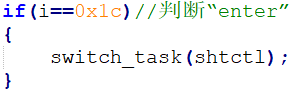
运用今天所学的图层知识，我们可以实现在不同图层之间的切换。这里我设计了一个 简单的图层切换，有三个互相覆盖的色块，我们通过按“enter”键，实现类似于通过 “Alt+Tab”键进行多任务切换。

对于这个功能的实现，主要涉及到对今天所学代码的运用。如何创建一个图层、为其 分配缓冲内存以及为其调度高度。通过这样一个功能的实现，可以很好的让我们运用 今天的知识。

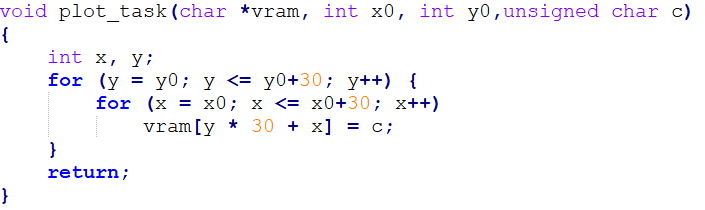
**实现代码：**



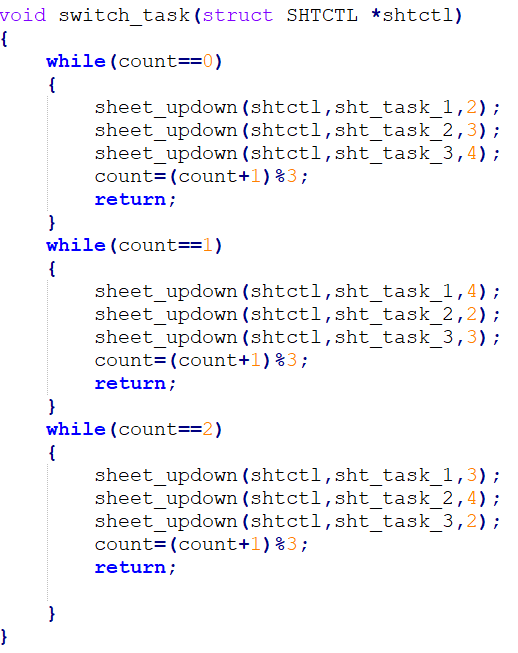
*（初始化四步骤）*



*（判断回车键，进行切换）*

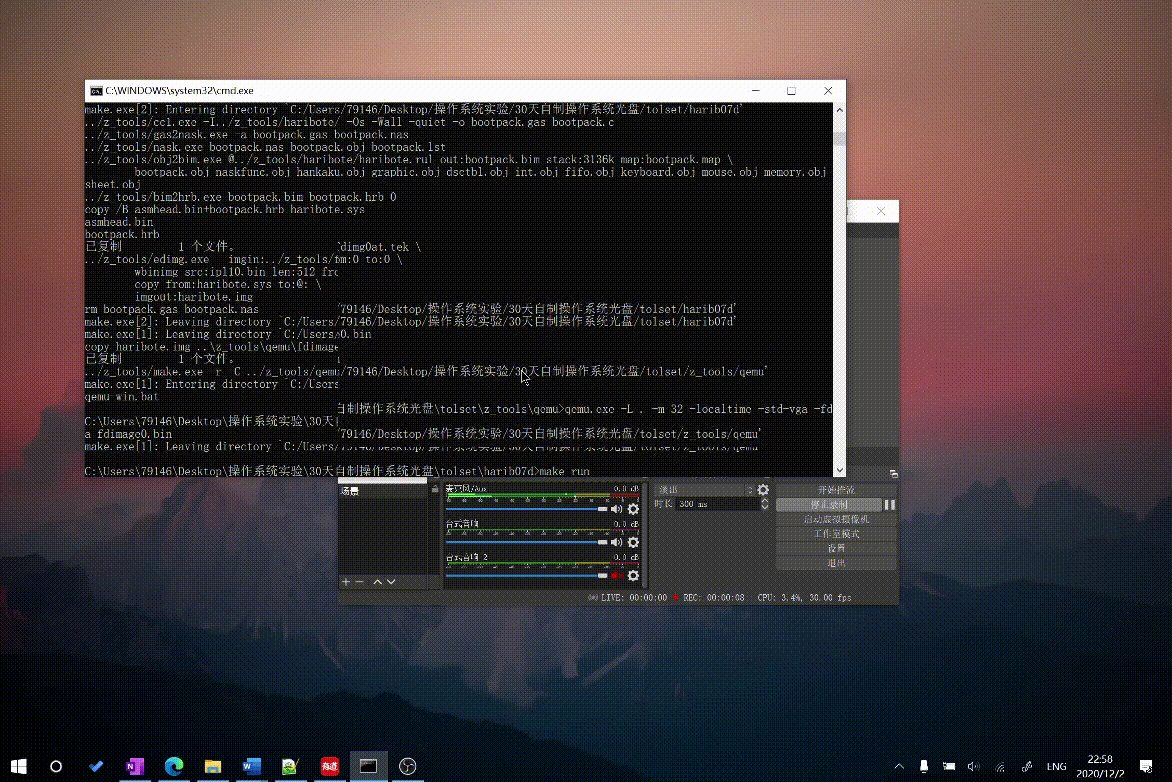


*（准备缓冲区内容）*



*（切换，使得不同的层位于最高层）*

**效果截图：**



## 四、实验心得体会

呃……这次试验挺赶的，都超过ddl了，感觉就是“积怨已久”的鼠标问题终于圆满解决，而且提供的图层方法还能为我们之后的问题提供解决方式，感觉很不错，冲冲冲！

（赶紧发给老师了……逃）