Day 10

Phase 1

内存分配按4KB为单位进行向上舍入，这样能够一定程度上减少内存碎片的产生，提高内存管理的效率。

```
unsigned int memman_alloc_4k(struct MEMMAN *man, unsigned int size)
{
    unsigned int a;
    size = (size + 0xfff) & 0xfffff000;
    a = memman_alloc(man, size);
    return a;
}

int memman_free_4k(struct MEMMAN *man, unsigned int addr, unsigned int size)
{
    int i;
    size = (size + 0xfff) & 0xfffff000;
    i = memman_free(man, addr, size);
    return i;
}
```

Phase 2

SubPhase 1

引入图层概念。

为了正确处理堆叠，我们需要引入图层的概念。图层的概念我们在日常使用计算机的过程中经常会接触到。我们可以为每个图层单独保存他们的画面，然后根据需要对整个屏幕进行绘制。

一个图层都需要什么样的信息呢?

以下信息是必须的

变量名	含义
buf	存储画面具体内容的指针
bxsize	画面横向宽度
bysize	画面纵向宽度
vx0	图层左上角位于屏幕上位置的横坐标
vy0	图层左上角位于屏幕上位置的纵坐标
col_inv	透明色号
height	图层纵向顺序编号
flags	其他设定信息

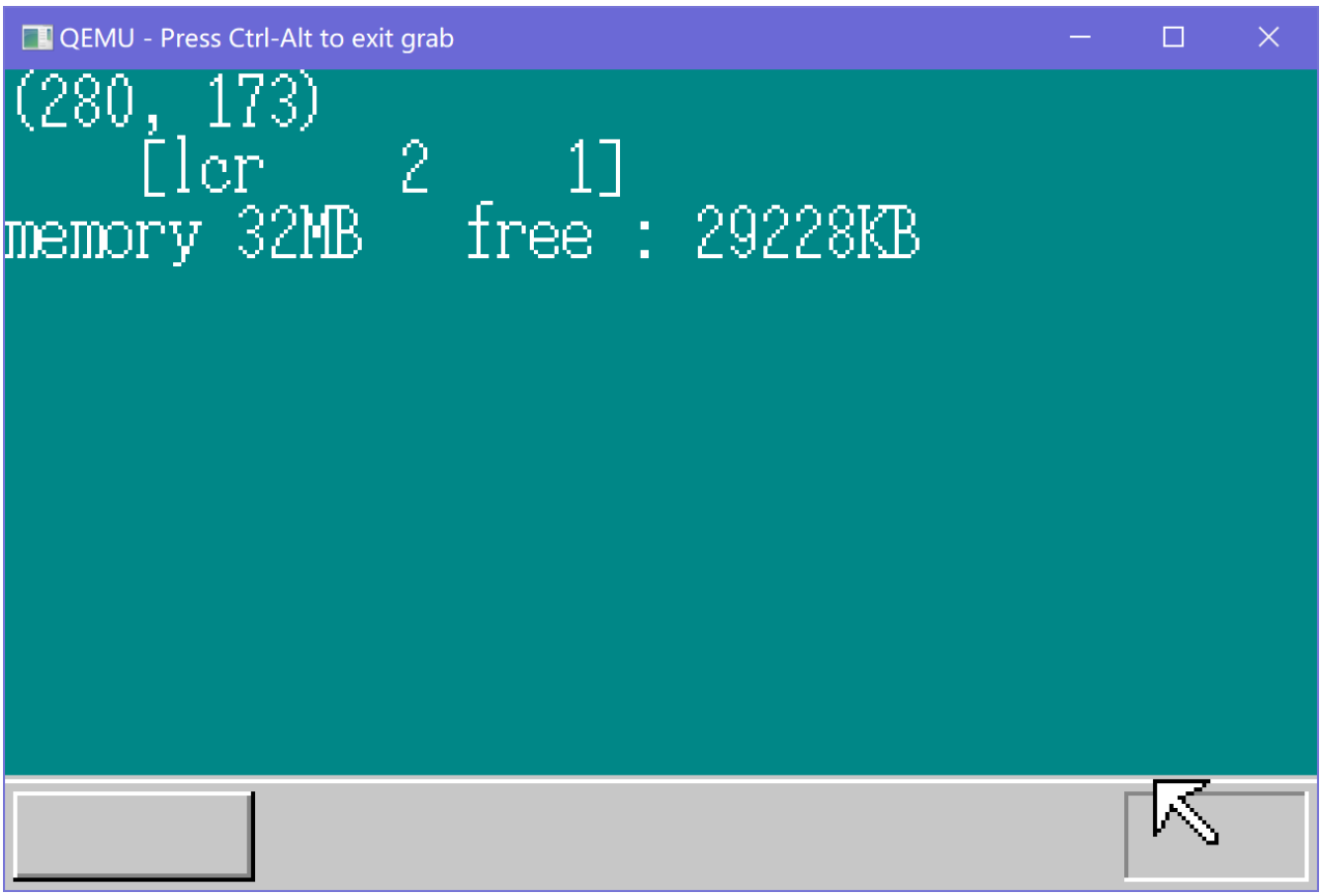
一个这样的sheet结构体共8个四字节变量（32位机指针是4字节）。

我们只准备256个这样的sheet，因为这个看起来够用了，毕竟是实验性的系统，哪怕是使用win10，也很少有人会同时开256个窗口。

我们需要像管理内存一样管理这256个sheets，所以也需要编制相应的分配及回收程序。

我们还需要编制相应的修改height的代码，以及绘制整个屏幕的函数。

完成相应的编制之后，我们并不需要修改putfont putblock等函数，因为作者之前留下vram参数大概是早有预谋，是为了避免添加叠加处理功能的时候修改更多的代码吧。



看起来一切工作正常

SubPhase 2

其实书上提到的屏幕闪烁问题，我并没有遇到。想必大家也没有遇到，估计是因为作者在写这本书的时候计算机机能并不好，而现在的机能相比以前提升很大，屏幕不闪烁也是意料之中了。

不过屏幕刷新还是有优化的地方的，我们每次鼠标移动，都会重绘整个屏幕，这是十分效率低下的。实际上，我们只需要重绘鼠标那16x16个像素就好了。

同样的，文字打印也会重绘整个屏幕，这里也需要优化的。所以我们重新编制一下绘图函数，令他只重绘屏幕的一个矩形部分，而不是整个屏幕范围。

另外作者终于打算在这里处理打印部分超出现有图层/屏幕外部的情况了（笑），我们早在之前刚开始做字符串打印的时候就考虑到了这种情况。

不过他还是暂时还不打算让鼠标的右下部份超出屏幕。

```
void sheet_refreshsub(struct SHTCTL *ctl, int vx0, int vy0, int vx1, int vy1) {
    int h, bx, by, vx, vy, bx0, by0, bx1, by1;
    unsigned char *buf, c, *vram = ctl->vram;
    struct SHEET *sht;
    for (h = 0; h <= ctl->top; h++) {
        sht = ctl->sheets[h];
        buf = sht->buf;
        bx0 = vx0 - sht->vx0;
        by0 = vy0 - sht->vy0;
        bx1 = vx1 - sht->vx0;
        by1 = vy1 - sht->vy0;
        if (bx0 < 0)
            bx0 = 0;
        if (by0 < 0)
            by0 = 0;
        if (bx1 > sht->bxsize)
            bx1 = sht->bxsize;
        if (by1 > sht->bysize)
            by1 = sht->bysize;
        for (by = by0; by < by1; by++) {
            vy = sht->vy0 + by;
            for (bx = bx0; bx < bx1; bx++) {
                vx = sht->vx0 + bx;
                c = buf[by * sht->bxsize + bx];
                if (c != sht->col_inv) {
                    vram[vy * ctl->xsize + vx] = c;
                }
            }
        }
    }
    return;
}
```