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 3 月 23 日

<table>
<tr><td>学院</td><td>大数据与软件学院</td><td>年级、专业、班</td><td>2022 级软件工程 2 班</td><td>姓名</td><td>潘铷葳</td><td>成绩</td><td></td></tr>
<tr><td>课程名称</td><td colspan="2">操作系统</td><td>实验项目名　称</td><td colspan="2">线程的创建</td><td>指导教师</td><td>刘寄</td></tr>
<tr><td>教师评语</td><td colspan="7">教师签名：
年　月　日</td></tr>
</table>

一、 实验目的

通过实现一个多线程程序，探索多线程并发执行的能力，以及不同排序算法在并发环境下的表现，来掌握线程的创建。

二、实验内容

随机生成N组非负整数列表，然后创建N个线程，分别用N种不同的排序算法对列表进行排序

- 创建
  - int task_create(void *tos, void (*func)(void *pv), void *pv)
    - tos ： 用户栈的栈顶指针
    - func ： 线程函数
    - pv ： 传递给线程函数func的参数
    - 返回值 ： 大于0，则表示新创建线程之ID
- 退出
  - int task_exit(int code_exit);
    - code_exit ： 线程的退出代码
- 获取线程自己的ID
  - int task_getid();
- 等待线程退出
  - int task_wait(int tid, int *pcode_exit);
    - tid ： 要等待线程之ID
    - pcode_exit ： 如果非NULL，用于保存线程tid的退出代码
- Step1：定义线程函数
- Step2：申请线程栈
  - 线程退出后，才能把用户栈用free释放掉！
- Step3：创建线程

三、实验过程原始记录(数据、图表、计算等)

下面是 main.c 文件的完整代码和编写逻辑：

```c
/*
 * 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
 */
```

__main()函数：

在此函数中，创建了一个大小为 32MB 的堆内存，并使用 TLSF）算法初始化该堆内存。

这个函数主要用于初始化全局变量 g_heap，使得后续的动态内存分配可以使用这个堆内存。

```c
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);
}
```

画线自定义函数和睡眠函数：

drawLine()函数用于在屏幕上绘制一条线段，其参数包括起始点和终止点的坐标、额外的 x 偏移量和颜色。

mySleep()函数用于线程休眠一段时间，通过调用 nanosleep()函数实现。

```c
//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 = 0;
    tim.tv_nsec = 10000000;
    nanosleep(&tim, &tim2);
}
```

排序算法函数：

insertSort()实现了插入排序算法，通过将每个元素逐个插入到已排序的部分中。

selsort()实现了选择排序算法，每次选择未排序部分中的最小元素，并与已排序部分的末尾元素交换。

bubsort()实现了冒泡排序算法，通过不断地交换相邻的元素，将最大（或最小）元素逐步“浮”到数组的顶端。

quicksort()实现了快速排序算法，通过选择一个基准元素，将数组划分为两个子数组，小于基准的放在左边，大于基准的放在右边，然后递归地对子数组进行排序。

```
//第一种排序算法：插入排序
void insertSort(int* data,int n)
    {
        int i,j;

        for(i=1;i<n;i++)
        {
            int temp=data[i];
            for(j=i;j>0&&data[j-1]>temp;j--)
            {
                drawLine(0,j*6+60,data[j],j*6+60,0,RGB(0,0,0));//覆盖原来的线
                data[j]=data[j-1];
                drawLine(0, j * 6 + 60, data[j], j * 6 + 60, 0, 0x4682B4);
            }
            drawLine(0, j * 6 + 60, data[j] , j * 6 + 60,0,RGB(0,0,0));
            data[j]=temp;
            drawLine(0, j * 6 + 60, data[j], j * 6 + 60, 0, 0x4682B4);
            mySleep();
        }
    }

//第二种  选择排序
void selsort(int *a, int n)
{
    int i, j,low,temp;
    for (i = 0; i < n - 1; i++)
    {
        low = i;
        for (j = i; j < n; j++)
        {
            if (a[low] > a[j])
                low = j;
        }
        //进行交换，先抹掉原来线段
        drawLine(0, low * 6 + 60, a[low], low * 6 + 60, 150, RGB(0, 0, 0));
        drawLine(0, i * 6 + 60, a[i], i * 6 + 60, 150, RGB(0, 0, 0));
        temp = a[low];
        a[low] = a[i];
        a[i] = temp;
        drawLine(0, low * 6 + 60, a[low], low * 6 + 60, 150, 0x4169E1);
        drawLine(0, i * 6 + 60, a[i], i * 6 + 60, 150, 0x4169E1);
        mySleep();

    }
}
```

```
//第三种排序，冒泡排序
void bubsort(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 + 60, a[j], j * 6 + 60, 300, RGB(0, 0, 0));
                drawLine(0, (j+1) * 6 + 60, a[j+1], (j+1) * 6 + 60, 300, RGB(0, 0, 0));
                temp = a[j];
                a[j] = a[j + 1];
                a[j + 1] = temp;
                drawLine(0, j * 6 + 60, a[j], j * 6 + 60, 300, 0x556B2F);
                drawLine(0, (j + 1) * 6 + 60, a[j + 1], (j + 1) * 6 + 60, 300, 0x556B2F);
                mySleep();
            }
        }
    }
}
```

```
//第四种排序,快速排序
void quicksort(int *a, int left, int right)
{
    if (left >= right)
        return;
    int j = right, l = left, r = right,temp;
    while (l < r)
    {
        while (a[l] <= a[j] && l < r)l++;
        while (l < r && a[r] >= a[j])r--;
        //swap(a[l], a[r]);
        drawLine(0, l * 6 + 60, a[l], l * 6 + 60, 450, RGB(0, 0, 0));
        drawLine(0, r* 6 + 60, a[r], r * 6 + 60, 450, RGB(0, 0, 0));
        temp = a[l];
        a[l] = a[r];
        a[r] = temp;
        drawLine(0, l * 6 + 60, a[l], l * 6 + 60, 450, 0xFFFF00);
        drawLine(0, r * 6 + 60, a[r], r * 6 + 60, 450, 0xFFFF00);
        mySleep();

    }
    //swap(a[j], a[l]);
    drawLine(0, l * 6 + 60, a[l], l * 6 + 60, 450, RGB(0, 0, 0));
    drawLine(0, j * 6 + 60, a[j], j * 6 + 60, 450, RGB(0, 0, 0));
    temp = a[l];
    a[l] = a[j];
    a[j] = temp;
    drawLine(0, l * 6 + 60, a[l], l * 6 + 60, 450, 0xFFFF00);
    drawLine(0, j * 6 + 60, a[j], j * 6 + 60, 450, 0xFFFF00);
    mySleep();
    quicksort(a, left, l - 1);
    quicksort(a, l + 1, right);
}
```

线程函数：

tsk_foo1()、tsk_foo2()、tsk_foo3()、tsk_foo4()分别是四个线程的执行函数，每个函数会生成一个长度为 150 的随机整数数组，并调用相应的排序算法对数组进行排序。

每个线程在排序前都会先将未排序的数组用不同的颜色绘制在屏幕上，然后调用相应的排序函数对数组进行排序，排序过程中绘制的线段会随着排序的进行而变化。

排序完成后，线程通过调用 task_exit(0)退出。

```
//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()%150;
        printf("%d\n",myCount_1[i]);
    }

    //显示未排序的画面
    for(k=0;k<150;k++)
    {
        drawLine(0,k*6+60,myCount_1[k],k*6+60,0, 0x4682B4);
    }

    mySleep();

    insertSort(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() % 150;
        printf("%d\n", myCount_2[i]);
    }

    //显示未排序的画面
    for (k = 0; k < 150; k++)
    {
        drawLine(0, k * 6 + 60, myCount_2[k], k * 6 + 60,150, 0x4169E1);
    }

    mySleep();
    selsort(myCount_2, 150);
    task_exit(0);
}
```

```c
void tsk_foo3(void* pv)
{
    time_t time(time_t * loc);
    srand(time(NULL));
    int myCount_3[150];
    int i, k;
    for (i = 0; i < 150; i++)
    {
        myCount_3[i] = rand() % 150;
        printf("%d\n", myCount_3[i]);
    }

    //显示未排序的画面
    for (k = 0; k < 150; k++)
    {
        drawLine(0, k * 6 + 60, myCount_3[k], k * 6 + 60, 300, 0x556B2F);
    }

    mySleep();
    bubsort(myCount_3, 150);
    task_exit(0);
}

void tsk_foo4(void* pv)
{
    time_t time(time_t * loc);
    srand(time(NULL));
    int myCount_4[150];
    int i, k;
    for (i = 0; i < 150; i++)
    {
        myCount_4[i] = rand() % 150;
        printf("%d\n", myCount_4[i]);
    }

    //显示未排序的画面
    for (k = 0; k < 150; k++)
    {
        drawLine(0, k * 6 + 60, myCount_4[k], k * 6 + 60, 450, 0xFFFF00);
    }
    mySleep();
    quicksort(myCount_4,0,149);
    task_exit(0);
}
```

main()函数：

main()函数是程序的入口函数，其中初始化了图形界面并创建了四个线程，分别执行四种不同的排序算法。

首先，为每个线程分配了 1MB 的栈空间，并通过调用 task_create()创建了四个线程，每个线程执行一个排序算法。

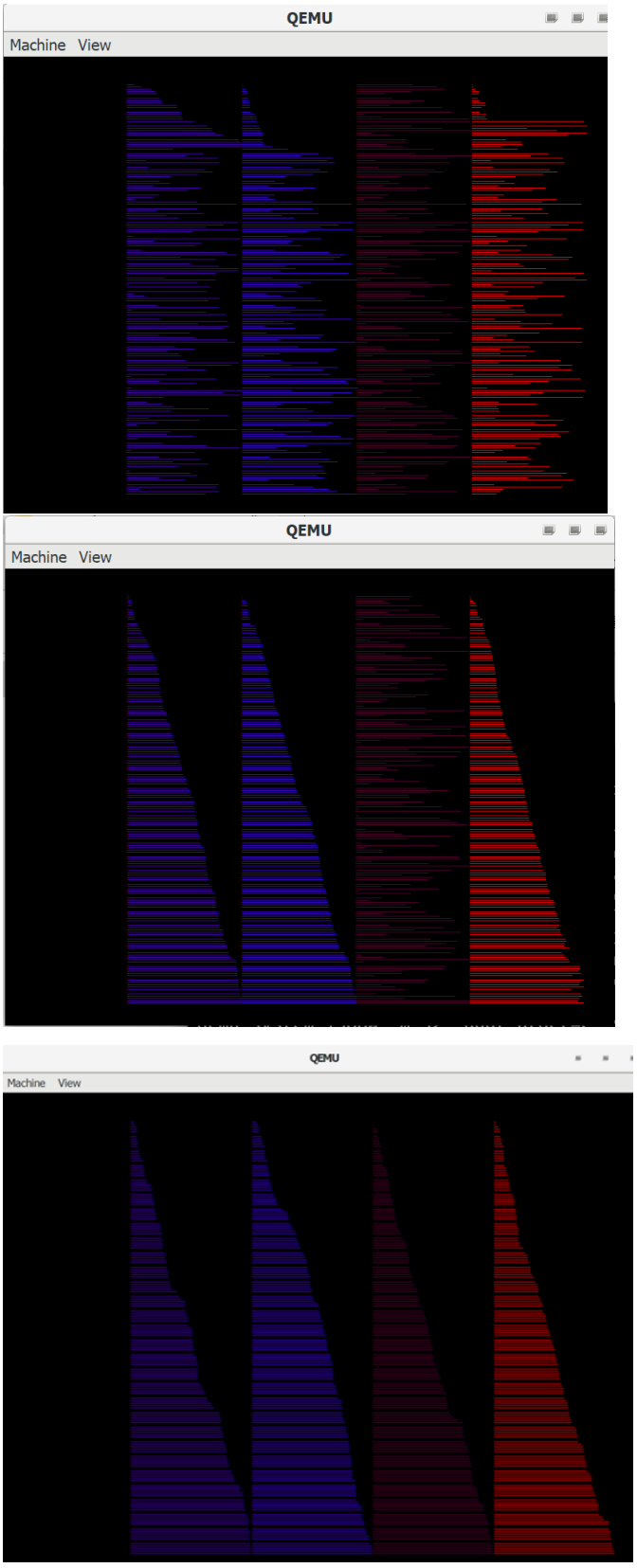
创建完线程后，释放了分配的栈空间，并进入一个死循环，等待所有线程退出。

```
/**
 * 第一个运行在用户模式的线程所执行的函数
 */
void main(void *pv)
{
    unsigned char *stack_foo_1,*stack_foo_2, * stack_foo_3,* stack_foo_4;
    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);
    stack_foo_4 = (unsigned char*)malloc(stack_size);

    init_graphic(0x143);
    task_create(stack_foo_1+stack_size,&tsk_foo1,(void *)0);
    task_create(stack_foo_2 + stack_size, &tsk_foo2, (void*)0);
    task_create(stack_foo_3 + stack_size, &tsk_foo3, (void*)0);
    task_create(stack_foo_4 + stack_size, &tsk_foo4, (void*)0);
    free(stack_foo_1);
    free(stack_foo_2);
    free(stack_foo_3);
    free(stack_foo_4);

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

四、实验结果及分析



在该实验中我们可以观察到动态的排序过程，其中，不同的算法排序时间不同，第二个算法（选择排序）速度最快，其次是第一个算法（插入算法），二者速度相差较小。其次是第四个算法（快速排序），第三个算法（冒泡排序）最慢。