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 DS1502

学 院 大数据与软件学院 年级 22 专业班 软件工程 2 班

学 生 姓 名 潘铷葳 学 号 20221982

开 课 时 间 2023 至 2024 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：DS1502　　2024　年 4 月 13 日

学院	大数据与软件学院	年级、专业、班	2022 级软件工程 2 班	姓名	潘铷葳	成绩	
课程名称	操作系统	实验项目名　称	线程的调度	指导教师	操作系统		
教师评语	教师签名： 年　月　日						

一、 实验目的

1．学习如何创建多个线程，并了解如何为每个线程分配合适的堆栈空间以及设置线程函数。

2．了解操作系统如何对多个线程进行调度和管理，以及了解不同线程之间的执行顺序和并发性。

3.了解操作系统如何根据线程的优先级调度线程的执行顺序，以及了解动态调整线程优先级的影响。

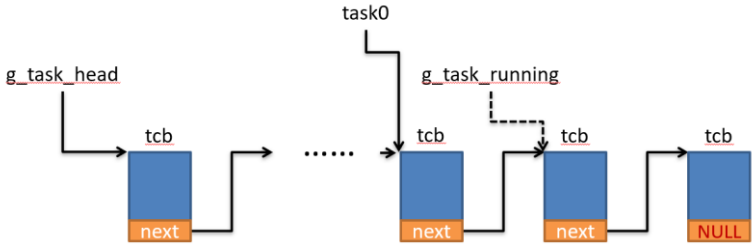
二、实验内容

优先级

静态优先级(nice)：内核不会修改它，不随时间而变化，除非用户通过系统调用setpriority进行修改。

动态优先级(priority)：内核根据线程使用CPU的状况、静态优先级nice和系统负荷计算出来，会随时间而变化。

最终的调度依据，即调度器只根据动态优先级进行调度。



注意

1、在系统启动过程中，g_task_running是NULL！

2、task0(tid=0)是系统空闲线程（system idle thread）

系统调用接口，线程相关函数：

- Step1：在“struct tcb”中增加线程的静态优先级 nice
- Step2：增加系统调用
  - int getpriority(int tid)
  - int setpriority(int tid, int prio)
- Step3：在“struct tcb”中，再增加两个属性
  - estcpu：表示线程最近使用了多少 CPU 时间
  - priority：表示线程的动态优先级
- Step4：增加一个全局变量
- Step5：属性计算
- Step6：修改 task.c 中的函数 schedule，实现优先级调度算法
- （续）Step5：测试调度器
  - 创建两个冒泡排序的线程，记为 A 和 B
  - 另外创建一个控制线程

三、实验过程原始记录(数据、图表、计算等)

静态优先级

（1） 在 Kernel.c 中添加静态优先级相关变量

```
struct tcb {
    /*hardcoded*/
    uint32_t    kstack;      /*saved top of the kernel stack for this task*/

    int         tid;         /* task id */
    int         state;       /* -1:waiting,0:running,1:ready,2:zombie */
#define TASK_STATE_WAITING  -1
#define TASK_STATE_READY     1
#define TASK_STATE_ZOMBIE    2

    int         timeslice;   //时间片
#define TASK_TIMESLICE_DEFAULT 4

    int         code_exit;   //保存该线程的退出代码
    struct wait_queue *wq_exit; //等待该线程退出的队列

    struct tcb  *next;
    struct fpu   fpu;        //数学协处理器的寄存器

    fixedpt estcpu; //表示线程最近花了多少时间

    fixedpt priority;//动态优先级
    #define PRI_USER_MIN 0
    #define PRI_USER_MAX 127

    int nice;//声明静态优先级
    #define NZERO 20

    uint32_t     signature;  //必须是最后一个字段
#define TASK_SIGNATURE 0x20160201
};
```

（2） 在 task.c 中的 sys_task_create 函数里初始化变量

```
static int tid = 0;
struct tcb *new;
new->nice = 0;//初始化nice
```

（3） 在 task.c 中添加获取优先级函数

```
//获取当前线程优先级
int sys_getpriority(int tid) {
    uint32_t flags;
        struct tcb* tsk;
    save_flags_cli(flags);
    //tid为零则获取当前线程
    if (tid == 0) {
        tsk = g_task_running;
    }//不为零则获取指定线程
    else {
        tsk = get_task(tid);
    }
    restore_flags(flags);
    if (tsk != NULL)return tsk->nice + NZERO;
    else return -1;
}
```

（4） 在 task.c 中添加设置优先级函数

```
//设置优先级
int sys_setpriority(int tid, int prio) {
    uint32_t flags;
    struct tcb* tsk;
    //prio超出范围则设置失败
    if (prio < 0 || prio>2 * NZERO - 1) {
        return -1;
    }
    save_flags_cli(flags);
    if (tid == 0) {
        tsk = g_task_running;
    }
    else {
        tsk = get_task(tid);
    }
    restore_flags(flags);
    if (tsk != NULL) {
        tsk->nice = prio - NZERO;
        return 0;
    }
    else {
        return -1;
    }
}
```

（5） 在 kernel.h 中进行声明上述函数

```
struct tcb *sys_task_create(void *tos, void (*func)(void *pv), void *pv);
void        sys_task_exit(int code_exit);
int         sys_task_wait(int tid, int *pcode_exit);
int         sys_task_getid();
void        sys_task_yield();

int sys_getpriority(int tid);
int sys_setpriority(int tid, int prio);

int printk(const char *fmt,...);
```

（6） 在 syscall-nr.h 中定义这两个函数的系统调用号

```
#define SYSCALL_getpriority 11
#define SYSCALL_setpriority 12
```

（7） 在 machdep.c 中的 syscall(struct context *ctx)函数中增加系统调用的分支

```
    //实验三
case SYSCALL_getpriority: {
    int tid = *(int*)(ctx->esp + 4);
    ctx->eax = sys_getpriority(tid);
}break;
case SYSCALL_setpriority: {
    int tid = *(int*)(ctx->esp + 4);
    int prio = *(int*)(ctx->esp + 8);
    ctx->eax = sys_setpriority(tid, prio);
}break;
```

（8）用户态下的汇编接口，在 syscall-wrapper.S 文件中声明上述函数

```
WRAPPER(getpriority)
WRAPPER(setpriority)
```

（9）最后在 syscall.h 中声明这两个系统调用函数

```
int getpriority(int tid);
int setpriority(int tid, int prio);
```

动态优先级

（1）在 Kernel.c 中添加静态优先级相关变量

a）Estcpu

fixedpt estcpu; //表示线程最近花了多少时间

b） Priority

```
fixedpt priority;//动态优先级
#define PRI_USER_MIN 0
#define PRI_USER_MAX 127
```

(2)在 task.c 中的 sys_task_create 函数里初始化变量

```
new->estcpu = 0;
new->priority = 0;
```

（3）在 timer.c 中 isr_timer(uint32_t irq, struct context *ctx)函数编写 estcpu 以及 priority 处理代码

```
void isr_timer(uint32_t irq, struct context *ctx)
{
    g_timer_ticks++;
    //sys_putchar('.');
    if(g_task_running != NULL) {
        //如果是task0在运行，则强制调度
        if(g_task_running->tid == 0) {
            g_resched = 1;
        } else {
            /*每次定时器中断：g_task_running->estcpu++，task0除外！*/
            g_task_running->estcpu = fixedpt_add(g_task_running->estcpu, FIXEDPT_ONE);
            /*每隔一秒计算一次*/
            if (g_timer_ticks % HZ == 0) {
                int nice;
                int nready = 0;//表示处于就绪状态的线程个数，task0除外！
                fixedpt ratio;
                struct tcb* select = g_task_head;
                while (select != NULL) {
                    //if(select==NULL) break;
                    if (select->state == TASK_STATE_READY) nready++;
                    nice = select->nice;
                    ratio = fixedpt_mul(FIXEDPT_TWO, g_load_avg);//每秒为所有线程更新一次
                    ratio = fixedpt_div(ratio, fixedpt_add(ratio, FIXEDPT_ONE));
                    select->estcpu = fixedpt_add(fixedpt_mul(ratio, select->estcpu), fixedpt_fromint(nice));
                    select = select->next;
                }
                /*g_load_avg=(59/60) ×g_load_avg+(1/60) × nready,
                  计算系统的平均负荷g_load_avg*/
                fixedpt r59_60 = fixedpt_div(fixedpt_fromint(59), fixedpt_fromint(60));
                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)));
            }
            //否则，把当前线程的时间片减一
            --g_task_running->timeslice;
            //如果当前线程用完了时间片，也要强制调度
            if(g_task_running->timeslice <= 0) {
                g_resched = 1;
                g_task_running->timeslice = TASK_TIMESLICE_DEFAULT;
            }

        }
    }
```

（4）修改 task.c 中的函数 schedule，实现优先级调度算法

```
void schedule()
{
/*********动态优先级调度*************/

        struct tcb* select = g_task_head;
        struct tcb* select_plus = g_task_running;

        while (select != NULL) {//遍历链表，计算所有线程的priority的值
            select->priority = PRI_USER_MAX -
                fixedpt_toint(fixedpt_div(select->estcpu, fixedpt_fromint(4))) -
                select->nice * 2;
            select = select->next;
        }

        select = g_task_head;
        while (select != NULL) {
            if ((select->tid != 0) && (select->state == TASK_STATE_READY)) {
                if (select->priority > select_plus->priority || select_plus->tid == 0)
                    select_plus = select;//选择等待队列中优先级最高的线程
            }
            select = select->next;
        }

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

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

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

        g_resched = 0;
        switch_to(select_plus);


}
```

验证静态优先级

Main.c 如下：

```
/*
 * vim: filetype=c:fenc=utf-8:ts=4:et:sw=4:sts=4
 */
#include <inttypes.h>
#include <stddef.h>
#include <math.h>
#include <stdio.h>
#include <sys/mman.h>
#include <syscall.h>
#include <netinet/in.h>
#include <stdlib.h>
#include "graphics.h"
#include<time.h>

extern void* tlsf_create_with_pool(void* mem, size_t bytes);
extern void* g_heap;
```

```
/**
 * GCC insists on __main
 *    http://gcc.gnu.org/onlinedocs/gccint/Collect2.html
 */
void __main()
{
    size_t heap_size = 32 * 1024 * 1024;
    void* heap_base = mmap(NULL, heap_size, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANON, -1,
0);
    g_heap = tlsf_create_with_pool(heap_base, heap_size);
}

//线程优先级
int tid_foo1, tid_foo2, tid_foo3;

//画线自定义函数
void drawLine(int x1, int y1, int x2, int y2, int extra_x, COLORREF cr)
{
    line(x1 + extra_x, (y1 / 5) * 3, x2 + extra_x, (y2 / 5) * 3, cr);
}

//睡眠函数
void mySleep()
{
    struct timespec tim, tim2;
    tim.tv_sec = 1;
    tim.tv_nsec = 10000000;
    nanosleep(&tim, &tim2);
}

//冒泡排序
void bubsort1(int* a, int n)
{
    int i, j, temp;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                //覆盖排序前的两条线段
                drawLine(-a[j], j * 6 , 0, j * 6, 250, RGB(0, 0, 0));
                drawLine(-a[j + 1], (j + 1) * 6 ,0, (j + 1) * 6 , 250, RGB(0, 0, 0));
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
```

```
                drawLine(-a[j], j * 6 , 0, j * 6 , 250, 0x4682B4);
                drawLine(-a[j + 1], (j + 1) * 6 , 0, (j + 1) * 6 , 250, 0x4682B4);
                //mySleep();
            }
            //mySleep();
        }
        mySleep();
    }
}

void bubsort2(int* a, int n)
{
    int i, j, temp;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n - i - 1; j++)
        {
            if (a[j] > a[j + 1])
            {
                //覆盖排序前的两条线段
                drawLine(0, j * 6 , a[j], j * 6 , 250, RGB(0, 0, 0));
                drawLine(0, (j + 1) * 6 , a[j + 1], (j + 1) * 6 , 250, RGB(0, 0, 0));
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
                drawLine(0, j * 6 , a[j], j * 6 , 250, 0xFFFF00);
                drawLine(0, (j + 1) * 6 , a[j + 1], (j + 1) * 6 , 250, 0xFFFF00);
                //mySleep();
            }
            //mySleep();
        }
        mySleep();
    }
}

//线程函数
void tsk_foo1(void* pv)
{
    time_t time(time_t * loc);
    srand(time(NULL));
    int myCount_1[150];
    int i, k;
    for (i = 0; i < 150; i++)
    {
        myCount_1[i] = rand() % 200;
        printf("%d\n", myCount_1[i]);
```

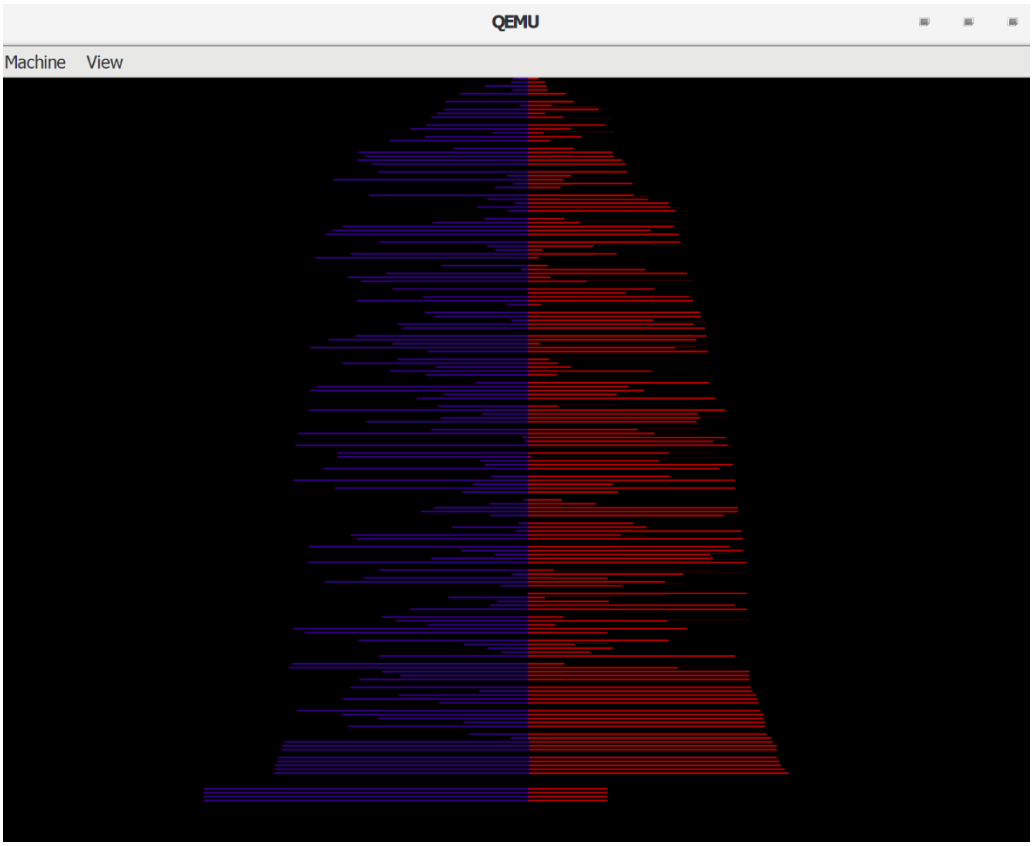
```
    }

    //显示未排序的画面
    for (k = 0; k < 150; k++)
    {
        drawLine(-myCount_1[k], k * 6 ,0, k * 6 , 250, 0x4682B4);
    }

    mySleep();
    bubsort1(myCount_1, 150);
    task_exit(0);
}

void tsk_foo2(void* pv)
{
    time_t time(time_t * loc);
    srand(time(NULL));
    int myCount_2[150];
    int i, k;
    for (i = 0; i < 150; i++)
    {
        myCount_2[i] = rand() % 200;
        printf("%d\n", myCount_2[i]);
    }

    //显示未排序的画面
    for (k = 0; k < 150; k++)
    {
        drawLine(0, k * 6 , myCount_2[k], k * 6 , 250, 0xFFFF00);
    }
    mySleep();
    bubsort2(myCount_2, 150);
    task_exit(0);
}


//优先级展示条
void show_priority(int tid, int k) {
    int length = 10 * getpriority(tid);
    int m = 0;
    switch (k)
    {
    case 1:
    {
        for (m = 0; m < 20; m++)
            drawLine(-250, 910+m, 0, 910+m, 250, RGB(0, 0, 0));
```

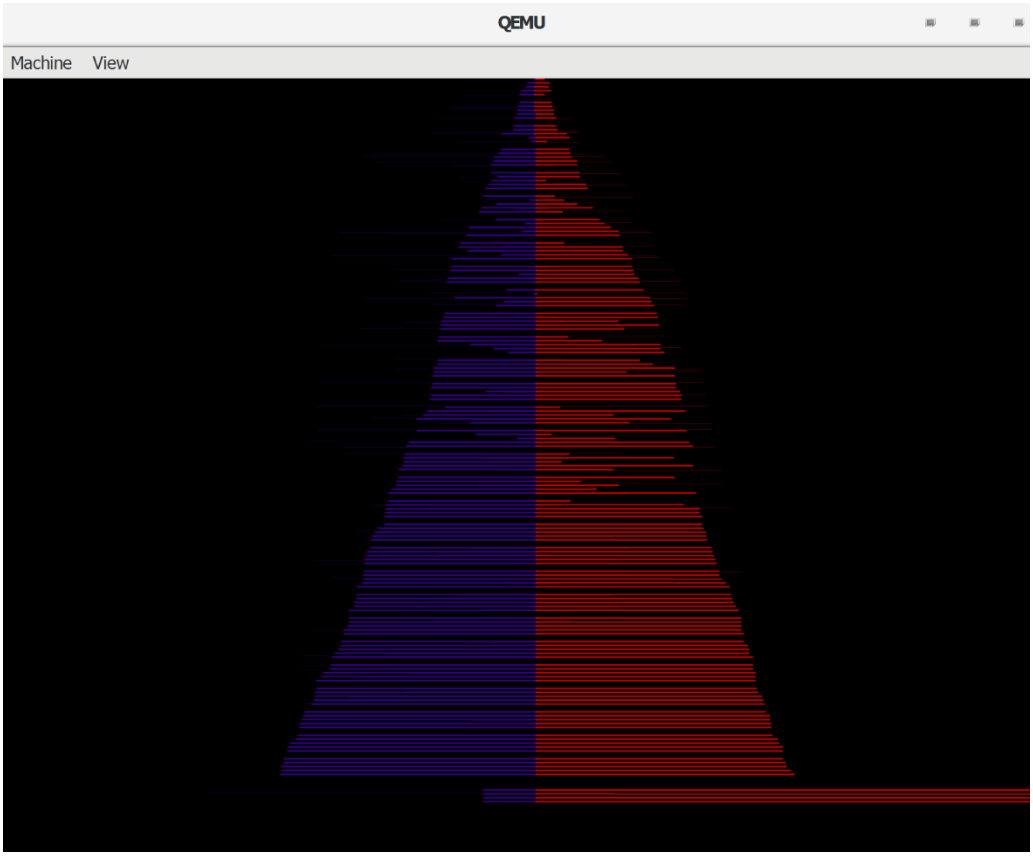
```
        for (m = 0; m < 20; m++)
            drawLine(-length, 910 + m, 0, 910 + m, 250, 0x4682B4);
    }break;
    case 2:
    {
        for (m = 0; m < 20; m++)
            drawLine(0, 910 + m, 200, 910 + m, 250, RGB(0, 0, 0));
        for (m = 0; m < 20; m++)
            drawLine(0, 910 + m, length, 910 + m, 250, 0xFFFF00);
    }break;
    default:
        break;
    }

}

//控制线程
void mytask_control(void* pv) {
    show_priority(tid_foo1,1);
    show_priority(tid_foo2, 2);
    int mykeypress;
    while (1) {
        mykeypress = getchar();
        switch (mykeypress)
        {
        case 0x4800://(up)
        {
            setpriority(tid_foo1, getpriority(tid_foo1) + 2);
            show_priority(tid_foo1, 1);
        }
        break;
        case 0x5000://(down)
        {
            setpriority(tid_foo1, getpriority(tid_foo1)-2);
            show_priority(tid_foo1, 1);
        }
        break;
//0x4d00(right)/0x4b00(left)
        case 0x4d00:
        {
            setpriority(tid_foo2, getpriority(tid_foo2) + 2);
            show_priority(tid_foo2, 2);
        }
        break;
        case 0x4b00:
        {
```

```
            setpriority(tid_foo2, getpriority(tid_foo2) - 2);
            show_priority(tid_foo2, 2);
        }
        break;
        default:
            break;
        }
    }
}

void main(void* pv)
{
    unsigned char* stack_foo_1, * stack_foo_2, * stack_foo_3;
    unsigned int  stack_size = 1024 * 1024;
    stack_foo_1 = (unsigned char*)malloc(stack_size);
    stack_foo_2 = (unsigned char*)malloc(stack_size);
    stack_foo_3 = (unsigned char*)malloc(stack_size);
    init_graphic(0x143);

    tid_foo1=task_create(stack_foo_1 + stack_size, &tsk_foo1, (void*)0);
    tid_foo2=task_create(stack_foo_2 + stack_size, &tsk_foo2, (void*)0);
    tid_foo3 = task_create(stack_foo_3 + stack_size, &mytask_control, (void*)0);
    setpriority(tid_foo3, 0);
    setpriority(tid_foo1, 10);
    setpriority(tid_foo2, 10);


    free(stack_foo_1);
    free(stack_foo_2);
    free(stack_foo_3);

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

四、实验结果及分析

最后效果图：（显示条越长，则优先级越低，相关视频在文件夹中）





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。