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 6 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	线程及其调度		指导教师		
教师评语	教师签名： 年　　月　　日						

一、实验目的（软件需求文档）

掌握线程的创建

掌握线程的调度

a) 静态优先级调度

b) 动态优先级调度

二、实验原理（软件设计文档）

系统调用接口，线程相关函数：

- Step1：定义线程函数

```
void tsk_foo(void *pv)
{
    printf("This is task foo with tid=%d\r\n“, task_getid());
    task_exit(0);
}
```

- Step2：申请用户栈

```
unsigned char *stack_foo;
stack_foo = (unsigned char *)malloc(1024*1024);
```

  - 线程退出后，才能把用户栈用 free 释放掉！

- Step3：创建线程

```
int tid_foo;
tid_foo = task_create(stack_foo+1024*1024, tsk_foo, (void *)0);
```

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）

随机生成 3 组非负整数列表，创建 3 个线程，分别用 3 种不同的排序算法（插入，冒泡，选择）对列表进行排序

三线程：

```
void tsk_foo_line1(void *pv)
{
    int m;
    int i;
    int arry[50];
    srand(time(NULL));
    for (i = 0; i<50; i++){
        m = random() % 200;
        if (m<0) { m = 0 - m; }
        draw(i * 10, 0, 0 + m);
        arry[i] = m;
    }
    sort_m(arry, 50, 0);
    task_exit(0);
}

void tsk_foo_line2(void *pv)
{
    int m;
    int i;
    int arry[50];
    srand(time(NULL));
    for (i = 0; i<50; i++){
        m = random() % 200;
        if (m<0) { m = 0 - m; }
        draw(i * 10, 345, 345 + m);
        arry[i] = m;
    }
    sort_x(arry, 50, 345);
    task_exit(0);
}


void tsk_foo_line3(void *pv)
{
    int m;
```

```
    int i;
    int arry[50];
    srand(time(NULL));
    for (i = 0; i<50; i++){
        m = random() % 200;
        if (m<0) { m = 0 - m; }
        draw(i * 10, 690, 690 + m);
        arry[i] = m;
    }
    sort_c(arry, 50, 690);
    task_exit(0);
}
void draw(int x, int y1, int y2){
    int i;
    for (i = y1; i<y2; i++)
        setPixel(i, x, RGB(255, 255, 255));
}

void resetBK(int x, int y1, int y2){
    int i;
    for (i = y1; i<y2; i++)
        setPixel(i, x, RGB(0, 0, 0));
}
```

三排序：

冒泡

```
void sort_m(int* arry, int n, int l){
    int i, j, tem;
    int t = 500 / n;

    for (i = 0; i<n; i++)
    for (j = 0; j<n - i - 1; j++){

        if (*(arry + j)>*(arry + j + 1)){
            resetBK(j*t, l, l + *(arry + j));
            resetBK(j*t + t, l, l + *(arry + j + 1));
            tem = *(arry + j);
            *(arry + j) = *(arry + j + 1);
            *(arry + j + 1) = tem;
            draw(j*t, l, l + *(arry + j));
            draw(j*t + t, l, l + *(arry + j + 1));
        }

    }
}
```

插入

```
void sort_c(int* arry, int n, int l){
    int i, j, key;
    int t = 500 / n;
    for (j = n - 2; j >= 0; j--){
        key = *(arry + j); i = j + 1;
        resetBK(j*t, l, l + key);
        while (i<n && *(arry + i)<key){
            *(arry + i - 1) = *(arry + i);
            draw(i*t - t, l, l + *(arry + i - 1));
            i = i + 1;
        }
        *(arry + i - 1) = key;
        draw(i*t - t, l, l + key);


    }
}
```

选择

```
void sort_x(int* arry, int n, int l){
    int i=0, j=0, lowindex = 0;
    int t = 500 / n;
    for (i = 0; i<n; i++){
     lowindex = i;
        for (j = n - 1; j>i; j--)
        if (arry[j] < arry[lowindex])
            lowindex = j;
        if (lowindex != i)
        {
            resetBK(i*t, l, l + *(arry + i));
            resetBK(lowindex*t, l, l + *(arry + lowindex));
            int temp = arry[i];
            arry[i] = arry[lowindex];
            arry[lowindex] = temp;
            draw(i*t, l, l + *(arry + i));
            draw(lowindex*t, l, l + *(arry + lowindex));
        }
    }
}
```

线程控制块 tcb 中增加 nice 属性，在函数 sys_task_create 中初始化 nice=0

```
/*
系统调用getpriority的执行函数
获取当前线程的优先级
*/


int sys_getpriority(int tid)
```

```
{
    if(tid==0) return g_task_running->nice+NZERO;//获取当前线程的nice值
    uint32_t flags;  struct tcb *tsk;
    save_flags_cli(flags);
    tsk = get_task(tid);
    restore_flags(flags);

    return tsk->nice+NZERO;             //获取线程的nice值
}

/*
系统调用setpriority的执行函数
调整当前线程的优先级
*/
//把线程tid的nice设为(prio-NZERO)
int sys_setpriority(int tid, int prio)
{
    uint32_t flags;  struct tcb *tsk;
    if(tid==0) {
        save_flags_cli(flags);
        g_task_running->nice=prio-20;//设置当前线程的nice值
        restore_flags(flags);
        return 0;
    }
    //if(tsk==NULL) return -1;
    if (prio<0) prio = 0;               //prio必须在[0,2*NZERO-1]
    if(prio>40) prio=40;
    save_flags_cli(flags);
    tsk = get_task(tid);                //用save_flags_cli/restore_flags保护
    restore_flags(flags);

    tsk->nice=prio-20;//设置线程的nice值
    return 0;
}
```

把这两个个函数做成系统调用，分别是 `getpriority(int tid)`, `setpriority(int tid, int prio)`

静态调度 `schedule`:

```
void schedule(){
    struct tcb *select = g_task_head;
    struct tcb *my_select=g_task_running;
    while(select!=NULL){
        if((select->tid != 0) &&(select->state == TASK_STATE_READY))
```

```
            {
                //if(my_select==NULL) {my_select=select; continue;}
                if(select->nice<=my_select->nice)//选择等待队列里的线程优先级高的
                    my_select=select;
                if(my_select->tid==0){//跳过task0运行
                    my_select=select;
                }
            }
        select=select->next;
    }

    if(my_select==g_task_running) {
        if(g_task_running->state == TASK_STATE_READY)
            return;
        my_select = task0; //仅当没有其他可运行的线程时，才能调度
    }
    g_resched = 0;
    switch_to(my_select);
}
```

线程控制块 tcb 中

增加 estcpu 属性，在函数 sys_task_create 中初始化 estcpu=0

增加 priority 属性，在函数 sys_task_create 中初始化 priority=0

timer.c 中增加全局变量 g_load_avg：表示系统的平均负荷

用浮点(float 表示 g_load_avg 和 estcpu：精度高，效率低

动态调度 schedule：

```
void schedule()
{
    struct tcb *select=g_task_head;
    struct tcb *my_select=g_task_running;
    while (select != NULL)
    {
        select->priority = 127 - fixedpt_toint(fixedpt_div(select->estcpu,
fixedpt_fromint(4))) - select->nice * 2; //计算所有线程的priority
        select = select->next;
    }
    //动态优先级
    select = g_task_head;
    while(select!=NULL){
        if((select->tid != 0) &&(select->state == TASK_STATE_READY)) {
            if(my_select->priority<select->priority)
                {
                    my_select=select;//选择等待队列里的线程优先级高的
```

```
                    }
                else if(my_select->tid==0)
                {
                    my_select=select;
                }
            }
            select=select->next;
    }

    if(my_select==g_task_running) {
        if(my_select->state == TASK_STATE_READY)
            return;
        my_select = task0;
    }

    printk("0x%d -> 0x%d\r\n", (g_task_running == NULL) ? -1 : g_task_running->tid,
select->tid);

    g_resched = 0;
    switch_to(my_select);
}
```

timer.c中添加如下

```
g_task_running->estcpu = fixedpt_add(g_task_running->estcpu, FIXEDPT_ONE);//计算线程使用CPU
时间estcpu

    if(g_timer_ticks % HZ==0){//每隔一秒计算一次
        int nready=0; //表示处于就绪状态的线程个数
        struct tcb *my_select=g_task_head;
        int nice;//g_task_running->nice;
                //my_select=g_task_head;
        fixedpt ratio;
        while(my_select!=NULL){
            if(my_select->state==TASK_STATE_READY) nready++;
            nice=my_select->nice;
            ratio = fixedpt_mul(FIXEDPT_TWO, g_load_avg);                    //每秒钟为所有
线程（运行、就绪和等待）更新一次
            ratio = fixedpt_div(ratio, fixedpt_add(ratio, FIXEDPT_ONE));
            my_select->estcpu = fixedpt_add(fixedpt_mul(ratio,my_select->estcpu),
                                    fixedpt_fromint(nice));
            my_select=my_select->next;
        }
        fixedpt r59_60 = fixedpt_div(fixedpt_fromint(59), fixedpt_fromint(60));//计算系统的
平均负荷g_load_avg
        fixedpt r01_60 = fixedpt_div(FIXEDPT_ONE,fixedpt_fromint(60));
        g_load_avg = fixedpt_add(fixedpt_mul(r59_60, g_load_avg),
                        fixedpt_mul(r01_60,  fixedpt_fromint(nready)));
```

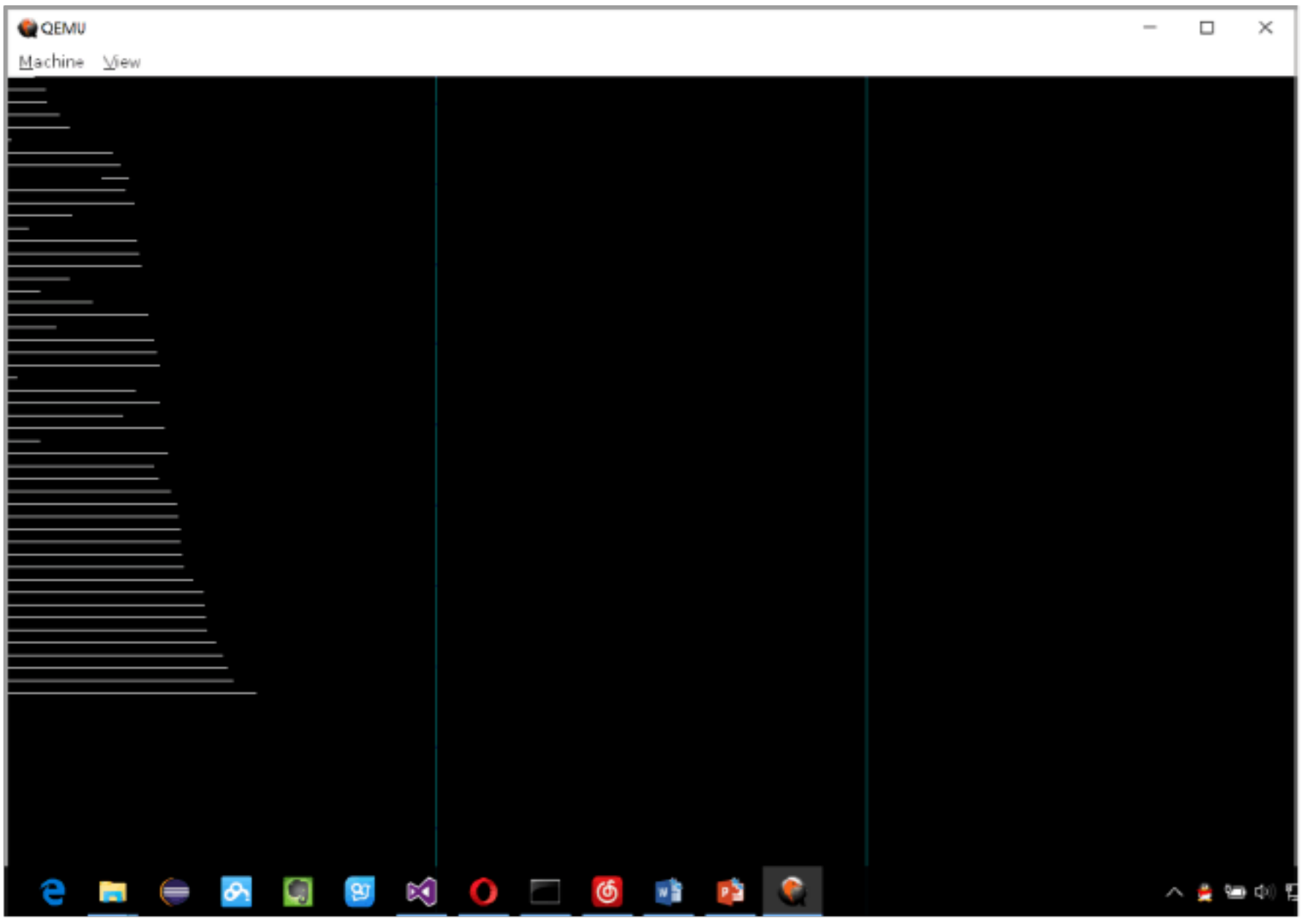
```
    }

```

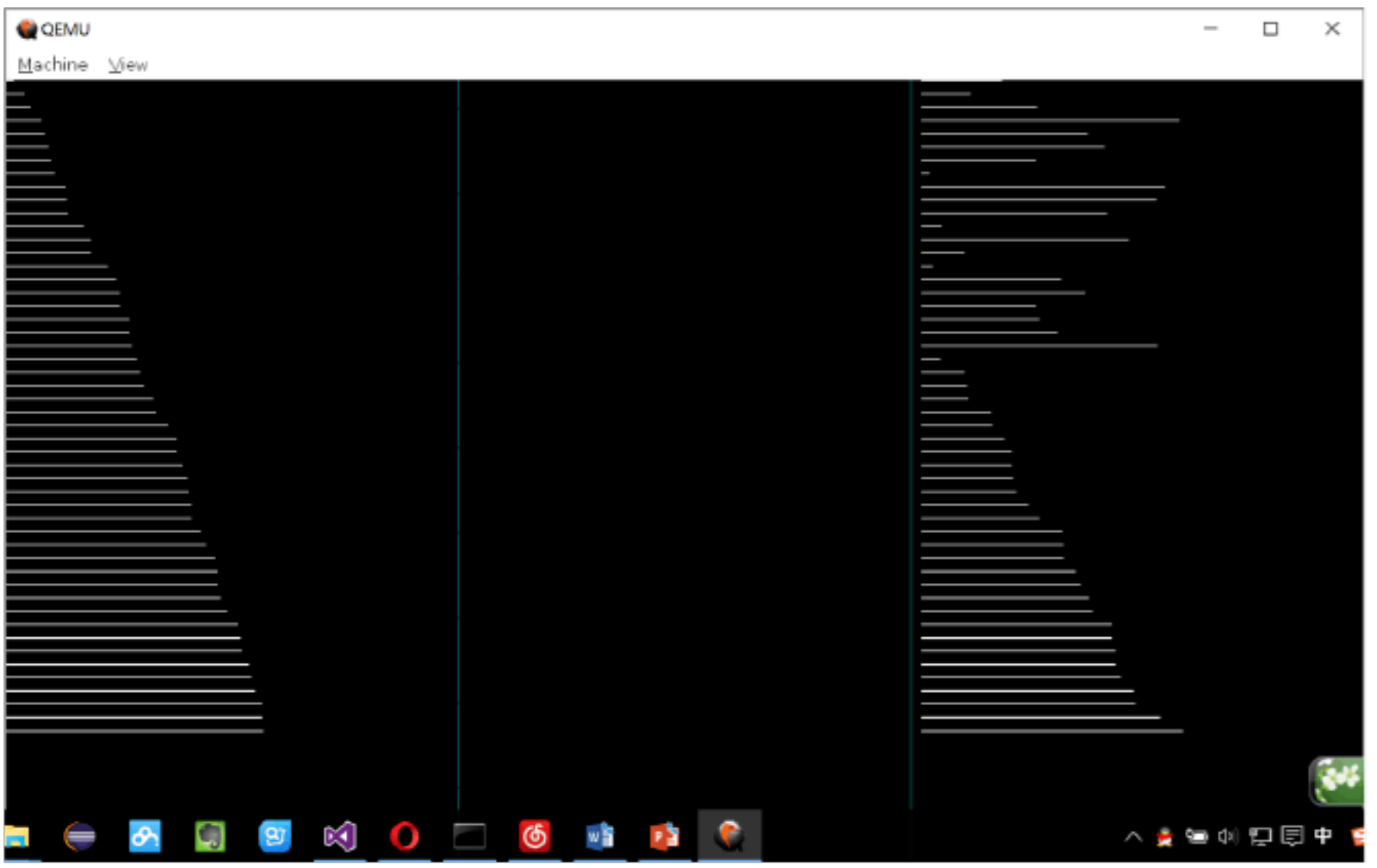
主函数：

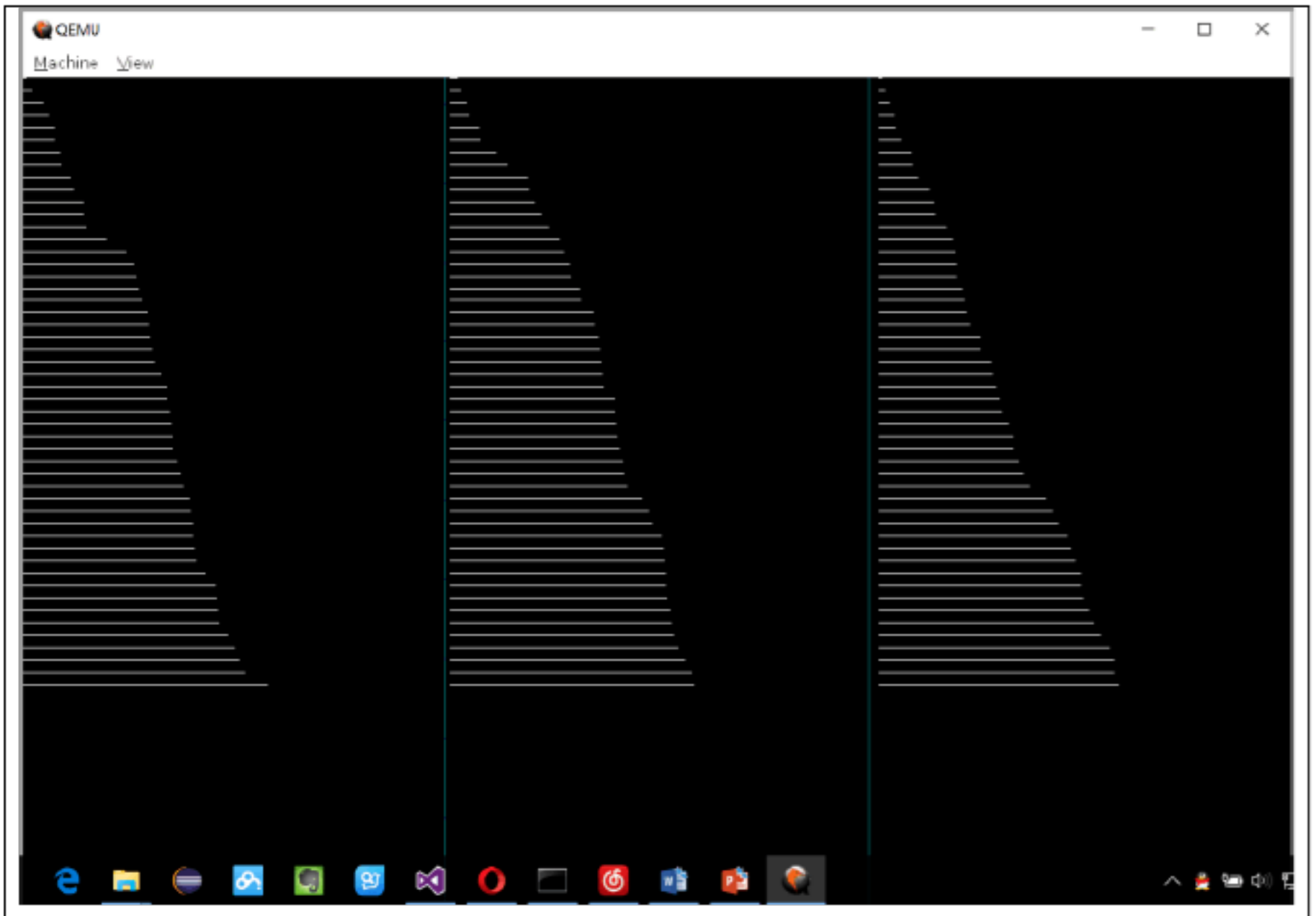
```
int mode = 0x0118;
    initGraphics(mode);
    int y = 0;
    for (y = 0; y < g_mib.YResolution; y++){
        setPixel(g_mib.XResolution / 3, y, RGB(0, 125, 125));
        setPixel(g_mib.XResolution / 3 * 2, y, RGB(0, 125, 125));
    }
    int* pcode_exit;
    //申请用户栈
    unsigned char* stack_foo1 = (unsigned char *)malloc(1024 * 1024);
    unsigned char* stack_foo2 = (unsigned char *)malloc(1024 * 1024);
    unsigned char* stack_foo3 = (unsigned char *)malloc(1024 * 1024);
    unsigned char* stack_foo4 = (unsigned char *)malloc(1024 * 1024);

    int tid_foo1, tid_foo2, tid_foo3;
    setpriority(0, 8);
    //创建冒泡排序线程函数1
    tid_foo1 = task_create(stack_foo1 + 1024 * 1024, tsk_foo_line1, (void *)0);
    setpriority(tid_foo1, 1);
    //创建选择排序线程函数2
    tid_foo2 = task_create(stack_foo2 + 1024 * 1024, tsk_foo_line2, (void *)0);
    setpriority(tid_foo2, 10);
    //创建插入排序线程函数3
    tid_foo3 = task_create(stack_foo3 + 1024 * 1024, tsk_foo_line3, (void *)0);
    setpriority(tid_foo3, 8);
    setpriority(0, 35);

    //用户栈释放
    task_wait(tid_foo1, pcode_exit);
    free(stack_foo1);
    task_wait(tid_foo2, pcode_exit);
    free(stack_foo2);
    task_wait(tid_foo3, pcode_exit);
    free(stack_foo3);
```

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）

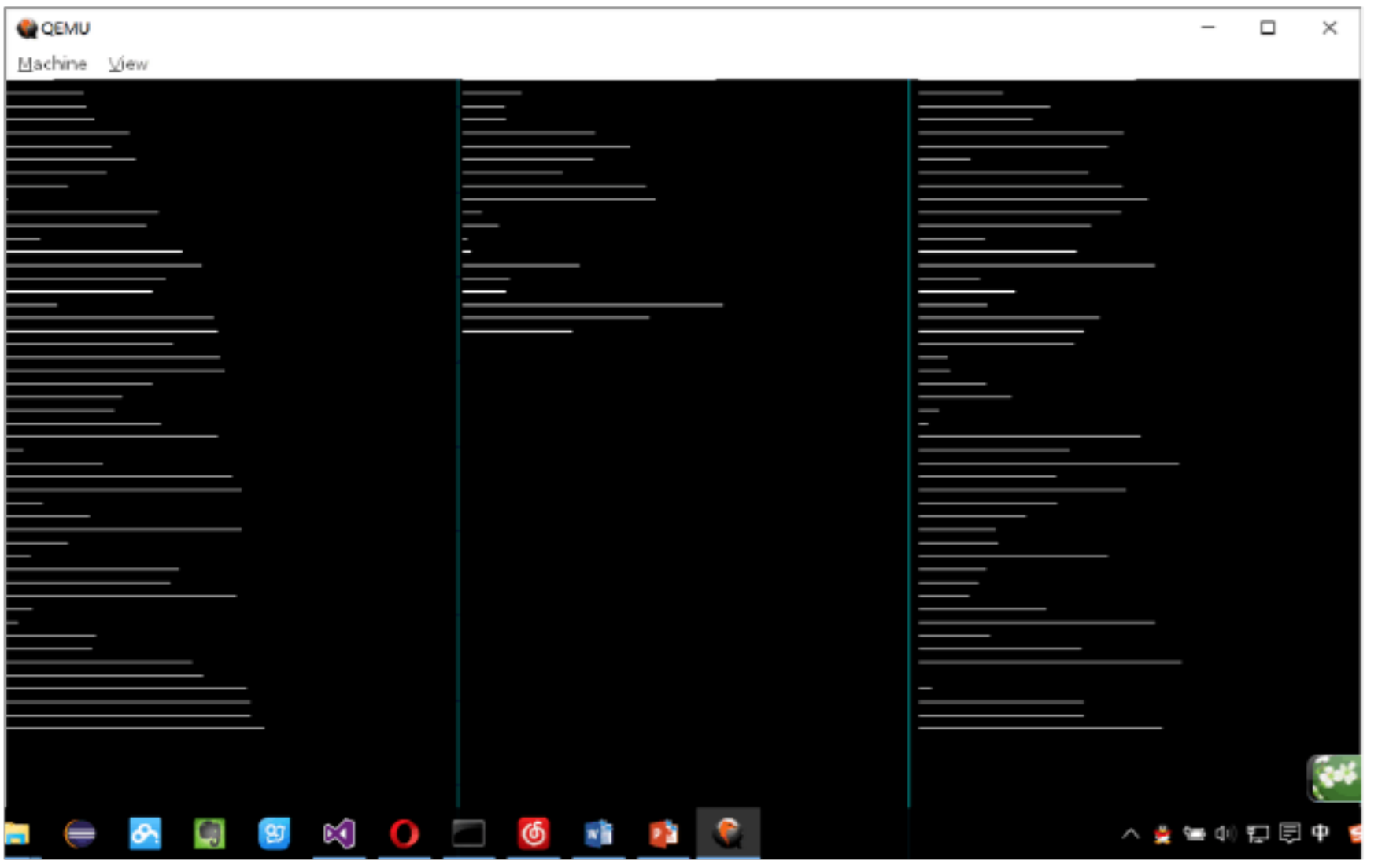
静态优先级：

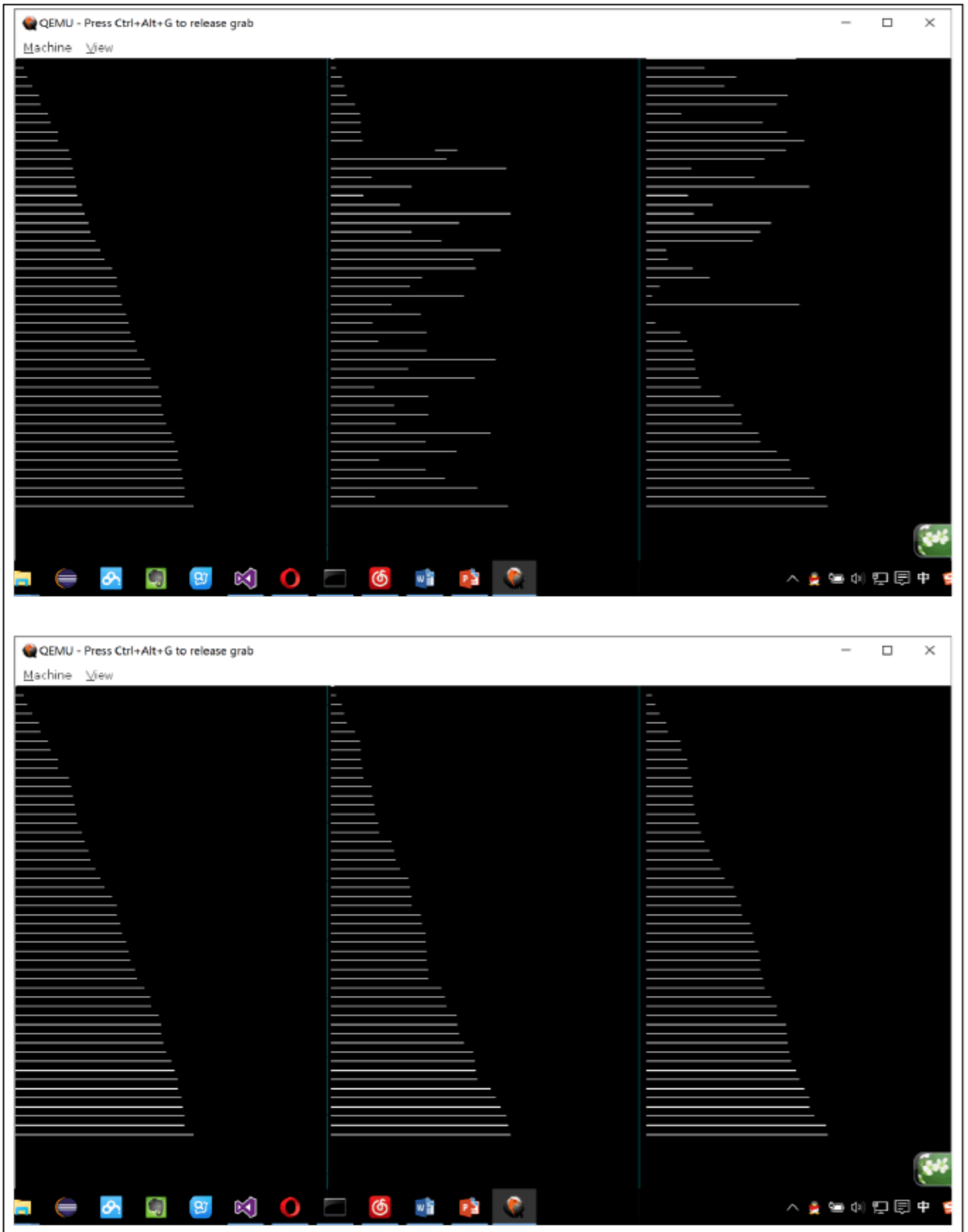






动态优先级：




QEMU - Press Ctrl+Alt+G to release grab
Machine View
QEMU - Press Ctrl+Alt+G to release grab
Machine View