

Proj3 Multithreaded Sorting Application & Fork-Join Sorting Application

by 郑航 520021911347

Proj3 Multithreaded Sorting Application & Fork-Join Sorting Application

- 1 Introduction
 - 1.1 Objectives
 - 1.2 Environment
- 2 Project 1—Multithreaded Sorting Application
 - 2.1 Abstract
 - 2.2 接受输入的数据并分段
 - 2.3 初始化创建三个子线程
 - 2.4 实现快速排序
 - 2.4 实现归并函数
 - 2.5 结果输出以及资源释放
 - 2.6 实现结果
 - 2.7 完整代码
- 3 Project 2—Fork-Join Sorting Application
 - 3.1 Abstract
 - 3.2 功能选择菜单及数据生成/输入
 - 3.3 实例化对象并加入线程池开始执行
 - 3.4 conquer部分
 - 3.4.1 insertion_sort
 - 3.4.2 seletion_sort
 - 3.5 divide部分
 - 3.5.1 Mergesort.java
 - 3.5.2 Quicksort.java
 - 3.6 运行结果
 - 3.7 完整代码
 - 3.7.1 Mergesort.java
 - 3.7.2 Quicksort.java
- 4 Difficulty& Summary
 - 4.1 difficulty
 - 4.2 summary

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：

```
1  arr = malloc(sizeof(int) * 10);
2  int len = 0;
3  int maxlen = 10;
4  printf("please input the array to be sorted:\n");
5  while (scanf("%d", &arr[len]))
6  {
7      len++;
8      if (len == maxlen)
9      {
10         int *tmp = realloc(arr, sizeof(int) * maxlen * 2);
11         arr = tmp;
12     }
13 }
```

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

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

```
1 pthread_t sort_thread1, sort_thread2, merge_thread;
2 pthread_attr_t sort_thread1_attr, sort_thread2_attr, merge_thread_attr;
3 pthread_attr_init(&sort_thread1_attr);
4 pthread_attr_init(&sort_thread2_attr);
5 pthread_attr_init(&merge_thread_attr);
6
7 int ret = pthread_create(&sort_thread1, &sort_thread1_attr, sort, arg1);
8 if (ret == -1)
9 {
10     printf("Create sort_thread1 error!\n");
11     return 1;
12 }
13 ret = pthread_create(&sort_thread2, &sort_thread2_attr, sort, arg2);
14 if (ret == -1)
15 {
16     printf("Create sort_thread2 error!\n");
17     return 1;
18 }
19 ret = pthread_create(&merge_thread, &merge_thread_attr, merge, &len);
20 if (ret == -1)
21 {
22     printf("Create merge_thread error!\n");
23     return 1;
24 }
```

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 $O(n\lg n)$ ，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

```
1 int divide(int a[], int low, int high)
2 {
3     int k = a[low];
4     do
5     {
6         while (low < high && a[high] >= k)
7             --high;
8         if (low < high)
9         {
10             a[low] = a[high];
11             ++low;
12         }
13         while (low < high && a[low] <= k)
14             ++low;
15         if (low < high)
16         {
17             a[high] = a[low];
18             --high;
19         }
20     }
```

```

20     } while (low != high);
21     a[low] = k;
22     return low;
23 }
24
25 void quickSort(int a[], int low, int high)
26 {
27     int mid;
28
29     if (low >= high)
30         return;
31     mid = divide(a, low, high);
32     quickSort(a, low, mid - 1); //排序左一半
33     quickSort(a, mid + 1, high); //排序右一半
34 }
35
36 // 包裹函数
37 void sort(void *arg)
38 {
39     int begin = ((int *)arg)[0];
40     int end = ((int *)arg)[1];
41     quickSort(arr, begin, end);
42 }

```

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

```

1 void merge(void *arg)
2 {
3     int len = *((int *)arg);
4     int p1 = 0, p2 = len / 2 + 1;
5     int count = 0;
6     while (p1 <= len / 2 && p2 < len)
7     {
8         if (arr[p1] < arr[p2])
9             result[count] = arr[p1++];
10        else
11            result[count] = arr[p2++];
12        count++;
13    }
14    if (p2 == len && p1 <= len / 2)
15        while (count < len)
16            result[count++] = arr[p1++];
17
18    else if (p2 < len && p1 > len / 2)
19        while (count < len)
20            result[count++] = arr[p2++];
21 }

```

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:

```
1  for (int i = 0; i < len; i++)
2  {
3      printf("%d ", result[i]);
4  }
5  printf("\n");
6
7  free(arr);
8  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

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <pthread.h>
4
5  void sort(void *arg);
6  void merge();
7  int *arr;
8  int *result;
9
10 int main()
11 {
12     arr = malloc(sizeof(int) * 10);
13     int len = 0;
14     int maxlen = 10;
15     printf("please input the array to be sorted:\n");
16     while (scanf("%d", &arr[len]))
17     {
18         len++;
19         if (len == maxlen)
20         {
21             int *tmp = realloc(arr, sizeof(int) * maxlen * 2);
22             arr = tmp;
23         }
24     }
25
26     int arg1[2] = {0, len / 2};
27     int arg2[2] = {len / 2 + 1, len - 1};
28
29     pthread_t sort_thread1, sort_thread2, merge_thread;
30     pthread_attr_t sort_thread1_attr, sort_thread2_attr, merge_thread_attr;
31     pthread_attr_init(&sort_thread1_attr);
32     pthread_attr_init(&sort_thread2_attr);
```

```

33     pthread_attr_init(&merge_thread_attr);
34
35     int ret = pthread_create(&sort_thread1, &sort_thread1_attr, sort,
36                             arg1);
37     if (ret == -1)
38     {
39         printf("Create sort_thread1 error!\n");
40         return 1;
41     }
42     ret = pthread_create(&sort_thread2, &sort_thread2_attr, sort, arg2);
43     if (ret == -1)
44     {
45         printf("Create sort_thread2 error!\n");
46         return 1;
47     }
48     pthread_join(sort_thread1, NULL); //阻塞主线程, 直到子线程结束才恢复执行
49     pthread_join(sort_thread2, NULL);
50
51     result = malloc(sizeof(int) * len);
52     ret = pthread_create(&merge_thread, &merge_thread_attr, merge, &len);
53     if (ret == -1)
54     {
55         printf("Create merge_thread error!\n");
56         return 1;
57     }
58
59     pthread_join(merge_thread, NULL);
60
61     for (int i = 0; i < len; i++)
62     {
63         printf("%d ", result[i]);
64     }
65     printf("\n");
66
67     free(arr);
68     free(result);
69     return 0;
70 }
71
72 //快速排序( $O(N\log N)$ )    不稳定排序
73 int divide(int a[], int low, int high)
74 {
75     int k = a[low];
76     do
77     {
78         while (low < high && a[high] >= k)
79             --high;
80         if (low < high)
81         {
82             a[low] = a[high];
83             ++low;
84         }
85         while (low < high && a[low] <= k)
86             ++low;
87         if (low < high)
88         {
89             a[high] = a[low];

```

```

90         --high;
91     }
92 } while (low != high);
93 a[low] = k;
94 return low;
95 }
96
97 void quickSort(int a[], int low, int high)
98 {
99     int mid;
100
101     if (low >= high)
102         return;
103     mid = divide(a, low, high);
104     quickSort(a, low, mid - 1); //排序左一半
105     quickSort(a, mid + 1, high); //排序右一半
106 }
107
108 // 包裹函数
109 void sort(void *arg)
110 {
111     int begin = ((int *)arg)[0];
112     int end = ((int *)arg)[1];
113     quickSort(arr, begin, end);
114 }
115
116 void merge(void *arg)
117 {
118     int len = *((int *)arg);
119     int p1 = 0, p2 = len / 2 + 1;
120     int count = 0;
121     while (p1 <= len / 2 && p2 < len)
122     {
123         if (arr[p1] < arr[p2])
124             result[count] = arr[p1++];
125         else
126             result[count] = arr[p2++];
127         count++;
128     }
129     if (p2 == len && p1 <= len / 2)
130         while (count < len)
131             result[count++] = arr[p1++];
132
133     else if (p2 < len && p1 > len / 2)
134         while (count < len)
135             result[count++] = arr[p2++];
136 }

```

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

```
1      Scanner sc = new Scanner(System.in);
2      System.out.print("Please input the length of the array to be
sorted:\n");
3      int len = sc.nextInt();
4      Integer[] array = new Integer[len];
5
6      System.out.print("mode1: randomly generate an array of integers
automatically\n");
7      System.out.print("mode2: input your own array\n");
8      System.out.print("please select the mode:(1/2)\n");
9      int flag = sc.nextInt();if(flag==1)
10     {
11         System.out.print(
12             "please input the high bound of the randomized integers:(the
low bound is 0)\n");
13         int bound = sc.nextInt() + 1;
14         java.util.Random rand = new java.util.Random();
15         for (int i = 0; i < len; i++) {
16             array[i] = rand.nextInt(bound);
17         }
18     }else if(flag==2)
19     {
20         System.out.print("please input the array:\n");
21         for (int i = 0; i < len; i++) {
22             array[i] = sc.nextInt();
23         }
24     }else
25     {
26         System.out.print("Error:invalid input,exit\n");
27         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

```
1  Mergesort task = new Mergesort(0, len - 1, array);
2  ForkJoinPool pool = new ForkJoinPool();
3  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

```
1  if (end - begin < THRESHOLD) {
2      // conquer stage
3      // using seletion sorting
4      this.insertion_sort(begin, end);
5  }
```

```
1  protected void insertion_sort(int begin, int end) {
2      for (int i = begin + 1; i <= end; i++) {
3          int key = array[i];
4          int j = i - 1;
5          while ((j >= begin) && (key < array[j])) {
6              array[j + 1] = array[j];
7              j--;
8          }
9          array[j + 1] = key;
10     }
11 }
```

3.4.2 seletion_sort

```
1  if (end - begin < THRESHOLD) {
2      // conquer stage
3      // using seletion sorting
4      this.seletion_sort(begin, end);
5  }
```

```

1      protected void selection_sort(int begin, int end) {
2          int min, tmp;
3          for (int i = begin; i <= end; i++) {
4              min = i;
5              for (int j = i + 1; j <= end; ++j)
6                  if (array[j] < array[min])
7                      min = j;
8              tmp = array[i];
9              array[i] = array[min];
10             array[min] = tmp;
11         }
12     }

```

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

```

1      // divide stage
2      int mid = begin + (end - begin) / 2;
3
4      Mergesort leftTask = new Mergesort(begin, mid, array);
5      Mergesort rightTask = new Mergesort(mid + 1, end, array);
6
7      leftTask.fork();
8      rightTask.fork();
9
10     rightTask.join();
11     leftTask.join();
12
13     // merge
14     Integer[] tmparr = new Integer[end - begin + 1];
15     int p1 = begin, p2 = mid + 1, count = 0;
16     while(p1<=mid&& p2<=end)
17     {
18         if (array[p1] < array[p2])
19             tmparr[count] = array[p1++];
20         else
21             tmparr[count] = array[p2++];
22         count++;
23     }
24     if(p1>mid&&p2<=end)
25         while (p2 <= end)
26             tmparr[count++] = array[p2++];
27     else if(p1<=mid&&p2>end)
28         while (p1 <= mid)
29             tmparr[count++] = array[p1++];
30     for(int i = begin; i<=end; i++)
31         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

```
1 // divide stage
2 // quicksort,pick array[begin] as a pivot
3 int pivot = array[begin];
4 int low=begin,high=end;
5 do
6 {
7     while (low < high && array[high] >= pivot)
8         --high;
9     if (low < high) {
10         array[low] = array[high];
11         ++low;
12     }
13     while (low < high && array[low] <= pivot)
14         ++low;
15     if (low < high) {
16         array[high] = array[low];
17         --high;
18     }
19 }while(low!=high);
20 array[low]=pivot;
21
22 Quicksort leftTask = new Quicksort(begin, low - 1, array);
23 Quicksort rightTask = new Quicksort(low + 1, end, array);
24
25 leftTask.fork();
26 rightTask.fork();
27
28 rightTask.join();
29 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, 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, 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

```
1  import java.util.Arrays;
2  import java.util.Scanner;
3  import java.util.concurrent.*;
4
5  public class Mergesort extends RecursiveAction {
6      static final int THRESHOLD = 10;
7
8      private int begin;
9      private int end;
10     private Integer[] array;
11
12     public Mergesort(int begin, int end, Integer[] array) {
13         this.begin = begin;
14         this.end = end;
15         this.array = array;
16     }
17
18     protected void compute() {
19         if (end - begin < THRESHOLD) {
20             // conquer stage
21             // using insertion sorting
22             this.insertion_sort(begin, end);
23
24         } else {
25             // divide stage
26             int mid = begin + (end - begin) / 2;
27
28             Mergesort leftTask = new Mergesort(begin, mid, array);
29             Mergesort rightTask = new Mergesort(mid + 1, end, array);
30
31             leftTask.fork();
32             rightTask.fork();
33
34             rightTask.join();
35             leftTask.join();
36
37             // merge
38             Integer[] tmparr = new Integer[end - begin + 1];
39             int p1 = begin, p2 = mid + 1, count = 0;
40             while (p1 <= mid && p2 <= end) {
41                 if (array[p1] < array[p2])
42                     tmparr[count] = array[p1++];
43                 else
44                     tmparr[count] = array[p2++];
45                 count++;
46             }
47             // copy back to array
48             for (int i = begin; i <= end; i++)
49                 array[i] = tmparr[i - begin];
50         }
51     }
52
53     private void insertion_sort(int begin, int end) {
54         for (int i = begin + 1; i <= end; i++)
55             insert(array[i], i);
56     }
57
58     private void insert(Integer value, int index) {
59         for (int i = index; i > begin; i--)
60             array[i] = array[i - 1];
61         array[index] = value;
62     }
63 }
```

```

46         }
47         if (p1 > mid && p2 <= end) {
48             while (p2 <= end)
49                 tmparr[count++] = array[p2++];
50
51         } else if (p1 <= mid && p2 > end) {
52             while (p1 <= mid)
53                 tmparr[count++] = array[p1++];
54         }
55         for (int i = begin; i <= end; i++)
56             array[i] = tmparr[i - begin];
57     }
58 }
59
60 public static void main(String[] args) {
61     Scanner sc = new Scanner(System.in);
62     System.out.print("Please input the length of the array to be
sorted:\n");
63     int len = sc.nextInt();
64     Integer[] array = new Integer[len];
65
66     System.out.print("mode1: randomly generate an array of integers
automatically\n");
67     System.out.print("mode2: input your own array\n");
68     System.out.print("please select the mode:(1/2)\n");
69     int flag = sc.nextInt();
70     if (flag == 1) {
71         System.out.print(
72             "please input the high bound of the randomized
integers:(the low bound is 0)\n");
73         int bound = sc.nextInt() + 1;
74         java.util.Random rand = new java.util.Random();
75         for (int i = 0; i < len; i++) {
76             array[i] = rand.nextInt(bound);
77         }
78     } else if (flag == 2) {
79         System.out.print("please input the array:\n");
80         for (int i = 0; i < len; i++) {
81             array[i] = sc.nextInt();
82         }
83     } else {
84         System.out.print("Error:invalid input,exit\n");
85         System.exit(1);
86     }
87
88     Mergesort task = new Mergesort(0, len - 1, array);
89     ForkJoinPool pool = new ForkJoinPool();
90     pool.invoke(task);
91
92     System.out.print("The result array:\n");
93     System.out.println(Arrays.toString(array));
94
95     sc.close();
96 }
97
98 protected void insertion_sort(int begin, int end) {
99     for (int i = begin + 1; i <= end; i++) {
100         int key = array[i];

```

```

101         int j = i - 1;
102         while ((j >= begin) && (key < array[j])) {
103             array[j + 1] = array[j];
104             j--;
105         }
106         array[j + 1] = key;
107     }
108 }
109 }

```

3.7.2 Quicksort.java

```

1  import java.util.Arrays;
2  import java.util.Scanner;
3  import java.util.concurrent.*;
4
5  public class Quicksort extends RecursiveAction {
6      static final int THRESHOLD = 10;
7
8      private int begin;
9      private int end;
10     private Integer[] array;
11
12     public Quicksort(int begin, int end, Integer[] array) {
13         this.begin = begin;
14         this.end = end;
15         this.array = array;
16     }
17
18     protected void compute() {
19         if (end - begin < THRESHOLD) {
20             // conquer stage
21             // using selection sorting
22             this.selection_sort(begin, end);
23
24         } else {
25             // divide stage
26             // quicksort, pick array[begin] as a pivot
27             int pivot = array[begin];
28             int low = begin, high = end;
29             do {
30                 while (low < high && array[high] >= pivot)
31                     --high;
32                 if (low < high) {
33                     array[low] = array[high];
34                     ++low;
35                 }
36                 while (low < high && array[low] <= pivot)
37                     ++low;
38                 if (low < high) {
39                     array[high] = array[low];
40                     --high;
41                 }
42             } while (low != high);
43             array[low] = pivot;

```

```

44
45         quicksort leftTask = new Quicksort(begin, low-1, array);
46         quicksort rightTask = new Quicksort(low + 1, end, array);
47
48         leftTask.fork();
49         rightTask.fork();
50
51         rightTask.join();
52         leftTask.join();
53
54     }
55 }
56
57 public static void main(String[] args) {
58     Scanner sc = new Scanner(System.in);
59     System.out.print("Please input the length of the array to be
sorted:\n");
60     int len = sc.nextInt();
61     Integer[] array = new Integer[len];
62
63     System.out.print("mode1: randomly generate an array of integers
automatically\n");
64     System.out.print("mode2: input your own array\n");
65     System.out.print("please select the mode:(1/2)\n");
66     int flag = sc.nextInt();
67     if (flag == 1) {
68         System.out.print(
69             "please input the high bound of the randomized
integers:(the low bound is 0)\n");
70         int bound = sc.nextInt() + 1;
71         java.util.Random rand = new java.util.Random();
72         for (int i = 0; i < len; i++) {
73             array[i] = rand.nextInt(bound);
74         }
75     } else if (flag == 2) {
76         System.out.print("please input the array:\n");
77         for (int i = 0; i < len; i++) {
78             array[i] = sc.nextInt();
79         }
80     } else {
81         System.out.print("Error:invalid input,exit\n");
82         System.exit(1);
83     }
84
85     Quicksort task = new Quicksort(0, len - 1, array);
86     ForkJoinPool pool = new ForkJoinPool();
87     pool.invoke(task);
88
89     System.out.print("The result array:\n");
90     System.out.println(Arrays.toString(array));
91
92     sc.close();
93 }
94
95 protected void selection_sort(int begin, int end) {
96     int min, tmp;
97     for (int i = begin; i <= end; i++) {
98         min = i;

```

```
99         for (int j = i + 1; j <= end; ++j)
100             if (array[j] < array[min])
101                 min = j;
102         tmp = array[i];
103         array[i] = array[min];
104         array[min] = tmp;
105     }
106 }
107 }
108
109
```

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