重 庆 大 学

学 生 实 验 报 告

实验课程名称 操作系统原理

开课实验室 重庆大学 DS1501

学 院 软件工程 年级 2013 专业班

学 生 姓 名 学 号

开 课 时 间 2015 至 2016 学年第 一 学期

总 成 绩	
教师签名	洪明坚

重庆大学软件学院制

..

《操作系统原理》实验报告

开课实验室：DS1501　　2016 年 1 月 7 日

学院	软件学院	年级、专业、班		姓名		成绩	
课程名称	操作系统原理	实验项目名　　称	线程同步		指导教师		
教师评语	教师签名： 年　　月　　日						

- 实验目的（软件需求文档）

掌握信号量的实现与应用

二、实验原理（软件设计文档）

互斥的实现

- 一般不直接用 sti/cli，而是用
  - save_flags_cli(flags)

保存 EFLAGS 的值到一个变量 flags 中，然后 IF=0

  - restore_flags(flags)

把变量 flags 的值恢复到 EFLAGS 中

线程的睡眠和唤醒

- 睡眠
  - void sleep_on(struct wait_queue **head)
    - 参数 head 是睡眠队列的头指针的指针
- 唤醒
  - void wake_up(struct wait_queue **head, int n)
    - 参数 n 表示要唤醒的线程个数
      - n 小于 0 表示唤醒该队列中的所有线程
- sleep_on 和 wake_up 必须在关中断环境中运行
  - 用 save_flags_cli/restore_flags 保护

实现信号量

编辑文件 epos/sem.c，实现如下四个函数

void *sys_sem_create(int value)

value 是信号量的初值

.

分配内存要用 kmalloc，不能用 malloc!

成功返回信号量句柄 hsem，否则返回 NULL

int sys_sem_destroy(void *hsem)

释放内存要用 kfree，不能用 free!

成功返回 0，否则返回-1

int sys_sem_wait(void *hsem)

P 操作，要用 save_flags_cli/restore_flags 和函数 sleep_on

成功返回 0，否则返回-1

int sys_sem_signal(void *hsem)

V 操作，要用 save_flags_cli/restore_flags 和函数 wake_up

成功返回 0，否则返回-1

把这四个函数做成系统调用，分别是 sem_create/destroy/wait/signal

解决实验（二）中的花屏现象

在 graphics.c 中，

定义一个表示信号量的全局变量

```
static void *hsem = NULL;
```

分别在函数 initGraphics 和 exitGraphics 中创建和销毁信号量 hsem

在函数 setPixel 中，用信号量 hsem 保护下面的临界区

```
switchBank(HIWORD(addr));
 *p      = getBValue(cr);
 *(p+1)  = getGValue(cr);
 *(p+2)  = getRValue(cr);
```

实现生产者/消费者

生产者生成随机数后，要画到缓冲区

消费者排序后，要清空缓冲区

.

三、使用仪器、材料（软硬件开发环境）

Notepad++

expenv

四、实验步骤（实现的过程）

```
//定义信号量结构体
struct semaphore{
    int value;
    /*processes blocked by this semaphore*/
    struct wait_queue *wq_kbd;
};
```

sem.c:

```
void *sys_sem_create(int value)
{
    struct semaphore *hsem;
    char *p;
    p = (char *)kmalloc(sizeof(struct semaphore));
    hsem = (struct semaphore *)p;
    hsem->value=value;//赋初值
    hsem->wq_kbd = NULL;//!!!非常重要!!!

    if(hsem==NULL)    return NULL;
    else              return (struct semaphore *)hsem;//成功返回信号量句柄hsem
}

int sys_sem_destroy(void *hsem)
{
    kfree(hsem);
    return 0;
}

int sys_sem_wait(void *hsem)
{
    uint32_t flags;
    save_flags_cli(flags);
    ((struct semaphore*)hsem)->value--;//P操作
    if(((struct semaphore*)hsem)->value < 0) {
        sleep_on(&(((struct semaphore*)hsem)->wq_kbd));
    }
    restore_flags(flags);
    return 0;
}

int sys_sem_signal(void *hsem)
```

.

..

```
{
    uint32_t flags;
    save_flags_cli(flags);
    ((struct semaphore*)hsem)->value++;//V操作
    if(((struct semaphore*)hsem)->value<=0){
        wake_up(&(((struct semaphore*)hsem)->wq_kbd), 1);//等待队列中唤醒一个线程
    }
    restore_flags(flags);
    return 0;
}
```

把这四个函数做成系统调用 sem_create/destroy/wait/signal

graphics.c 中

```
//定义一个表示信号量的全局变量
static void *hsem = NULL;
hsem=sem_create(1);//创建信号量
sem_destroy(hsem);//销毁信号量
```

以及

```
sem_wait(hsem);//用信号量hsem保护画点临界区
    switchBank(HIWORD(addr));
    *p      = getBValue(cr);
    *(p+1) = getGValue(cr);
    *(p+2) = getRValue(cr);
     sem_signal(hsem);
```

main.c：

声明信号量：

```
# define arrayNumber 50
#define N  4  //缓冲区数

static void* mutex = NULL;    //互斥信号量
static void* full = NULL;    //已生产产品数
static void* empty = NULL;     //空闲数
int startposition = 0;  //生产者缓冲区坐标
static int count = 0; //消费者缓冲区坐标
int temp[4][50] = { 0 };
int array[N][arrayNumber];
```

.

生产者消费者函数：

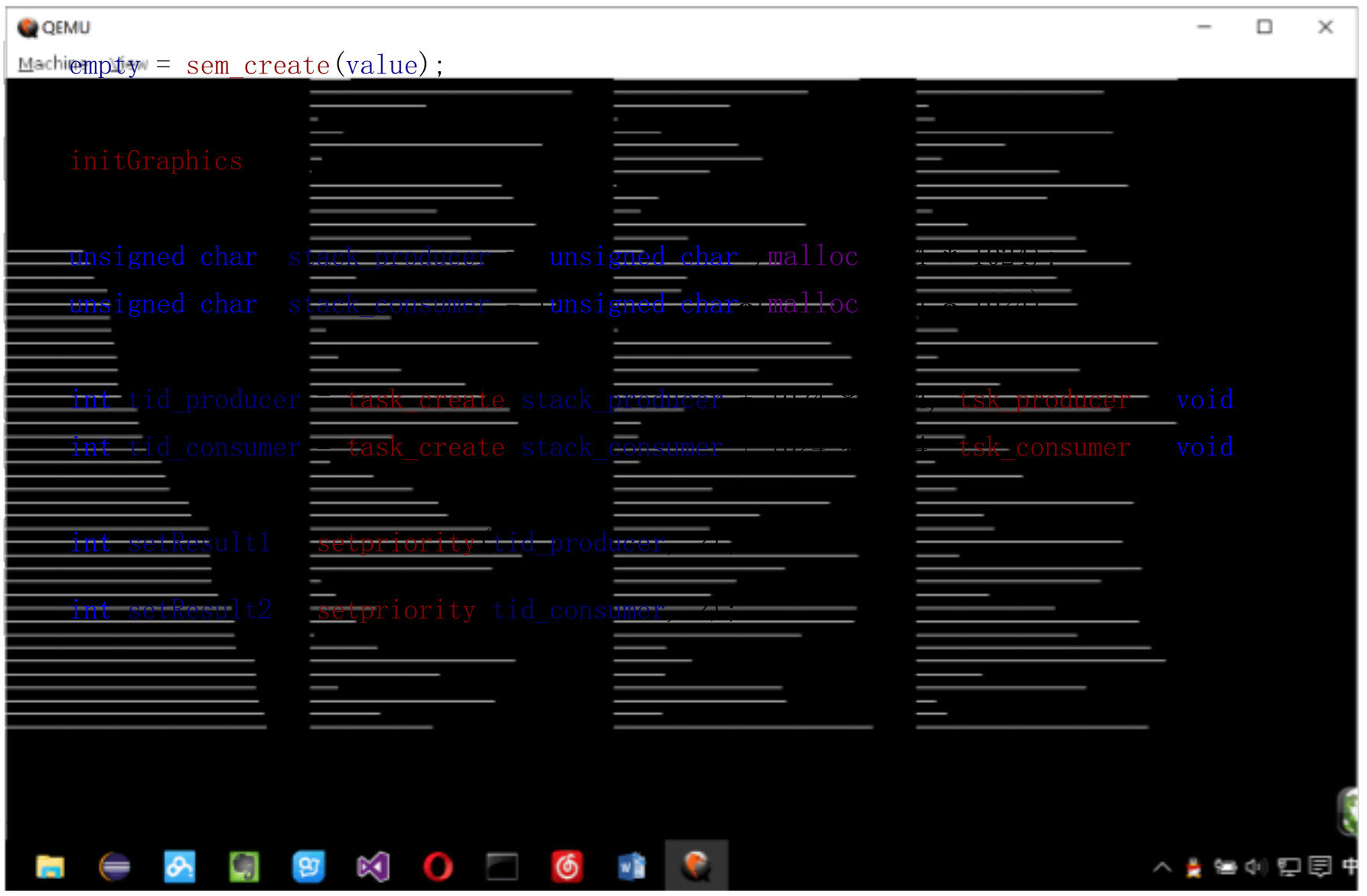
```
void tsk_producer(void *pv){

    while (1)
    {
        int i = 0;
        srand((unsigned)time(NULL));
        sem_wait(empty);//是否有空闲位置？
        sem_wait(mutex);
        for (i = 0; i <= arrayNumber - 1; i++)
        {
            array[startposition][i] = random() % 200;
            draw(i * 10, startposition * 230, startposition * 230 + array[startposition][i]);
            DELAY(100000);
        }
        startposition = (++startposition) % N;
        sem_signal(mutex);
        sem_signal(full);//有产品生产好
    }
    task_exit(0);
}

void tsk_consumer(void *pv){

    while (1)
    {
        sem_wait(full);//还有产品么？
        sem_wait(mutex);
        int i;
        for (i = 0; i <= arrayNumber - 1; i++)
        {
            temp[count][i] = array[count][i];
        }
        sem_signal(mutex);//让出临界区使生产与消费并行
        sort_m(temp[count], arrayNumber, count * 230); //冒泡排序
        resetAllBK(temp[count], arrayNumber, count * 230); //清空该缓冲区
        count=(++count)%N;
        sem_signal(empty);//消费完一个产品
    }
    task_exit(0);
}
```

.

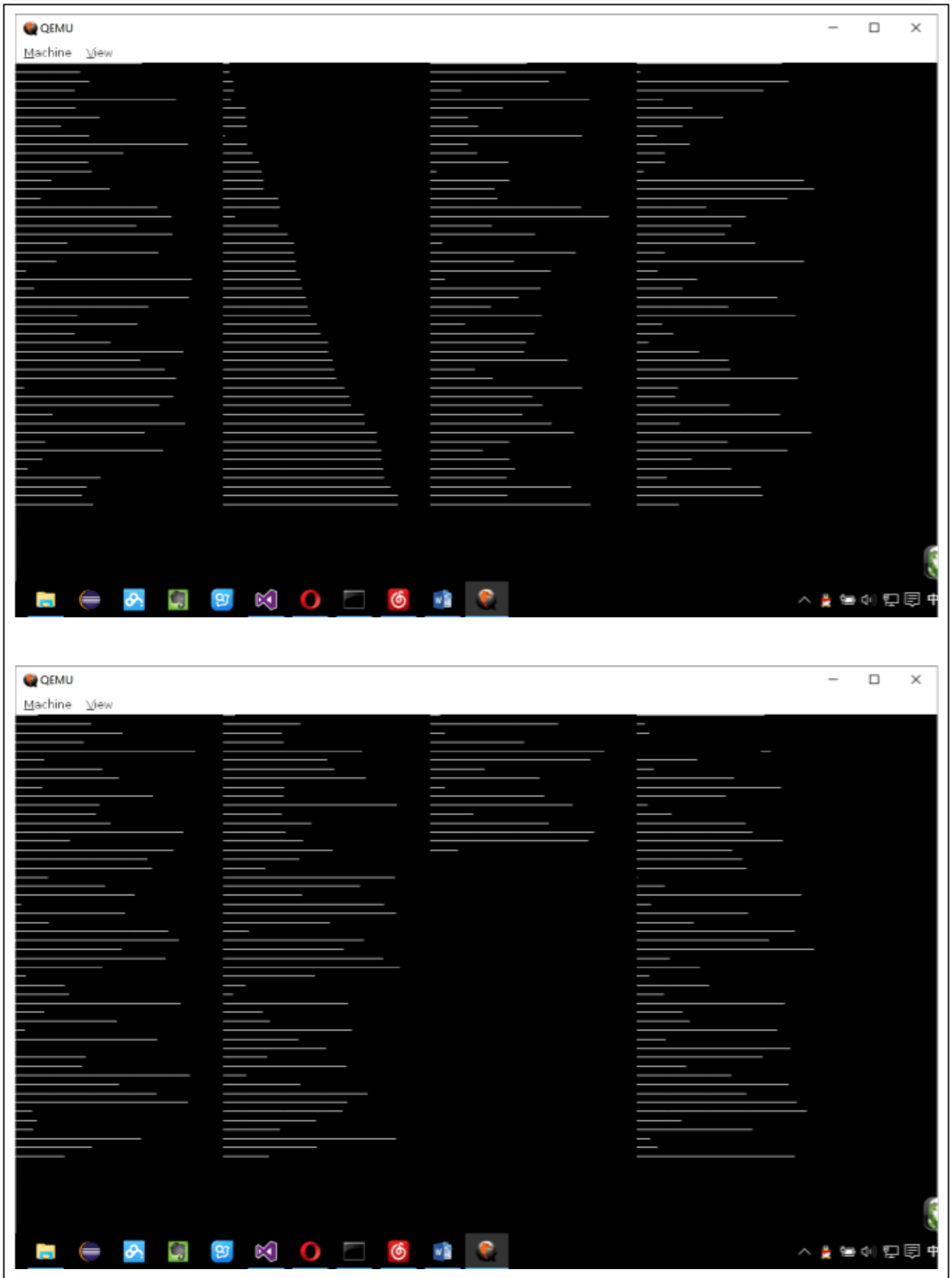
主函数：

```
    int value = 1;
    mutex = sem_create(value);

    value = 0;
    full = sem_create(value);

    value = N;
    empty = sem_create(value);

    initGraphics(0x0118);

    unsigned char *stack_producer = (unsigned char*)malloc(1024 * 1024);
    unsigned char *stack_consumer = (unsigned char*)malloc(1024 * 1024);

    int tid_producer = task_create(stack_producer + 1024 * 1024, tsk_producer, (void *)0);
    int tid_consumer = task_create(stack_consumer + 1024 * 1024, tsk_consumer, (void *)0);

    int setResult1 = setpriority(tid_producer, 2);
    int setResult2 = setpriority(tid_consumer, 2);
```

五、实验结果及分析（实现的效果，包括屏幕截图、系统总体运行情况和测试情况等）

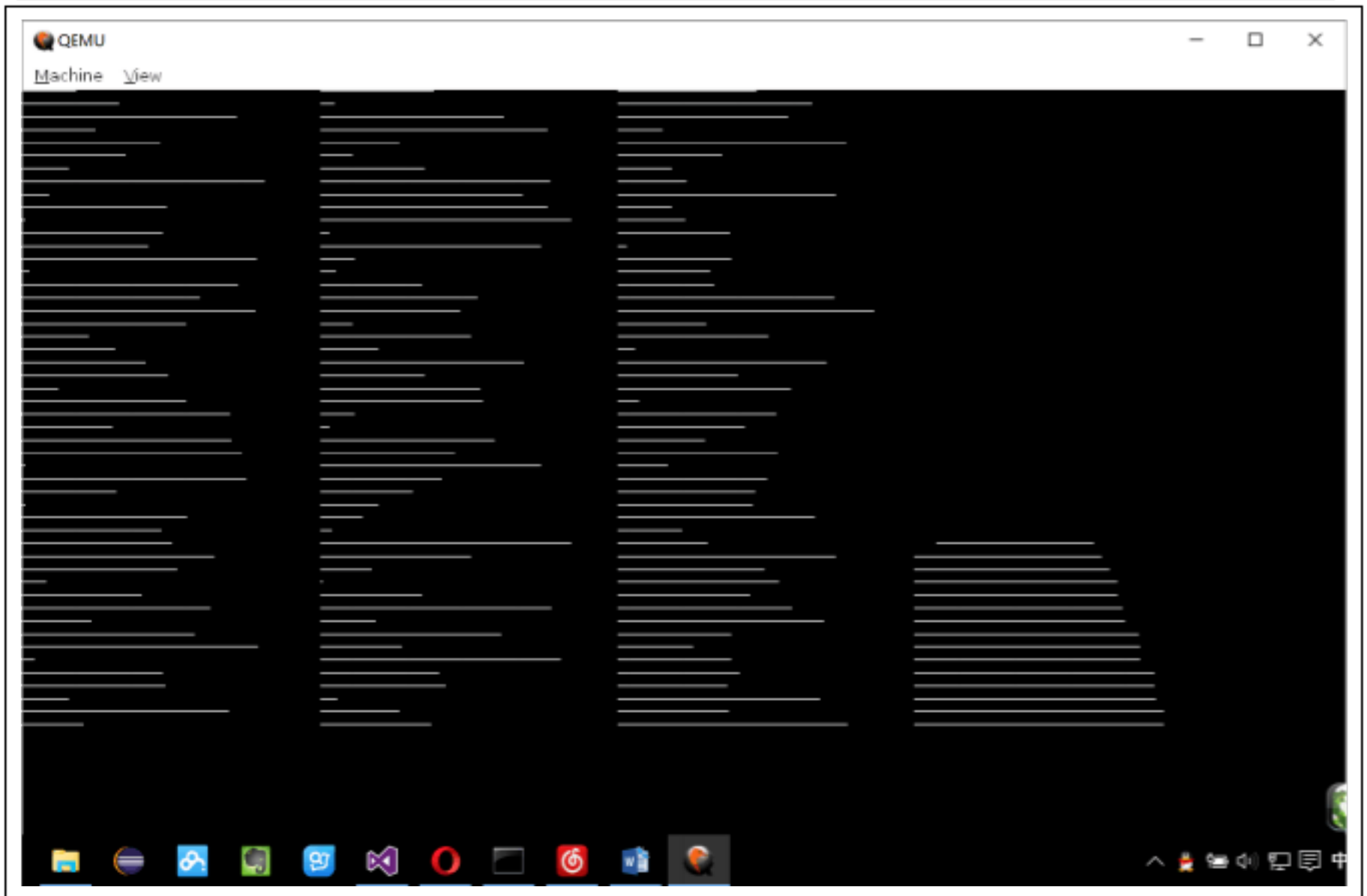


..


QEMU
Machine View
QEMU
Machine View

.

..



通过互斥量保护 Setpixel 解决了实验二优先级调度的花屏现象

通过做本实验，我对书本上信号量概念有了实践性的理解

.