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	实验四：线程同步	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.

- 实现信号量
  - 编辑文件kernel/sem.c，实现如下四个函数
    - int 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ID，否则返回-1
    - int sys_sem_destroy(int semid)
      - 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(int semid)
      - 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(int semid)
      - 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

2.

- 生产者/消费者
  - Step1：首先，在图形模式下将屏幕沿垂直方向分成N份，作为N个缓冲区。其次，创建两个线程，其中一个是生产者，负责生成随机数并填到缓冲区中；另一个线程是消费者，负责把缓冲区中的随机数进行排序
    - 生产者生成随机数后，要画到缓冲区
    - 消费者完成排序之后，要清除缓冲区
  - Step2：创建一个控制线程
    - 用键up/down控制生产者的优先级，用键right/left控制消费者的优先级
    - 把控制线程的静态优先级设置到最高，以保证控制效果
    - 在屏幕上用进度条动态显示生产者和消费者的静态优先级

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)

先实现四个系统调用；

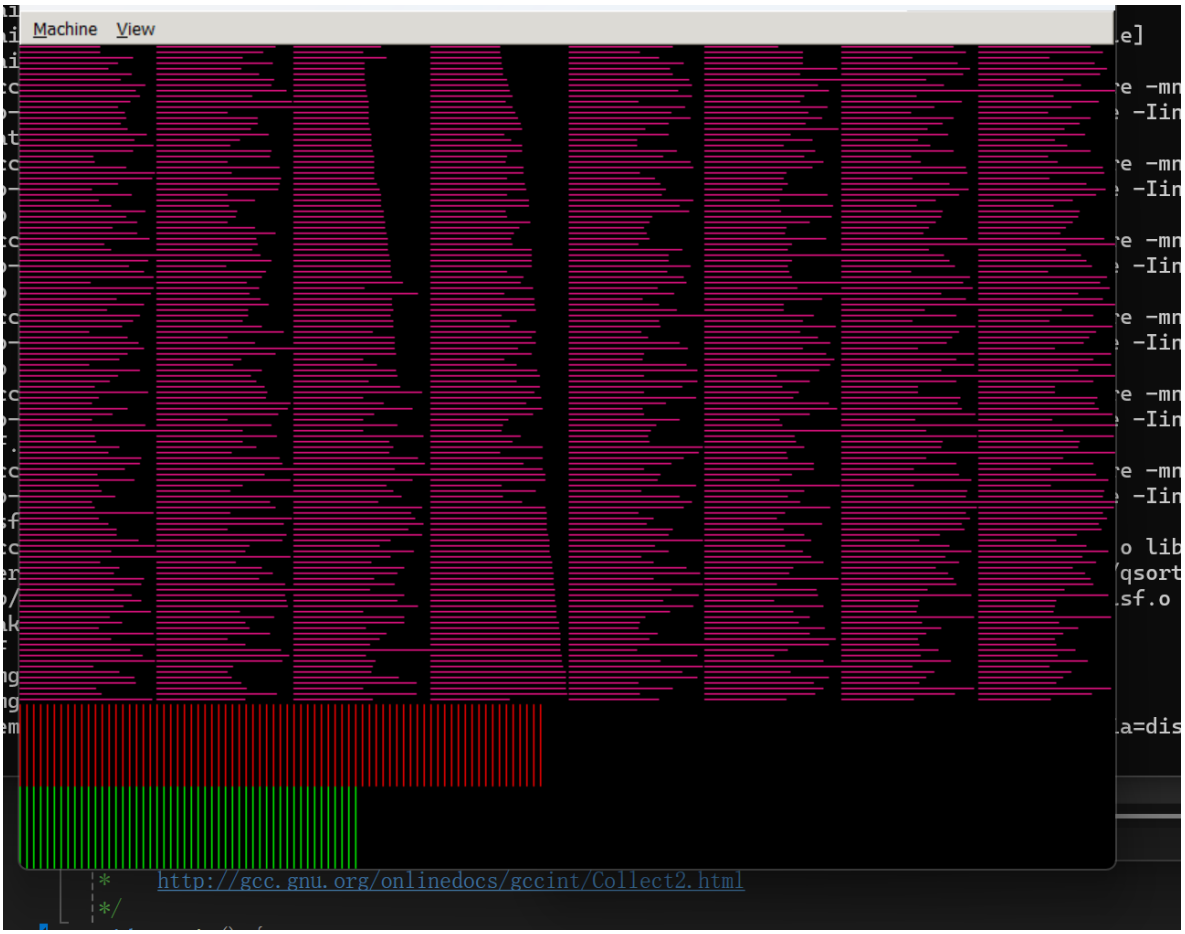
```
78)!
Success to create semaphore!
-1072604000
Semaphore wait success.
Semaphore signal success.
Semaphore destroyed.
```

实现信号量和缓冲区

```c
// 全局定义（内核或用户态）
#define BUFFER_SIZE 9  // 缓冲区容量
int mutex, empty, full; // 信号量 ID
// 在全局定义中添加一个优先级互斥锁：
int priority_mutex;
void init_semaphores() {
    mutex = sem_create(1);   // 互斥锁
    empty = sem_create(BUFFER_SIZE); // 初始空闲位置数
    full = sem_create(0);    // 初始已填充数据数
    priority_mutex = sem_create(1); // 新增优先级锁
}

// 缓冲区实现
int buffer[BUFFER_SIZE][120]; // 存储数组以可视化
int in = 0;  // 生产者写入位置
int out = 0; // 消费者读取位置

void insert_item(int* items) {
    int i;
    for (i = 0; i < 120; i++) {
        buffer[in][i] = items[i];
    }
    in = (in + 1) % BUFFER_SIZE;
}

int* remove_item() {
    int i;
    int current_out = out;
    for (i = 0; i < 120; i++) {
        item[current_out][i] = buffer[current_out][i];
        buffer[current_out][i] = 0; // 清空缓冲区位置
    }
    out = (out + 1) % BUFFER_SIZE;
    return item[out - 1];
}
```

再创建四个线程（生产者，消费者，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：

```
/*
 * vim: filetype=c:fenc=utf-8:ts=4:et:sw=4:sts=4
 */
/*动态化优先级//消费者始终没有实现循环使用缓冲区*/
#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 <time.h>
#include "graphics.h"

extern void* tlsf_create_with_pool(void* mem, size_t bytes);
extern void* g_heap;
int a[120] = {}; // 左侧
int b[120] = {}; // 右侧
int item[9][120] = {};
```

```
int priority_producer = 127;
int priority_consumer = 10;
int tid_consumer;
int tid_producer;


// 全局定义（内核或用户态）
#define BUFFER_SIZE 9  // 缓冲区容量
int mutex, empty, full; // 信号量 ID
// 在全局定义中添加一个优先级互斥锁：
int priority_mutex;
void init_semaphores() {
    mutex = sem_create(1);   // 互斥锁
    empty = sem_create(BUFFER_SIZE); // 初始空闲位置数
    full = sem_create(0);   // 初始已填充数据数
    priority_mutex = sem_create(1); // 新增优先级锁
}

// 缓冲区实现
int buffer[BUFFER_SIZE][120]; // 存储数组以可视化
int in = 0;  // 生产者写入位置
int out = 0; // 消费者读取位置

void insert_item(int* items) {
    int i;
    for (i = 0; i < 120; i++) {
        buffer[in][i] = items[i];
    }
    in = (in + 1) % BUFFER_SIZE;
}

int* remove_item() {
    int i;
    int current_out = out;
    for (i = 0; i < 120; i++) {
        item[current_out][i] = buffer[current_out][i];
        buffer[current_out][i] = 0; // 清空缓冲区位置
    }
    out = (out + 1) % BUFFER_SIZE;
    return item[out - 1];
}



/**
 * 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);
}

struct timespec req = {
    .tv_sec = 0,       // 秒
    .tv_nsec = 500000000 // 纳秒（500,000,000 ns = 0.5 秒）
};

// 生产者线程
void tsk_producer(void* pv) {
    tid_producer = task_getid();
    setpriority(task_getid(), priority_producer);

    while (1) {
        // 1. 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;

        for (i = 0; i < 120; i++) {
            a[i] = (rand() % 51) + 50; // 为了可视化明显，生成 50~100 之间的数
        }
        // 2. 等待空闲缓冲区
        sem_wait(empty);     // 若 empty > 0，继续；否则阻塞

        // 3. 获取互斥锁
        sem_wait(mutex);     // 进入临界区

        // 4. 将数据放入缓冲区
        insert_item(a);

        // 5. 释放互斥锁
        sem_signal(mutex);

        // 6. 通知消费者有新数据
        sem_signal(full);    // full++
        nanosleep(&req, NULL);


    }
```

```
    task_exit(0); // 不能直接 return，必须调用 task_exit
}

// 消费者线程
void tsk_consumer(void* pv) {
    tid_consumer = task_getid();
    setpriority(task_getid(), priority_consumer);
    int cur_out;
    int k;
    while (1)
    {
        // 1. 等待有数据可消费
        sem_wait(full);      // 若 full > 0，继续；否则阻塞

        // 2. 获取互斥锁
        sem_wait(mutex);     // 进入临界区

        // 3. 从缓冲区取出数据并
        int* it = remove_item();
        cur_out = (out - 1 + BUFFER_SIZE) % BUFFER_SIZE;

        // 4. 释放互斥锁
        sem_signal(mutex);

        // 5. 通知生产者有空闲位置
        sem_signal(empty);   // empty++

        // 6. 消费数据
        for (k = 0; k < 120; k++) {
            item[cur_out][k] = it[k];
        }
        int i;
        int j;


        // 冒泡排序
        for (i = 0; i < 119; i++) {
            for (j = 0; j < 119 - i; j++) {
                if (item[cur_out][j] > item[cur_out][j + 1]) {
                    int t = item[cur_out][j];
                    item[cur_out][j] = item[cur_out][j + 1];
                    item[cur_out][j + 1] = t;
                }
            }
            struct timespec ts = { .tv_sec = 0, .tv_nsec = priority_consumer * 500000 };
            nanosleep(&ts, NULL); // 避免休眠时间弱化优先级，让优先级参与到休眠时间
```

```
        }
       // nanosleep(&req, NULL);
    }

    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_item[9][120] = { 0 };
    int last_p_item[9][120] = { 0 };
    int last_priority_pro = 0;
    int last_priority_con = 0;


    while (1) {
        //int safe_in, safe_out;
        //// 获取当前的 in 和 out 值
        //sem_wait(mutex);
        //safe_in = in;
        //safe_out = out;
        //sem_signal(mutex);

        // 更新生产者生产原始数组
        int j;
        int i;
        for (j = 0; j < BUFFER_SIZE; j++) {
            for (i = 0; i < 120; i++) {
                if (last_p_item[j][i] != buffer[j][i]) {
                    line(j * 100, 4 * i, j * 100 +100, 4 * i, RGB(0, 0, 0));   // 清除旧线
                    line(j * 100, 4 * i, j * 100 + buffer[j][i], 4 * i, RGB(255, 20, 147));   //
绘制新线
                    last_p_item[j][i] = buffer[j][i];
                }
            }
        }

        ////// 实时更新消费者
        //int cur_out = (safe_out - 1 + BUFFER_SIZE) % BUFFER_SIZE;
        //for (i = 0; i < 120; i++) {
        //    if (last_item[cur_out][i] != item[cur_out][i]) {
        //        line(cur_out * 100, 4 * i, cur_out * 100 + 100, 4 * i, RGB(0, 0, 0));   // 清
除旧线
        //        line(cur_out * 100, 4 * i, cur_out * 100 + item[cur_out][i], 4 * i, RGB(255,
20, 147));   // 绘制新线
```

```
        //        last_item[cur_out][i] = item[cur_out][i];
        //    }
        //
        //}
        // 实时更新消费者所有缓冲区位置
        for (j = 0; j < BUFFER_SIZE; j++) {
            for (i = 0; i < 120; i++) {
                if (last_item[j][i] != item[j][i]) {
                    // 清除旧线，绘制新线
                    line(j * 100, 4 * i, j * 100 + 100, 4 * i, RGB(0, 0, 0));
                    line(j * 100, 4 * i, j * 100 + item[j][i], 4 * i, RGB(255, 20, 147));
                    last_item[j][i] = item[j][i];
                }
            }
        }

       // nanosleep(&req, NULL); // 避免休眠时间弱化优先级，让优先级参与到休眠时间
        ///清空缓冲区
        //for (i = 0; i < 120; i++)
        //{
        //    last_item[cur_out][i] = 0;
        //    //line(cur_out * 100, 4 * i, cur_out * 100 + 100, 4 * i, RGB(0, 0, 0));   // 清
除旧线
        //}
        //绘制优先级(纵坐标各占60)
        // 获取当前优先级（加锁）
        sem_wait(priority_mutex);
        int current_pro_priority = priority_producer;
        int current_con_priority = priority_consumer;
        sem_signal(priority_mutex);

        // 更新生产者优先级显示
        if (current_pro_priority != last_priority_pro) {
            // 清除旧显示
            for (i = 0; i < 127; i++) {
                line(i * 5, 480, i * 5, 480 + 60, RGB(0, 0, 0));
            }
            // 绘制新优先级（红色）
            for (i = 0; i < current_pro_priority; i++) {
                line(i * 5, 480, i * 5, 480 + 60, RGB(255, 0, 0));
            }
            last_priority_pro = current_pro_priority;
        }

        // 更新消费者优先级显示
        if (current_con_priority != last_priority_con) {
```

```
            // 清除旧显示
            for (i = 0; i < 127; i++) {
                line(i * 5, 480 + 60, i * 5, 480 + 120, RGB(0, 0, 0));
            }
            // 绘制新优先级（绿色）
            for (i = 0; i < current_con_priority; i++) {
                line(i * 5, 480 + 60, i * 5, 480 + 120, RGB(0, 255, 0));
            }
            last_priority_con = current_con_priority;
        }


    }
    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(), 0);
     //init_keyboard(); // 需要初始化键盘驱动
     while (1) {
         int key = getchar();
         switch (key)
         {
         case 0x4800:
         {
             sem_wait(priority_mutex);
             if (priority_producer < 117) {
                 priority_producer += 10;
                 setpriority(tid_producer, priority_producer);
             }
             sem_signal(priority_mutex);
             break;
         }
         case 0x5000:
         {
             sem_wait(priority_mutex);
             if (priority_producer > 10) {
                 priority_producer -= 10;
                 setpriority(tid_producer, priority_producer);
             }
             sem_signal(priority_mutex);
             break;
```

```
        }
        case 0x4d00:
        {
            sem_wait(priority_mutex);
            if (priority_consumer< 117) {
                priority_consumer += 10;
                setpriority(tid_consumer, priority_consumer);
            }
            sem_signal(priority_mutex);
            break;
        }
        case 0x4b00:
        {
            sem_wait(priority_mutex);
            if (priority_consumer > 10) {
                priority_consumer -= 10;
                setpriority(tid_consumer, priority_consumer);
            }
            sem_signal(priority_mutex);
            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
    init_semaphores();

    unsigned char* stack_producer;
    unsigned int  stack_size = 1024 * 1024;
    stack_producer = (unsigned char*)malloc(stack_size);
    int tid_p;
    tid_p = task_create(stack_producer + stack_size, &tsk_producer, (void*)0);

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

```
    int tid_consumer;
    tid_consumer = task_create(stack_consumer + stack_size, &tsk_consumer, (void*)0);

    unsigned char* stack_GUI;
    stack_GUI = (unsigned char*)malloc(stack_size);
    int tid_c;
    tid_c = 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);

    if (!stack_producer) {
        printf("Failed to allocate producer stack!\n");
    }
    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。