重 庆 大 学

学 生 实 验 报 告

实验课程名称 操作系统

开课实验室 DS1503

学 院 大数据与软件学院 年级 软件工程 专业班 01

学 生 姓 名 _____ 学 号 _____

开 课 时 间 2024 至 2025 学年第 二 学期

总 成 绩	
教师签名	

软件学院制

《操作系统》实验报告

开课实验室： **2025 年 3 月 1 日**

学院	大数据与软件学院	年级、专业、班	2023/软件工程/01	姓名		成绩	
课程名称	操作系统	实验项目名称	实验三：线程的调度	指导教师	刘寄		
教师评语	教师签名： 年 月 日						

一、实验目的

实现线程创建和简单图形化实例

二、实验内容

1.

- Step2：增加系统调用
  - int getpriority(int tid)
    - 成功返回线程tid的(nice+NZERO)，失败返回-1
  - int setpriority(int tid, int prio)
    - 把线程tid的nice设为(prio-NZERO)
      - prio必须在[0,2*NZERO-1]内
    - 成功返回0，失败返回-1
  - 注意
    - 如果参数tid=0，表示获取/设置当前线程的nice值。

2.

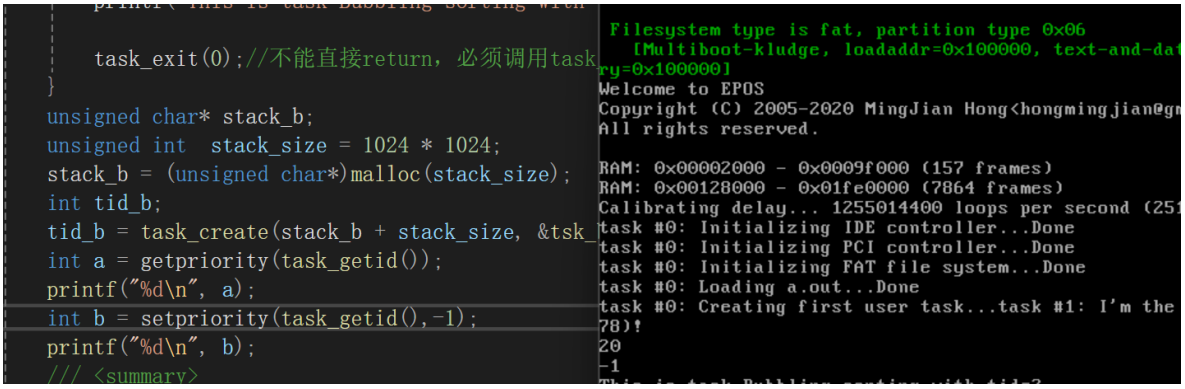
- Step5：测试调度器
  - 创建两个冒泡排序的线程，记为A和B
    - 分别占用屏幕的两个位置
      - 在屏幕上用进度条动态显示两个线程的静态优先级
    - 排序线程的速度不能太快或太慢，否则无法控制优先级
      - 用nanosleep让线程暂停运行一段时间
  - 另外创建一个控制线程
    - 循环等待键盘输入
      - int key = getchar();
    - 如果key=0x4800(up)/0x5000(down)
      - 调用setpriority调高/低A线程的静态优先级
      - 更新A的优先级进度条
    - 如果key=0x4d00(right)/0x4b00(left)
      - 调用setpriority调高/低B线程的静态优先级
      - 更新B的优先级进度条
    - 把控制线程的静态优先级设置到最高，以保证控制效果

三、使用仪器、材料

虚拟机，编译器

三、实验过程原始记录(数据、图表、计算等)

先实现两个线程优先级设置函数；



```
    task_exit(0);//不能直接return，必须调用task
}
unsigned char* stack_b;
unsigned int  stack_size = 1024 * 1024;
stack_b = (unsigned char*)malloc(stack_size);
int tid_b;
tid_b = task_create(stack_b + stack_size, &tsk_
int a = getpriority(task_getid());
printf("%d\n", a);
int b = setpriority(task_getid(),-1);
printf("%d\n", b);
/// <summary>
```

```
Filesystem type is fat, partition type 0x06
  [Multiboot-kludge, loadaddr=0x100000, text-and-da
ry=0x100000]
Welcome to EPOS
Copyright (C) 2005-2020 MingJian Hong<hongmingjian@gm
All rights reserved.

RAM: 0x00002000 - 0x0009f000 (157 frames)
RAM: 0x00128000 - 0x01fe0000 (7864 frames)
Calibrating delay... 1255014400 loops per second (251
task #0: Initializing IDE controller...Done
task #0: Initializing PCI controller...Done
task #0: Initializing FAT file system...Done
task #0: Loading a.out...Done
task #0: Creating first user task...task #1: I'm the
78)!
20
-1
```

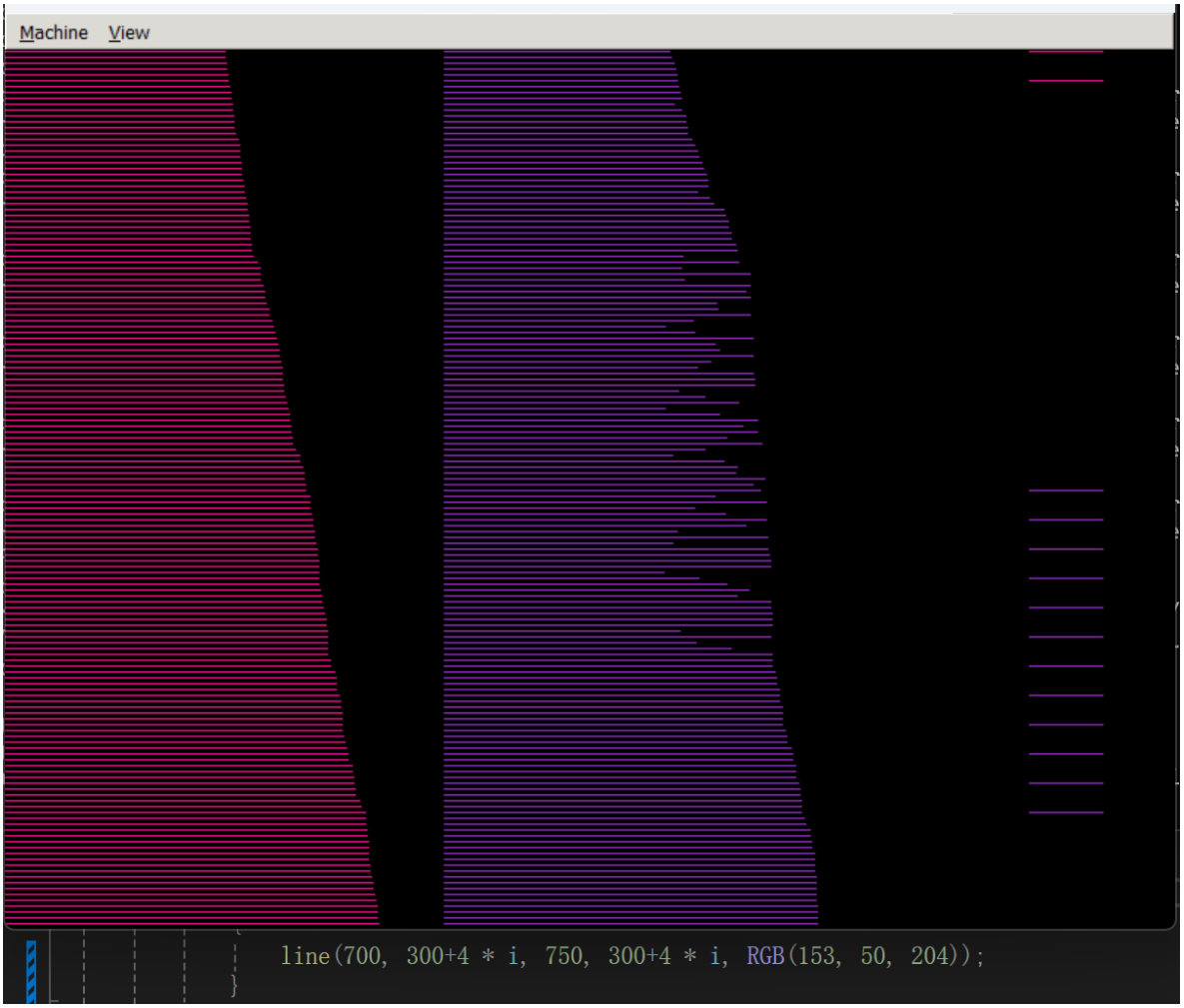
然后实现优先级算法schedule函数

```
void schedule()
{
    struct tcb* best = NULL;          // 新增：记录最高优先级的线程
    struct tcb* current = g_task_running;
    int max_priority = -1;            // 新增：记录最高优先级值

    /*--- 遍历任务链表，选择优先级最高的就绪线程 ---*/
    do {
        current = current->next;
        if (current == NULL)
            current = g_task_head;

        // 跳过 task0 和非就绪状态的线程
        if (current->tid == 0 || current->state != TASK_STATE_READY)
            continue;

        // 比较优先级（新增逻辑）
        if (best == NULL || current->priority > max_priority) {
            best = current;
            max_priority = current->priority;
        }

    } while (current != g_task_running); // 遍历一圈后结束
```

```
    /*--- 确定最终选择的线程 ---*/
    struct tcb* select = best;

    // 没有找到其他就绪线程时的处理（保留原逻辑）
    if (select == NULL) {
        if (g_task_running->state == TASK_STATE_READY)
            return;
        select = task0;
    }

    /*--- 原有安全检查 ---*/
    if (select->signature != TASK_SIGNATURE)
        printk("warning: kernel stack of task #%d overflow!!!", select->tid);

    /*--- 执行上下文切换 ---*/
    g_resched = 0;
    switch_to(select);
}
```

最后添加四个线程（冒泡A/B，GUI，控制）

五、实验结果及分析



设置的值越小优先级越高则效率越高越快完成排序

main源代码：

```
struct timespec req = {
    .tv_sec = 0,      // 秒
    .tv_nsec = 20000000  // 纳秒（500,000,000 ns = 0.5 秒）
};
void tsk_b(void* pv)
{
    tid_b = task_getid();
    setpriority(task_getid(), priority_b);
    printf("This is task Bubbling sorting with tid=%d\r\n", task_getid());
    //生成随机数组：
    unsigned int seed = (unsigned int)task_getid() + (unsigned int)time(NULL);
    srand(seed);// 将线程 id 作为种子的一部分避免生成数组一样
    int i;
    int j;

    for (i = 0; i < 150; i++) {
        b[i] = (rand() % 106) + 150;//为了可视化明显，生成 150~255 之间的数
    }
```

```
    // 冒泡排序
    for (i = 0; i < 149; i++)
    {

        for (j = 0; j < 149 - i; j++)
        {
            if (b[j] > b[j + 1])
            {
                int t = b[j];
                b[j] = b[j + 1];
                b[j + 1] = t;

            }
            /* 每次比较后短暂让出 CPU */
            nanosleep(&(struct timespec) { .tv_nsec = 100000 }, NULL); // 100 μs
        }
        nanosleep(&(struct timespec) { .tv_nsec = priority_b*500000 }, NULL);//避免休眠时间弱化优先级，让优先级参与到休眠时间
    }

    task_exit(0);//不能直接 return，必须调用 task_exit
}
void tsk_a(void* pv)
{
    tid_a = task_getid();
    setpriority(task_getid(), priority_a);
    //生成随机数组：
    unsigned int seed = (unsigned int)task_getid() + (unsigned int)time(NULL);
    srand(seed);// 将线程 id 作为种子的一部分避免生成数组一样
    int i;
    int j;

    for (i = 0; i < 150; i++) {
        a[i] = (rand() % 106) + 150;//为了可视化明显，生成 150~255 之间的数
    }
    // 冒泡排序
    for (i = 0; i < 149; i++)
    {
        for (j = 0; j < 149 - i; j++)
        {
            if (a[j] > a[j + 1])
            {
                int t = a[j];
                a[j] = a[j + 1];
                a[j + 1] = t;
```

```
            }
            /* 每次比较后短暂让出 CPU */
            nanosleep(&(struct timespec) { .tv_nsec = 100000 }, NULL); // 100 μs
        }
        nanosleep(&(struct timespec) { .tv_nsec = priority_a * 500000 }, NULL);//避免休眠时间弱化优先级，让优先级参与到休眠时间
    }
    task_exit(0);//不能直接 return，必须调用 task_exit
}
/*图形化界面*/
void tsk_GUI(void* pv) {
    init_graphic(0x0143);
    int last_a[150] = { 0 }, last_b[150] = { 0 };
    int last_a_priority = 0;
    int last_b_priority = 0;
    int i;
    while (1) {
        // 实时更新左侧（数组 a）
        for (i = 0; i < 150; i++) {
            if (last_a[i] != a[i]) {
                line(0, 4 * i, last_a[i], 4 * i, RGB(0, 0, 0));   // 清除旧线
                line(0, 4 * i, a[i], 4 * i, RGB(255, 20, 147));   // 绘制新线
                last_a[i] = a[i];
            }
        }

        // 实时更新右侧（数组 b）
        for (i = 0; i < 150; i++) {
            if (last_b[i] != b[i]) {
                line(300, 4 * i, 300 + last_b[i], 4 * i, RGB(0, 0, 0));
                line(300, 4 * i, 300 + b[i], 4 * i, RGB(153, 50, 204));
                last_b[i] = b[i];
            }
        }
        //绘制优先级
        if (last_a_priority > priority_a) {
            for (i = priority_a ; i < last_a_priority; i+=5)
            {
                line(700, 4 * i, 750, 4 * i, RGB(0,0,0));
            }
        }
        if (last_a_priority < priority_a) {
            for (i = last_a_priority; i < priority_a; i+=5)
            {
                line(700, 4 * i, 750, 4 * i, RGB(255, 20, 147));
            }
```

```
        }

        if (last_b_priority < priority_b) {
            for (i = priority_b; i < last_b_priority; i+=5)
            {
                line(700, 300+4 * i, 750,300+ 4 * i, RGB(0,0,0));
            }
        }
        if (last_b_priority < priority_b) {
            for (i = last_b_priority; i < priority_b; i+=5)
            {
                line(700, 300+4 * i, 750, 300+4 * i, RGB(153, 50, 204));
            }
        }
        // 控制刷新率（约 30FPS）
        for (i = 0; i < 50000; i++) asm volatile("nop");
    }
    task_exit(0);
}
/**
 * 第一个运行在用户模式的线程所执行的函数
 */
//控制线程
void tsk_control(void* pv) {
    setpriority(task_getid(), 127);
    //init_keyboard(); // 需要初始化键盘驱动
    while (1) {
        int key = getchar();
        switch (key)
        {
        case 0x4800:
        {

            if (priority_a < 117) {
                priority_a+=10;
                setpriority(tid_a, priority_a);
            }
            break;
        }
        case 0x5000:
        {
            if (priority_a > 10) {
                priority_a-=10;
                setpriority(tid_a, priority_a);
            }
```

```
            break;
        }
        case 0x4d00:
        {
            if (priority_b < 117) {
                priority_b+=10;
                setpriority(tid_b, priority_b);
            }

            break;
        }
        case 0x4b00:
        {
            if (priority_b > 10) {
                priority_b-=10;
                setpriority(tid_b, priority_b);
            }

            break;
        }
        default:
            break;
        }


    }

    task_exit(0);
}

void main(void *pv)
{
    printf("task #%d: I'm the first user task(pv=0x%08x)!\r\n",
            task_getid(), pv);

    //TODO: Your code goes here
    //

    unsigned char* stack_b;
    unsigned int  stack_size = 1024 * 1024;
    stack_b = (unsigned char*)malloc(stack_size);
    int tid_b;
    tid_b = task_create(stack_b + stack_size, &tsk_b, (void*)0);

    unsigned char* stack_a;
    stack_a = (unsigned char*)malloc(stack_size);
```

```
    int tid_a;
    tid_a = task_create(stack_a + stack_size, &tsk_a, (void*)0);

    unsigned char* stack_GUI;
    stack_GUI = (unsigned char*)malloc(stack_size);
    int tid_GUI;
    tid_GUI = task_create(stack_GUI + stack_size, &tsk_GUI, (void*)0);

    unsigned char* stack_control;
    stack_control = (unsigned char*)malloc(stack_size);
    int tid_control;
    tid_control = task_create(stack_control + stack_size, &tsk_control, (void*)0);
    /// <summary>
    /// ssh -i C:\Users\dys14\.ssh\si_20231265.pub  si@10.236.101.66
    ///这里测试用例试探提交条件
    /// </summary>
    /// 0001

    while(1)
        ;
    task_exit(0);
}
```

实验报告打印格式说明

1. 标题：三号加粗黑体
2. 开课实验室：5 号加粗宋体
3. 表中内容：

(1) 标题：5 号黑体

(2) 正文：5 号宋体

4. 纸张：16 开(20cm×26.5cm)
5. 版芯

上距：2cm

下距：2cm

左距：2.8cm

右距：2.8cm

说明： 1、“年级专业班”可填写为“00 电子 1 班”，表示 2000 级电子工程专业第 1 班。

2、实验成绩可按五级记分制（即优、良、中、及格、不及格），或者百分制记载，若需要将实验成绩加入对应课程总成绩的，则五级记分应转换为百分制。