

# Day 26

今天的任务真不少，我们要做的事情有

- 提高窗口移动速度
- 启动时只打开一个命令行窗口
- 增加更多的命令行窗口
- 实现命令行窗口的关闭
- 实现start命令
- 实现ncst命令

提高窗口的移动速度在于提高窗体的绘图速度并减少不必要的绘图。

通过分析sheet\_refreshmap我们发现每次改变像素点颜色之前都会有个if判断该点是否是透明的。在某些图层当中，例如窗体，我们并没有透明部分，所以这个检查是多余的，我们可以在最外层添加一个判断，判断正在绘图的这个窗口是否含有透明部分，如果没有透明的部分就进入没有判断的循环，反之进入有判断的循环。

```
if (sht->col_inv == -1) {
    /*无透明色*/
    for (by = by0; by < by1; by++) {
        vy = sht->vy0 + by;
        for (bx = bx0; bx < bx1; bx++) {
            vx = sht->vx0 + bx;
            map[vy * ctl->xsize + vx] = sid;
        }
    }
} else {
    /*有透明色*/
    for (by = by0; by < by1; by++) {
        vy = sht->vy0 + by;
        for (bx = bx0; bx < bx1; bx++) {
            vx = sht->vx0 + bx;
            if (buf[by * sht->bxsize + bx] != sht->col_inv) {
                map[vy * ctl->xsize + vx] = sid;
            }
        }
    }
}
```

注意到我们是将内存中连续区域复制到显存的多组连续区域当中，我们目前每次复制一字节，如果我们一次能赋值更多字节就好了。

作者通过指针转换的方式实现了一次复制4字节。但这弄得有点小麻烦，其实memcpy函数已经是能够按照机器字长进行拷贝的，我们直接用memcpy应该也可以提高速度的。（作者这种写法反而使得窗体只能移动到4的倍数的位置上了）

还有一些其他的trick可以使用，但作者并没有使用。例如**循环展开**：循环展开可以充分利用CPU的多个功能单元，使得指令能够并行执行从而得到常数级别的指令运行速度提升。

我们把这些trick也应用到sheet\_refreshsub上

```
for (i = 0; i < i1; i++) { /* 4的倍数部分*/
    if (p[i] == sid4) {
        q[i] = r[i];
    } else {
        bx2 = bx + i * 4;
        vx = sht->vx0 + bx2;
        if (map[vy * ctl->xsize + vx + 0] == sid) {
            vram[vy * ctl->xsize + vx + 0] = buf[by * sht->bysize + bx2 + 0];
        }
        if (map[vy * ctl->xsize + vx + 1] == sid) {
            vram[vy * ctl->xsize + vx + 1] = buf[by * sht->bysize + bx2 + 1];
        }
        if (map[vy * ctl->xsize + vx + 2] == sid) {
            vram[vy * ctl->xsize + vx + 2] = buf[by * sht->bysize + bx2 + 2];
        }
        if (map[vy * ctl->xsize + vx + 3] == sid) {
            vram[vy * ctl->xsize + vx + 3] = buf[by * sht->bysize + bx2 + 3];
        }
    }
}
for (bx += i1 * 4; bx < bx1; bx++) { /*剩余部分逐字节写入*/
    vx = sht->vx0 + bx;
    if (map[vy * ctl->xsize + vx] == sid) {
        vram[vy * ctl->xsize + vx] = buf[by * sht->bysize + bx];
    }
}
```

得益于计算机性能的提升，即使实在模拟器当中运行，目前为止的窗体移动速度已经足够快了

但是还是有优化的空间的。

当FIFO中有多个数据时进行绘图是不明智的，因为我们绘制完的窗体在FIFO中的数据得到处理之后马上又会变更了，这个FIFO处理的很快，我们绘制完的窗体用户还没怎么看见马上就又需要重绘了，所以我们决定等待FIFO为空之后再进行绘图。

```
if (fifo32_status(&fifo) == 0) {
    /* FIFO为空，当存在搁置的绘图操作时立即执行*/ /*从此开始*/
    if (new_mx >= 0) {
        io_sti();
        sheet_slide(sht_mouse, new_mx, new_my);
        new_mx = -1;
    } else if (new_wx != 0x7fffffff) {
        io_sti();
        sheet_slide(sht, new_wx, new_wy);
        new_wx = 0x7fffffff;
    } else {
        task_sleep(task_a);
        io_sti();
    } /*到此结束*/
} else {
    // .....
}
```

```

} else {
    /*没有按下鼠标左键*/
    mmx = -1; /*切换到一般模式*/
    if (new_wx != 0x7fffffff) { /*从此开始*/
        sheet_slide(sht, new_wx, new_wy); /*固定图层位置*/
        new_wx = 0x7fffffff;
    } /*到此结束*/
}

```

我们把鼠标移动的位置暂时的保存下来，而不真的移动鼠标的位置（new\_mx new\_my）

窗体同理，不过我们用来标记暂时不需要绘图的值用的是0x7fffffff而不是-1，这是因为窗体坐标是有可能为-1的。

当鼠标左键弹起推出窗口移动模式的时候，我们也要立即更新窗口的位置，这是因为用户可能接着回去操作移动别的窗口，sht会发生改变，导致我们之后移动的是错误的窗体。

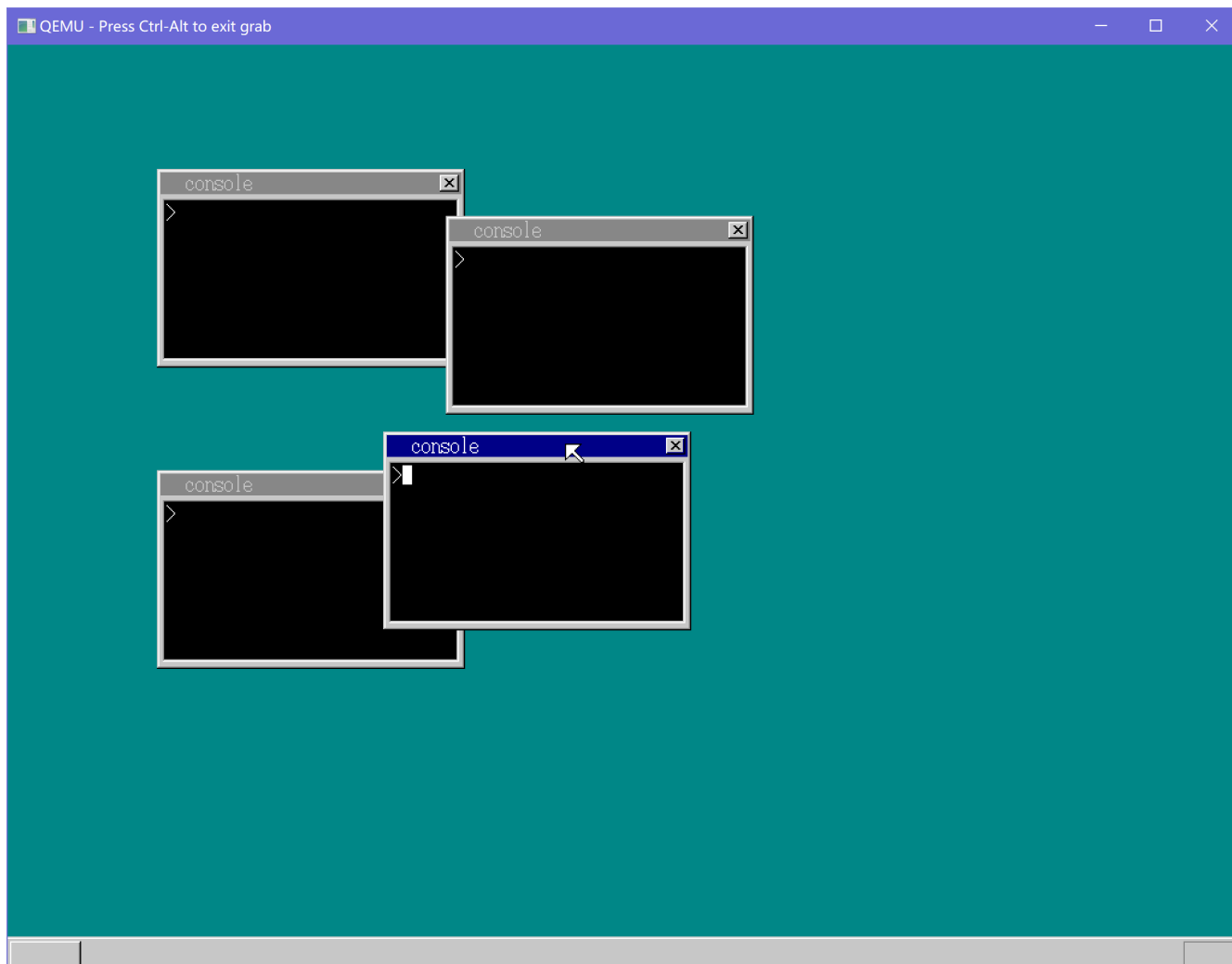
（其实做了这个优化之后直观上感觉速度和之前没有什么提升）

系统启动的时候自动打开两个命令行窗口很奇怪，一般都是先开一个，然后用户根据需求打开更多的窗口。我们先把命令行窗口的数量减少到一个，然后设定一个打开命令行窗口的快捷键 Shift + F2

```

if (i == 256 + 0x3c && key_shift != 0) { /* Shift+F2 */
    keywin_off(key_win);
    key_win = open_console(shtctl, memtotal);
    sheet_slide(key_win, 32, 4);
    sheet_updown(key_win, shtctl->top);
    keywin_on(key_win);
}

```



既然已经实现了打开多个命令行窗口，我们也应该实现命令行窗口的关闭功能。具体思路是回退打开命令行窗口所造成的影响。

- 释放内存空间
- 回收图层
- 回收task
- 回收栈

为了能够回收栈，我们把栈地址记录在task当中

把必要信息在open\_console中记录在task中

```
task->cons_stack = memman_alloc_4k(memman, 64 * 1024);
task->tss.esp = task->cons_stack + 64 * 1024 - 12;
```

然后我们的思路就比较清晰了。

```
void close_constask(struct TASK *task)
{
    struct MEMMAN *memman = (struct MEMMAN *) MEMMAN_ADDR;
    task_sleep(task);
    memman_free_4k(memman, task->cons_stack, 64 * 1024);
    memman_free_4k(memman, (int) task->fifo.buf, 128 * 4);
}
```

```

    task->flags = 0;
    return;
}
void close_console(struct SHEET *sht)
{
    struct MEMMAN *memman = (struct MEMMAN *) MEMMAN_ADDR;
    struct TASK *task = sht->task;
    memman_free_4k(memman, (int) sht->buf, 256 * 165);
    sheet_free(sht);
    close_constask(task);
    return;
}

```

先释放内存，然后释放图层，最后关闭任务。关闭任务的部分比较有趣：我们先让任务休眠，这是为了将任务从ready队列中安全地一处出来，这样就不会再调度到这个任务了。然后我们才可以安全地释放栈和FIFO缓冲区。当内存释放完毕之后，我们给task的flags标记为0，以便下次分配使用。

然后我们要做的就是将exit命令和我们上面写的函数进行绑定。

有一个比较头疼的问题，就是如果我们的console自己调用了close\_console的话，那么它sleep了自己之后，接下来的代码都无法执行了，所以我们让另一个task来进行这个操作。我们之前只是取消了task\_a的窗口，但是task\_a本身还在，所以我们让他来负责close\_console就好了。

通过命令行窗口任务向task\_a的FIFO发送数据，然后task\_a读到数据并关闭对应的console。

console发送完数据之后就可以休眠了。

```

void cmd_exit(struct CONSOLE *cons, int *fat)
{
    struct MEMMAN *memman = (struct MEMMAN *) MEMMAN_ADDR;
    struct TASK *task = task_now();
    struct SHTCTL *shtctl = (struct SHTCTL *) *((int *) 0x0fe4);
    struct FIFO32 *fifo = (struct FIFO32 *) *((int *) 0x0fec);
    timer_cancel(cons->timer);
    memman_free_4k(memman, (int) fat, 4 * 2880);
    io_cli();
    fifo32_put(fifo, cons->sht - shtctl->sheets0 + 768);
    io_sti();
    for (;;) {
        task_sleep(task);
    }
}

```

```

/*-----HariMain-----*/
else if (768 <= i && i <= 1023) {
    close_console(shtctl->sheets0 + (i - 768));
}


```

如果我们的所有窗口都被关闭了，是会出问题的，这里我们需要处理一下

```

if (key_win != 0 && key_win->flags == 0) {
    if (shtctl->top == 1) { /*只剩鼠标和背景*/
        key_win = 0;
    } else {
        key_win = shtctl->sheets[shtctl->top - 1];
        keywin_on(key_win);
    }
}

```

然后鼠标点击  的部分我们也进行修改，以便我们能够通过鼠标来关闭命令行窗口

```

if ((sht->flags & 0x10) != 0) { /*是否为应用程序
窗口? */
    //.....
} else { /*命令行窗口*/
    task = sht->task;
    io_cli();
    fifo32_put(&task->fifo, 4);
    io_sti();
}

```

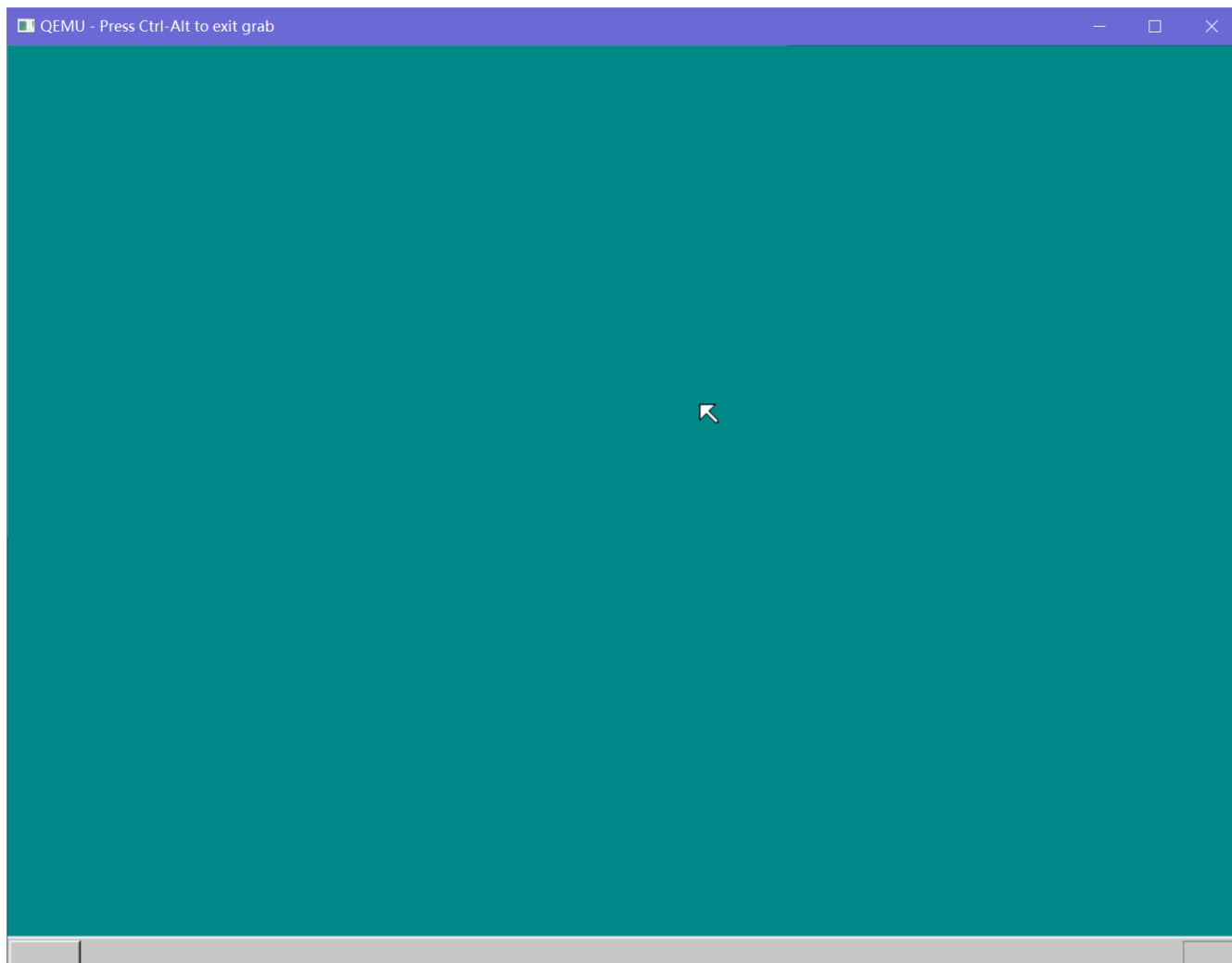
consoletask中对队列数据处理部分要稍微改一下

```

if (fifo32_status(&task->fifo) == 0) {
    //.....
} else {
    //.....
    if (i == 4) {
        cmd_exit(&cons, fat);
    }
    //.....
}

```

测试一下



工作正常

实现start命令，start命令的功能是打开一个新的命令行窗口并运行指定的程序

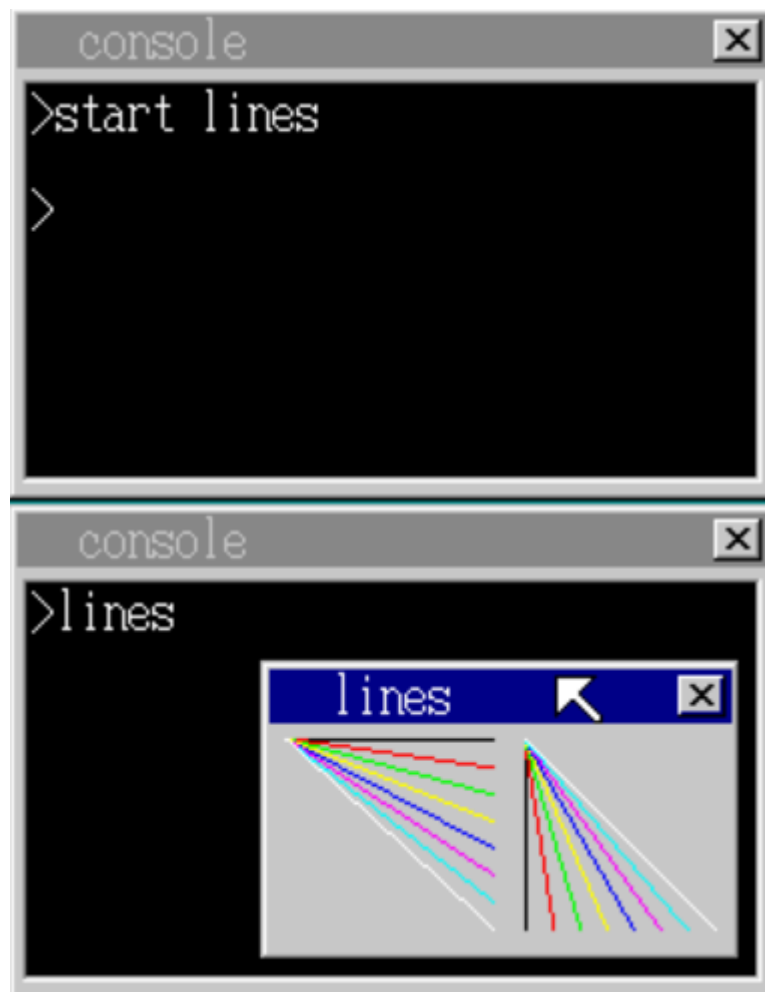
```
} else if (strncmp(cmdline, "start ", 6) == 0) {  
    cmd_start(cons, cmdline, memtotal);  
}
```

```
void cmd_start(struct CONSOLE *cons, char *cmdline, int memtotal)  
{  
    struct SHTCTL *shtctl = (struct SHTCTL *) *((int *) 0x0fe4);  
    struct SHEET *sht = open_console(shtctl, memtotal);  
    struct FIFO32 *fifo = &sht->task->fifo;  
    int i;  
    sheet_slide(sht, 32, 4);  
    sheet_updown(sht, shtctl->top);  
    for (i = 6; cmdline[i] != 0; i++) {  
        fifo32_put(fifo, cmdline[i] + 256);  
    }  
    fifo32_put(fifo, 10 + 256); /*回车键*/  
    cons_newline(cons);  
}
```

```
    return;  
}
```

思路就是新建一个命令行窗口，然后把命令按键逐个发送过去。

试一下吧



成功

今天最后的任务是ncst命令。ncst是no console start的缩写，意思是不打开命令行窗口的启动。

思路是禁止向命令行窗口显示内容（避免出现错误）

将命令行任务的 `cons->sht` 规定为0。由于没有窗口，所以命令行窗口的内建命令我们可以直接忽略。



```

/*-----cons_runcmd-----*/
if (strcmp(cmdline, "mem") == 0 && cons->sht != 0) {
    cmd_mem(cons, memtotal);
} else if (strcmp(cmdline, "cls") == 0 && cons->sht != 0) {
    cmd_cls(cons);
} else if (strcmp(cmdline, "dir") == 0 && cons->sht != 0) {
    cmd_dir(cons);
} else if (strncmp(cmdline, "type ", 5) == 0 && cons->sht != 0) {
    cmd_type(cons, fat, cmdline);
}

```

```

void cmd_ncst(struct CONSOLE *cons, char *cmdline, int memtotal)
{
    struct TASK *task = open_constask(0, memtotal);
    struct FIFO32 *fifo = &task->fifo;
    int i;
    for (i = 5; cmdline[i] != 0; i++) {
        fifo32_put(fifo, cmdline[i] + 256);
    }
    fifo32_put(fifo, 10 + 256);
    cons_newline(cons);
    return;
}

```

充分利用已经做好的功能，我们仍然使用和start相似的做法

然后我们在cons\_putchar和newline中也进行修改，也加上类似的判断。然后屏蔽输出这部分我们就完成了。

然后就是程序结束后的自动退出

```

if (sheet == 0) {
    cmd_exit(&cons, fat);
} /*这一句应该放在cons_runcmd之后*/

```

cmd\_exit也需要修改，有命令行窗口时，我们可以通过图层的地址告诉task\_a需要结束哪个任务，现在我们需要用FIFO的方法来告诉task\_a

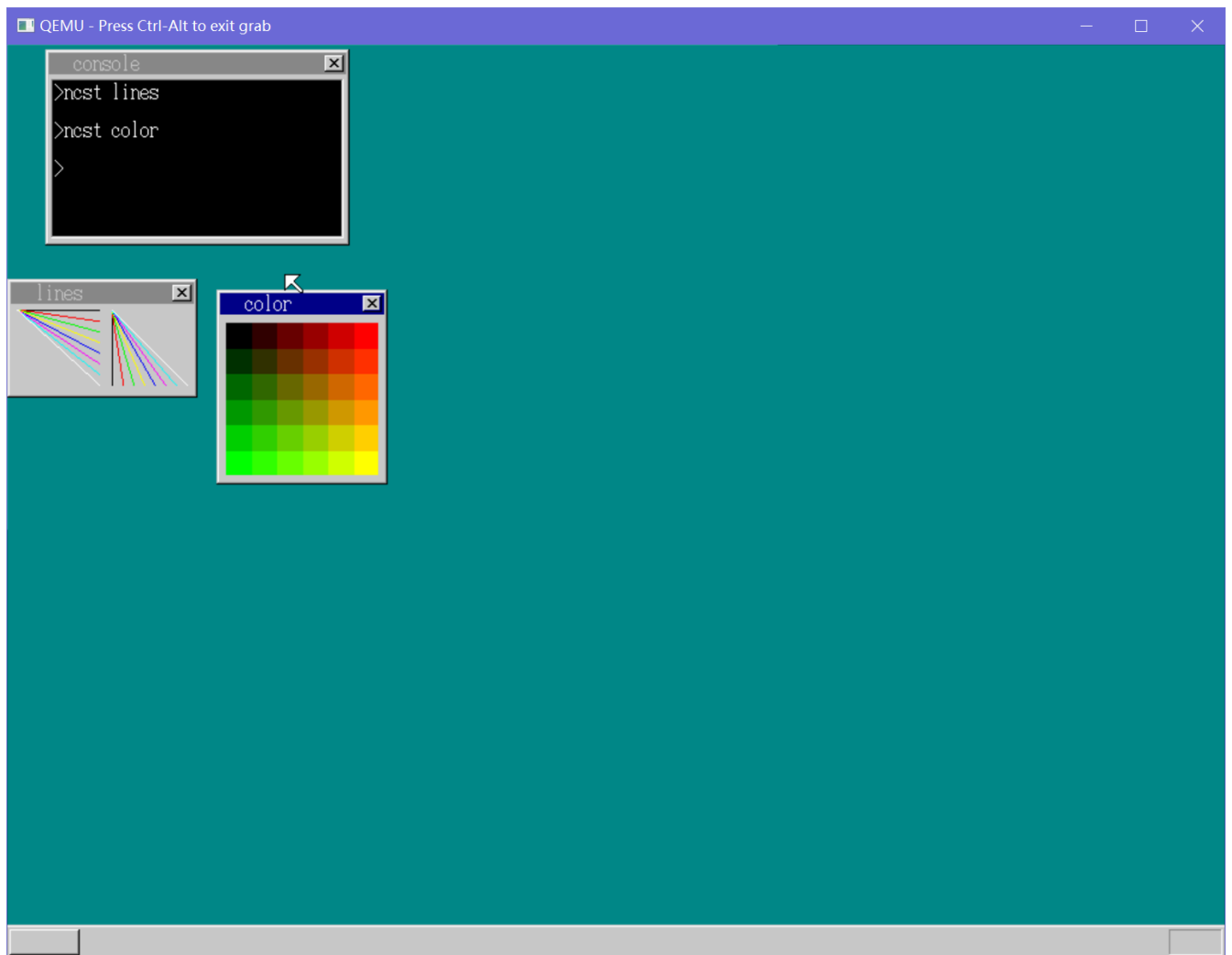
harimain中也需要些小修改

```

} else if (768 <= i && i <= 1023) { /*命令行窗口关闭处理*/
    close_console(shtctl->sheets0 + (i - 768));
} else if (1024 <= i && i <= 2023) {
    close_constask(taskctl->tasks0 + (i - 1024));
}

```

测试一下



点击关闭却出现了问题，应用程序窗口无法被关闭。