Proj3 Multithreaded Sorting Application & Fork-Join Sorting Application

by 郑航 520021911347

1 Introduction

1.1 Objectives

- 掌握Pthread线程库的基本用法，并实现一个基于Pthread的多线程排序程序
- 掌握java线程库的基本用法，实现两个基于java线程库的多线程排序程序

1.2 Environment

- win10下的VMware Workstation Pro中运行的Ubuntu18.04

2 Project 1—Multithreaded Sorting Application

2.1 Abstract

实现一个基于Pthread的多线程排序程序，功能：接受一串不限长度的int类型的数据，分两半在两个子线程内排序后再归并为一个已排序的result数组，并输出

主要分五部分：

- 接受输入的数据并分段
- 初始化创建三个子线程，分别是两个排序线程和一个归并线程
- 实现快速排序算法
- 实现归并函数
- 结果输出以及资源释放

2.2 接受输入的数据并分段

利用全局的int型数组arr进行存储，一开始的想法是先由用户输入待排序数组的长度len，再依次读取len个数据；但后续觉得这样的方式不够灵活也不符合日常应用要求，用户很可能并不知晓或懒于计算待排序数组的长度，因此更合理的方式是支持不限长度的数据

要实现这个功能，可采用类似循环队列的方法，先创建一个较小的maxlen长度的数组，在数据长度达到maxlen后进行加倍扩容，既可保证灵活性，又可在不浪费过多空间的基础上减少额外的时间开销，代码如下：

```
arr = malloc(sizeof(int) * 10);
int len = 0;
int maxlen = 10;
printf("please input the array to be sorted:\n");
while (scanf("%d", &arr[len]))
{
    len++;
    if (len == maxlen)
    {
        int *tmp = realloc(arr, sizeof(int) * maxlen * 2);
        arr = tmp;
    }
}
```

分段只需分为等长的两部分即可，将index存于两个两位的int数组中，可作为后续新线程的函数参数

```
int arg1[2] = {0, len / 2};
int arg2[2] = {len / 2 + 1, len-1};
```

2.3 初始化创建三个子线程

这部分没有什么难点，只需要学习一下Pthread几个主要函数的用法即可，注意各参数的取地址符&以及对pthread_create返回结果的判断，判断是否创建成功，养成良好的编程习惯

```c
pthread_t sort_thread1, sort_thread2, merge_thread;
pthread_attr_t sort_thread1_attr, sort_thread2_attr, merge_thread_attr;
pthread_attr_init(&sort_thread1_attr);
pthread_attr_init(&sort_thread2_attr);
pthread_attr_init(&merge_thread_attr);

int ret = pthread_create(&sort_thread1, &sort_thread1_attr, sort, arg1);
if (ret == -1)
{
    printf("Create sort_thread1 error!\n");
    return 1;
}
ret = pthread_create(&sort_thread2, &sort_thread2_attr, sort, arg2);
if (ret == -1)
{
    printf("Create sort_thread2 error!\n");
    return 1;
}
ret = pthread_create(&merge_thread, &merge_thread_attr, merge, &len);
if (ret == -1)
{
    printf("Create merge_thread error!\n");
    return 1;
}
```

2.4 实现快速排序

两个子线程中，利用快速排序分别对arr前后两半进行排序

quicksort的时间复杂度，最差为O(n^2)，平均为O(nlgn)，其实现细节在数据结构课程中有进行深入学习，在此不加赘述

```c
int divide(int a[], int low, int high)
{
    int k = a[low];
    do
    {
        while (low < high && a[high] >= k)
            --high;
        if (low < high)
        {
            a[low] = a[high];
            ++low;
        }
        while (low < high && a[low] <= k)
            ++low;
        if (low < high)
        {
            a[high] = a[low];
            --high;
        }
    } while (low != high);
    a[low] = k;
    return low;
}

void quickSort(int a[], int low, int high)
{
    int mid;

    if (low >= high)
        return;
    mid = divide(a, low, high);
    quickSort(a, low, mid - 1);  //排序左一半
    quickSort(a, mid + 1, high); //排序右一半
}

// 包裹函数
void sort(void *arg)
{
    int begin = ((int *)arg)[0];
    int end = ((int *)arg)[1];
    quickSort(arr, begin,end);
}
```

2.4 实现归并函数

merge函数思路比较简单，利用两个指针p1和p2，分别从头至尾遍历arr的前后两部分，每次将两指针所指较小的那个数据放于result中并递增该指针，直到所有数据都转移至result

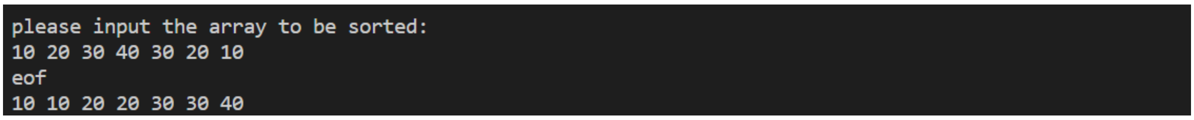
```
void merge(void *arg)
{
    int len = *((int *)arg);
    int p1 = 0, p2 = len / 2 + 1;
    int count = 0;
    while (p1 <= len / 2 && p2 < len)
    {
        if (arr[p1] < arr[p2])
            result[count] = arr[p1++];
        else
            result[count] = arr[p2++];
        count++;
    }
    if (p2 == len && p1 <= len / 2)
        while (count < len)
            result[count++] = arr[p1++];

    else if (p2 < len && p1 > len / 2)
        while (count < len)
            result[count++] = arr[p2++];
}
```

2.5 结果输出以及资源释放

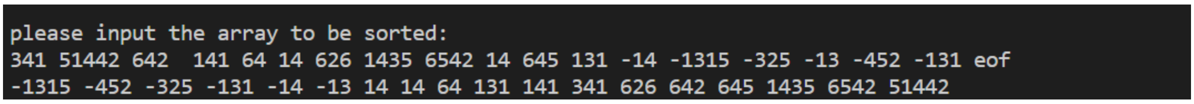
在输出result后，记得释放掉动态申请的堆栈资源arr和result：

```c
for (int i = 0; i < len; i++)
{
    printf("%d ", result[i]);
}
printf("\n");

free(arr);
free(result);
```

2.6 实现结果



```
please input the array to be sorted:
10 20 30 40 30 20 10
eof
10 10 20 20 30 30 40
```



```
please input the array to be sorted:
341 51442 642  141 64 14 626 1435 6542 14 645 131 -14 -1315 -325 -13 -452 -131 eof
-1315 -452 -325 -131 -14 -13 14 14 64 131 141 341 626 642 645 1435 6542 51442
```

2.7 完整代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

void sort(void *arg);
void merge();
int *arr;
int *result;

int main()
{
    arr = malloc(sizeof(int) * 10);
    int len = 0;
    int maxlen = 10;
    printf("please input the array to be sorted:\n");
    while (scanf("%d", &arr[len]))
    {
        len++;
        if (len == maxlen)
        {
            int *tmp = realloc(arr, sizeof(int) * maxlen * 2);
            arr = tmp;
        }
    }

    int arg1[2] = {0, len / 2};
    int arg2[2] = {len / 2 + 1, len - 1};

    pthread_t sort_thread1, sort_thread2, merge_thread;
    pthread_attr_t sort_thread1_attr, sort_thread2_attr, merge_thread_attr;
    pthread_attr_init(&sort_thread1_attr);
    pthread_attr_init(&sort_thread2_attr);
    pthread_attr_init(&merge_thread_attr);

    int ret = pthread_create(&sort_thread1, &sort_thread1_attr, sort, arg1);
    if (ret == -1)
    {
        printf("Create sort_thread1 error!\n");
        return 1;
    }
    ret = pthread_create(&sort_thread2, &sort_thread2_attr, sort, arg2);
    if (ret == -1)
    {
        printf("Create sort_thread2 error!\n");
        return 1;
    }

    pthread_join(sort_thread1, NULL); //阻塞主线程，直到子线程结束才恢复执行
    pthread_join(sort_thread2, NULL);

    result = malloc(sizeof(int) * len);
    ret = pthread_create(&merge_thread, &merge_thread_attr, merge, &len);
    if (ret == -1)
    {
        printf("Create merge_thread error!\n");
        return 1;
    }

    pthread_join(merge_thread, NULL);

    for (int i = 0; i < len; i++)
    {
        printf("%d ", result[i]);
    }
    printf("\n");

    free(arr);
    free(result);
    return 0;
}

//快速排序(o(NlogN))    不稳定排序
int divide(int a[], int low, int high)
{
    int k = a[low];
    do
    {
        while (low < high && a[high] >= k)
            --high;
        if (low < high)
        {
            a[low] = a[high];
            ++low;
        }
        while (low < high && a[low] <= k)
            ++low;
        if (low < high)
        {
            a[high] = a[low];
            --high;
        }
    } while (low != high);
    a[low] = k;
    return low;
}

void quickSort(int a[], int low, int high)
{
    int mid;

    if (low >= high)
        return;
    mid = divide(a, low, high);
    quickSort(a, low, mid - 1);  //排序左一半
    quickSort(a, mid + 1, high); //排序右一半
}

// 包裹函数
void sort(void *arg)
{
    int begin = ((int *)arg)[0];
    int end = ((int *)arg)[1];
    quickSort(arr, begin, end);
}

void merge(void *arg)
{
    int len = *((int *)arg);
    int p1 = 0, p2 = len / 2 + 1;
    int count = 0;
    while (p1 <= len / 2 && p2 < len)
    {
        if (arr[p1] < arr[p2])
            result[count] = arr[p1++];
        else
            result[count] = arr[p2++];
        count++;
    }
    if (p2 == len && p1 <= len / 2)
        while (count < len)
            result[count++] = arr[p1++];

    else if (p2 < len && p1 > len / 2)
        while (count < len)
            result[count++] = arr[p2++];
}
```

3 Project 2—Fork-Join Sorting Application

3.1 Abstract

- 实现两个基于java线程库的多线程排序程序，分别利用快速排序以及归并排序
- 功能：设计了两个模式，mode1是产生给定数量和给定上界的随机数并排序，mode2是接受一串已知长度的int类型的数据并排序
- 排序：两种排序方法都会对目标数组进行切断，当某段长度小于某个threshold后，就对该段执行简单的插入排序或直接选择排序，并将各已排序段进行merge，merge方法与Project 1中相同
- 主要分四部分：
    - 功能选择菜单及数据生成/输入
    - 实例化对象并加入线程池开始执行
    - len<threshold的conquer部分，执行插入排序或直接选择排序
    - len>threshold的divide部分，按快速排序和归并排序的方式分别进行分段，并开启新的子线程进行排序，然后对已排序的部分进行merge

3.2 功能选择菜单及数据生成/输入

- 功能选择菜单主要靠System.out.print输出菜单信息，以及一些简单的读入和条件判断来完成，采用一个int类型的flag来表示mode1/2
- flag=1，表示用户选择自动产生随机数的模式，接受用户输入的数据上界bound后，利用Random类及其成员函数nextInt（bound）来产生随机数
- flag=2，表示用户选择输入自己的待排序数据，直接逐个接受后存入array即可

```java
    Scanner sc = new Scanner(System.in);
    System.out.print("Please input the length of the array to be sorted:\n");
    int len = sc.nextInt();
    Integer[] array = new Integer[len];

    System.out.print("mode1: randomly generate an array of integers automatically\n");
    System.out.print("mode2: input your own array\n");
    System.out.print("please select the mode:(1/2)\n");
    int flag = sc.nextInt();if(flag==1)
    {
        System.out.print(
                "please input the high bound of the randomized integers:(the low bound is 0)\n");
        int bound = sc.nextInt() + 1;
        java.util.Random rand = new java.util.Random();
        for (int i = 0; i < len; i++) {
            array[i] = rand.nextInt(bound);
        }
    }else if(flag==2)
    {
        System.out.print("please input the array:\n");
        for (int i = 0; i < len; i++) {
            array[i] = sc.nextInt();
        }
    }else
    {
        System.out.print("Error:invalid input,exit\n");
        System.exit(1);
    }
```

3.3 实例化对象并加入线程池开始执行

该部分只需要类比课本的java多线程案例SumTask.java即可，实例化对象和线程池，并将对象放入运行

```
    Mergesort task = new Mergesort(0, len - 1, array);
    ForkJoinPool pool = new ForkJoinPool();
    pool.invoke(task);
```

3.4 conquer部分

该部分只需对长度小于threshold的数据段用简单的插入排序或直接选择排序进行排序即可，本次在Mergesort.java和Quicksort.java中分别选择了插入排序insertion_sort和直接选择排序seletion_sort，都作为类的成员函数进行了封装，实现如下：

3.4.1 insertion_sort

```
    if (end - begin < THRESHOLD) {
        // conquer stage
        // using seletion sorting
        this.insertion_sort(begin, end);
    }
```

```
    protected void insertion_sort(int begin, int end) {
        for (int i = begin + 1; i <= end; i++) {
            int key = array[i];
            int j = i - 1;
            while ((j >= begin) && (key < array[j])) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = key;
        }
    }
```

3.4.2 seletion_sort

```
    if (end - begin < THRESHOLD) {
        // conquer stage
        // using seletion sorting
        this.seletion_sort(begin, end);
    }
```

```
protected void seletion_sort(int begin, int end) {
    int min, tmp;
    for (int i = begin; i <= end; i++) {
        min = i;
        for (int j = i + 1; j <= end; ++j)
            if (array[j] < array[min])
                min = j;
        tmp = array[i];
        array[i] = array[min];
        array[min] = tmp;
    }
}
```

3.5 divide部分

3.5.1 Mergesort.java

- 分段方法简单，只需要等分为前后两段后即可分别再放入新线程中进行排序，但是排序后的两段的元素间没有大小关系，因此需要merge为一个段
- merge的方法与project 1中相同

```
// divide stage
int mid = begin + (end - begin) / 2;

Mergesort leftTask = new Mergesort(begin, mid, array);
Mergesort rightTask = new Mergesort(mid + 1, end, array);

leftTask.fork();
rightTask.fork();

rightTask.join();
leftTask.join();

// merge
Integer[] tmparr = new Integer[end - begin + 1];
int p1 = begin, p2 = mid + 1, count = 0;
while(p1<=mid&&p2<=end)
{
    if (array[p1] < array[p2])
        tmparr[count] = array[p1++];
    else
        tmparr[count] = array[p2++];
    count++;
}
if(p1>mid&&p2<=end)
    while (p2 <= end)
        tmparr[count++] = array[p2++];
else if(p1<=mid&&p2>end)
    while (p1 <= mid)
        tmparr[count++] = array[p1++];
for(int i = begin;i<=end;i++)
    array[i]=tmparr[i-begin];
```

3.5.2 Quicksort.java

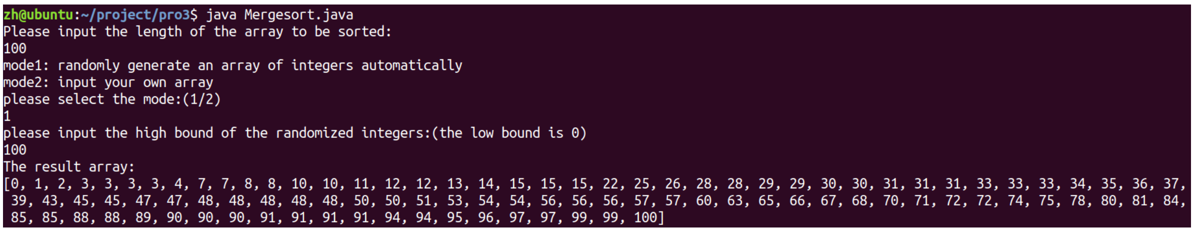
- 分段方法较为复杂，需要选择一个pivot然后将arr以pivot为标准分为前后两段，前一段的每个元素都小于等于pivot，后一段的每个元素都大于等于pivot
- 根据分段方法，前后两段的元素间已经有大小关系，因此无需merge过程

```java
    // divide stage
    // quicksort,pick array[begin] as a pivot
    int pivot = array[begin];
    int low=begin,high=end;
    do
    {
        while (low < high && array[high] >= pivot)
            --high;
        if (low < high) {
            array[low] = array[high];
            ++low;
        }
        while (low < high && array[low] <= pivot)
            ++low;
        if (low < high) {
            array[high] = array[low];
            --high;
        }
    }while(low!=high);
    array[low]=pivot;

    Quicksort leftTask = new Quicksort(begin, low - 1, array);
    Quicksort rightTask = new Quicksort(low + 1, end, array);

    leftTask.fork();
    rightTask.fork();

    rightTask.join();
    leftTask.join();
```

3.6 运行结果

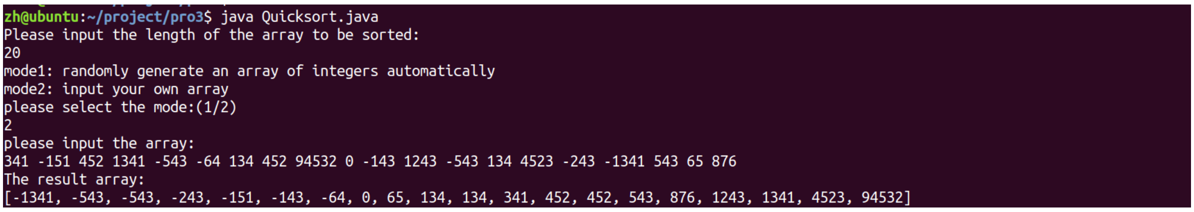
两个程序的输入输出是完全一致的，因此只展示一份运行结果：

mode1 自动产生随机数：



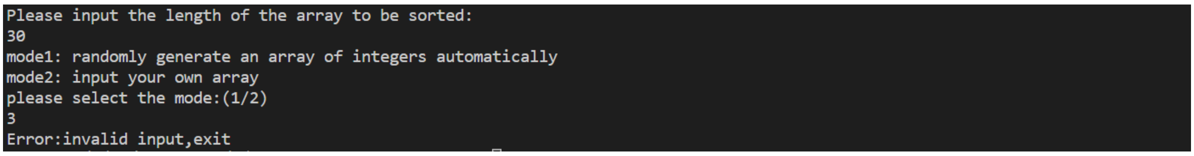
```
zh@ubuntu:~/project/pro3$ java Mergesort.java
Please input the length of the array to be sorted:
100
mode1: randomly generate an array of integers automatically
mode2: input your own array
please select the mode:(1/2)
1
please input the high bound of the randomized integers:(the low bound is 0)
100
The result array:
[0, 1, 2, 3, 3, 3, 3, 4, 7, 7, 8, 8, 10, 10, 11, 12, 12, 13, 14, 15, 15, 15, 22, 25, 26, 28, 28, 29, 29, 30, 30, 31, 31, 31, 33, 33, 33, 34, 35, 36, 37,
 39, 43, 45, 45, 47, 47, 48, 48, 48, 48, 48, 50, 50, 51, 53, 54, 54, 56, 56, 56, 57, 57, 60, 63, 65, 66, 67, 68, 70, 71, 72, 72, 74, 75, 78, 80, 81, 84,
 85, 85, 88, 88, 89, 90, 90, 90, 91, 91, 91, 91, 94, 94, 95, 96, 97, 97, 99, 99, 100]
```

mode2 用户输入待排序数组：



```
zh@ubuntu:~/project/pro3$ java Quicksort.java
Please input the length of the array to be sorted:
20
mode1: randomly generate an array of integers automatically
mode2: input your own array
please select the mode:(1/2)
2
please input the array:
341 -151 452 1341 -543 -64 134 452 94532 0 -143 1243 -543 134 4523 -243 -1341 543 65 876
The result array:
[-1341, -543, -543, -243, -151, -143, -64, 0, 65, 134, 134, 341, 452, 452, 543, 876, 1243, 1341, 4523, 94532]
```

异常处理：



```
Please input the length of the array to be sorted:
30
mode1: randomly generate an array of integers automatically
mode2: input your own array
please select the mode:(1/2)
3
Error:invalid input,exit
```

3.7 完整代码

3.7.1 Mergesort.java

```java
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.*;

public class Mergesort extends RecursiveAction {
    static final int THRESHOLD = 10;

    private int begin;
    private int end;
    private Integer[] array;

    public Mergesort(int begin, int end, Integer[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }

    protected void compute() {
        if (end - begin < THRESHOLD) {
            // conquer stage
            // using insertion sorting
            this.insertion_sort(begin, end);

        } else {
            // divide stage
            int mid = begin + (end - begin) / 2;

            Mergesort leftTask = new Mergesort(begin, mid, array);
            Mergesort rightTask = new Mergesort(mid + 1, end, array);

            leftTask.fork();
            rightTask.fork();

            rightTask.join();
            leftTask.join();

            // merge
            Integer[] tmparr = new Integer[end - begin + 1];
            int p1 = begin, p2 = mid + 1, count = 0;
            while (p1 <= mid && p2 <= end) {
                if (array[p1] < array[p2])
                    tmparr[count] = array[p1++];
                else
                    tmparr[count] = array[p2++];
                count++;
            }
            if (p1 > mid && p2 <= end) {
                while (p2 <= end)
                    tmparr[count++] = array[p2++];

            } else if (p1 <= mid && p2 > end) {
                while (p1 <= mid)
                    tmparr[count++] = array[p1++];
            }
            for (int i = begin; i <= end; i++)
                array[i] = tmparr[i - begin];
        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Please input the length of the array to be sorted:\n");
        int len = sc.nextInt();
        Integer[] array = new Integer[len];

        System.out.print("mode1: randomly generate an array of integers automatically\n");
        System.out.print("mode2: input your own array\n");
        System.out.print("please select the mode:(1/2)\n");
        int flag = sc.nextInt();
        if (flag == 1) {
            System.out.print(
                    "please input the high bound of the randomized integers:(the low bound is 0)\n");
            int bound = sc.nextInt() + 1;
            java.util.Random rand = new java.util.Random();
            for (int i = 0; i < len; i++) {
                array[i] = rand.nextInt(bound);
            }
        } else if (flag == 2) {
            System.out.print("please input the array:\n");
            for (int i = 0; i < len; i++) {
                array[i] = sc.nextInt();
            }
        } else {
            System.out.print("Error:invalid input,exit\n");
            System.exit(1);
        }

        Mergesort task = new Mergesort(0, len - 1, array);
        ForkJoinPool pool = new ForkJoinPool();
        pool.invoke(task);

        System.out.print("The result array:\n");
        System.out.println(Arrays.toString(array));

        sc.close();
    }

    protected void insertion_sort(int begin, int end) {
        for (int i = begin + 1; i <= end; i++) {
            int key = array[i];
            int j = i - 1;
            while ((j >= begin) && (key < array[j])) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = key;
        }
    }
}
```

3.7.2 Quicksort.java

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.*;

public class Quicksort extends RecursiveAction {
    static final int THRESHOLD = 10;

    private int begin;
    private int end;
    private Integer[] array;

    public Quicksort(int begin, int end, Integer[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }

    protected void compute() {
        if (end - begin < THRESHOLD) {
            // conquer stage
            // using seletion sorting
            this.seletion_sort(begin, end);

        } else {
            // divide stage
            // quicksort,pick array[begin] as a pivot
            int pivot = array[begin];
            int low=begin,high=end;
            do {
                while (low < high && array[high] >= pivot)
                    --high;
                if (low < high) {
                    array[low] = array[high];
                    ++low;
                }
                while (low < high && array[low] <= pivot)
                    ++low;
                if (low < high) {
                    array[high] = array[low];
                    --high;
                }
            } while (low != high);
            array[low] = pivot;
```

```java

            Quicksort leftTask = new Quicksort(begin, low-1, array);
            Quicksort rightTask = new Quicksort(low + 1, end, array);

            leftTask.fork();
            rightTask.fork();

            rightTask.join();
            leftTask.join();

        }
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Please input the length of the array to be sorted:\n");
        int len = sc.nextInt();
        Integer[] array = new Integer[len];

        System.out.print("mode1: randomly generate an array of integers automatically\n");
        System.out.print("mode2: input your own array\n");
        System.out.print("please select the mode:(1/2)\n");
        int flag = sc.nextInt();
        if (flag == 1) {
            System.out.print(
                    "please input the high bound of the randomized integers:(the low bound is 0)\n");
            int bound = sc.nextInt() + 1;
            java.util.Random rand = new java.util.Random();
            for (int i = 0; i < len; i++) {
                array[i] = rand.nextInt(bound);
            }
        } else if (flag == 2) {
            System.out.print("please input the array:\n");
            for (int i = 0; i < len; i++) {
                array[i] = sc.nextInt();
            }
        } else {
            System.out.print("Error:invalid input,exit\n");
            System.exit(1);
        }

        Quicksort task = new Quicksort(0, len - 1, array);
        ForkJoinPool pool = new ForkJoinPool();
        pool.invoke(task);

        System.out.print("The result array:\n");
        System.out.println(Arrays.toString(array));

        sc.close();
    }

    protected void seletion_sort(int begin, int end) {
        int min, tmp;
        for (int i = begin; i <= end; i++) {
            min = i;
            for (int j = i + 1; j <= end; ++j)
                if (array[j] < array[min])
                    min = j;
            tmp = array[i];
            array[i] = array[min];
            array[min] = tmp;
        }
    }
}


```

4 Difficulty& Summary

4.1 difficulty

为了运行java程序需要先安装java环境，在Ubuntu上出于某种未知原因，一直apt-get失败，无法安装JDK，故只能现在windows本机安装JDK后完成程序的编写和运行。当程序完成后，再次尝试Ubuntu上安装java环境依然失败，最终采用在windows下载压缩包，然后用WinSCP传到Ubuntu上，并手动解压到合适的目录下的方式，后续还需要配置一些环境变量等，花费了较多不必要的时间

4.2 summary

本次通过实现排序程序的方式，对Pthread和java两个线程库进行了实操运用，对于多线程程序的执行逻辑以及代码细节有了更深入更直观的理解，是对课内理论知识的良好实践

同时，在两个环境下对java的安装和配置也让我对于环境配置有了更深的理解及掌握