操作系统 project-3 实验报告

* 姓名: 管仁阳　学号:519021911058　邮箱: guanrenyang@sjtu.edu.cn

1 实验名称

1. Multithreaded Sorting Application
2. Fork-Join Sorting Application

2 实验目的

2.1 Multithreaded Sorting Application:

创建一个多线程程序满足以下要求

1. 一个整数列表被分为两个相等长度的子列
2. 使用独立的线程使用任意排序算法排序这两个子列
3. 使用另一个独立的线程归并这两个子列的结果

2.2 Fork-Join Sorting Application:

使用 Java 的 fork-join 并行化 API 来实现快速排序和归并排序。

3 预备知识

3.1 Multithreaded Sorting Application:

3.1.1 pthread 线程创建

pthread_t 为线程标识符，*pthread_attr_t* 为线程参数。
pthread_attr_init() 函数初始化传入线程的参数
pthread_create() 函数的第一个参数为新创建的线程标识符，第二个参数为默认参数，第三个参数为子线程的工作函数名，第四个参数为传递给 runner() 函数的参数。

代码示例如下所示

```
1 pthread_t tid; /* the thread identifier */
2 pthread_attr_t attr; /* set of attributes for the thread */
3 pthread_attr_init(&attr);/* get the default attributes */
4 pthread_create(&tid,&attr,runner,argv[1]);/* create the thread */
```

3.1.2 pthread 线程间参数传递

自定义结构体 parameter 为传递的参数类型，runner_sort() 函数线程 tid_former 要执行的函数，结构体实例的指针作为第四个参数传递给 pthread_create() 函数。*注意: runner 接受参数为 void* 型，要强制转化为 parameter* 型*

```c
/*传递给ruuner函数的参数类型*/
struct parameter{
    int Start;
    int End;
};
/*runner 函数原型*/
void* runner_sort(void* para);
/*pthread_create() 函数调用方法*/
struct parameter* para_sort_former=(struct parameter*) malloc(sizeof(struct p
pthread_create(&tid_former,NULL,runner_sort,para_sort_former);
```

3.1.3 pthread 线程同步

在主线程中等待标识符为 tid 的子线程结束运行，之后主线程继续运行。

```c
/* wait for the thread to exit */
pthread_join(tid,NULL);
```

3.2 Fork-Join Sorting Application:

3.2.1 ForkJoinPool

Java 中的线程池。专门用来将大的任务拆分成小任务来完成 (fork)，并将小任务的结果进行合并 (join). invoke() 函数将任务提交到 pool。ForkJoinPool 会分配一个线程来执行 compute() 函数的任务。

```c
ForkJoinPool pool = new ForkJoinPool();
pool.invoke(task);
```

3.2.2 RecursiveAction

ForkJoinTask 的子类，是没有返回值的任务。

```c
public class Mergesort extends RecursiveAction {}
```

3.2.3 compute() 方法

compute() 方法为每一个被 invoke 到 ForkJoinPool 的任务要执行的方法。

4 实验内容

4.1 Multithreaded Sorting Application:

```c
# include <stdio.h>
# include <pthread.h>
# include <stdlib.h>
int *array;
int *result;
struct parameter
{
    int Start;
```

```
9       int End;
10  };
11  void* runner_sort(void* para){
12      struct parameter* para_sort=(struct parameter*) para;
13      if(para_sort->Start>=para_sort->End)
14          return NULL;
15      for(int i=para_sort->End;i>para_sort->Start;--i){
16          for(int j=para_sort->Start;j<i;++j){
17              if(array[j]>array[j+1]){
18                  int tmp=array[j];
19                  array[j]=array[j+1];
20                  array[j+1]=tmp;
21              }
22          }
23      }
24
25  }
26  void* runner_merge(void* para){
27
28      struct parameter* para_merge=(struct parameter*) para;
29
30      int n=para_merge->End+1;
31
32      int end_former=n/2-1;
33      int end_later=n-1;
34
35      int i=0,j=end_former+1;
36      int index_result=0;
37      while (i<=end_former&&j<=end_later)
38      {
39          if(array[i]<array[j]){
40              result[index_result]=array[i];
41              i++;
42          }
43          else{
44              result[index_result]=array[j];
45              j++;
46          }
47          index_result++;
48      }
49      if(i<=end_former){
50          for(i=i;i<=n/2-1;++i){
51              result[index_result]=array[i];
52              index_result++;
53          }
54      }
55      else if(j<=end_later){
56          for(j=j;j<n-1;++j){
57              result[index_result]=array[j];
58              index_result++;
```

```
59          }
60      }
61  }
62  int main(){
63      printf("Please input the length of your unsorted array:");
64      int n;
65      scanf("%d",&n);
66
67      array=(int*) malloc(n*sizeof(int));
68      result=(int*) malloc(n*sizeof(int));
69
70      printf("Please input the elements to sort:\n");
71      for(int i=0;i<n;++i){
72          scanf("%d",&array[i]);
73      }
74
75      pthread_t tid_former;//thread for sorting the former half
76      pthread_t tid_later; //thread for sorting the later half
77      struct parameter* para_sort_former=malloc(sizeof(struct parameter));
78      struct parameter* para_sort_later=malloc(sizeof(struct parameter));
79
80      para_sort_former->Start=0;
81      para_sort_former->End=n/2-1;
82      para_sort_later->Start=n/2;
83      para_sort_later->End=n-1;
84
85      int ERROR[2];
86      ERROR[0]=pthread_create(&tid_former,NULL,runner_sort,para_sort_former);
87      ERROR[1]=pthread_create(&tid_later,NULL,runner_sort,para_sort_later);
88
89      pthread_join(tid_former,NULL);
90      pthread_join(tid_later,NULL);
91
92      pthread_t tid_merge;
93
94      struct parameter* para_merge=malloc(sizeof(struct parameter));
95      para_merge->Start=0;
96      para_merge->End=n-1;
97
98      int ERROR_MERGE;
99      ERROR_MERGE=pthread_create(&tid_merge,NULL,runner_merge,para_merge);
100         if(ERROR_MERGE){
101             printf("Fail to create the merging thread!\n");
102             exit(1);
103         }
104     pthread_join(tid_merge,NULL);
105     printf("Array after sort:\n");
106     for(int i=0;i<n;++i){
107         printf("%d ",result[i]);
108     }
```

```
109     return 0;
110 }
```

4.2 Fork-Join Sorting Application:

4.2.1 MergeSort

```
1  import java.util.Arrays;
2  import java.util.Scanner;
3  import java.util.concurrent.*;
4
5  public class MergeSort extends RecursiveAction {
6  private int begin;
7  private int end;
8  private Integer[] array;
9
10 public MergeSort(int begin, int end, Integer[] array) {
11 this.begin = begin;
12 this.end = end;
13 this.array = array;
14 }
15
16 protected void compute() {
17 //Bubble sort
18 if (end - begin + 1 <= 10) {
19 for (int i = end; i >= begin + 1; -- i)
20 for (int j = begin; j < i; ++ j)
21         if (array[j].compareTo(array[j + 1]) > 0) {
22                 Integer temp = array[j];
23                 array[j] = array[j + 1];
24                 array[j + 1] = temp;
25         }
26 } else {
27 int mid = begin + (end - begin) / 2;
28
29 MergeSort leftTask = new MergeSort(begin, mid, array);
30 MergeSort rightTask = new MergeSort(mid + 1, end, array);
31
32 leftTask.fork();
33 rightTask.fork();
34
35 leftTask.join();
36 rightTask.join();
37
38 Integer[] temp = new Integer [end - begin + 1];
39
40 int pos1 = begin, pos2 = mid + 1, k = 0;
41 while (pos1 <= mid && pos2 <= end) {
42 if (array[pos1].compareTo(array[pos2]) <= 0) temp[k ++] = array[pos1 ++];
43 else temp[k ++] = array[pos2 ++];
```

```
}
while (pos1 <= mid) temp[k ++] = array[pos1 ++];
while (pos2 <= end) temp[k ++] = array[pos2 ++];

for (int i = 0; i < k; ++ i)
array[i + begin] = temp[i];
}
}

public static void main(String[] args) {
ForkJoinPool pool = new ForkJoinPool();
Scanner sc = new Scanner(System.in);

System.out.println("Please input the number of elements:");
int n=sc.nextInt();
Integer[] array = new Integer[n];
for (int i = 0; i < n; ++ i)
{
array[i]=sc.nextInt();
}

System.out.println("Before sorting:");
System.out.println(Arrays.toString(array));


MergeSort task = new MergeSort(0, n-1, array);

pool.invoke(task);

System.out.println("After sorting:");
System.out.println(Arrays.toString(array));
}
}
```

4.2.2 QuickSort

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

public class QuickSort extends RecursiveAction {

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;
```

```
15 }
16
17 protected void compute() {
18 if (end - begin + 1 <= 10) {
19 for (int i = end; i >= begin + 1; -- i)
20 for (int j = begin; j < i; ++ j)
21 if (array[j].compareTo(array[j + 1]) > 0) {
22         Integer temp = array[j];
23         array[j] = array[j + 1];
24         array[j + 1] = temp;
25 }
26 } else {
27 Integer pivot = array[begin];
28 int low = begin, high = end;
29 while (low < high) {
30 while (low < high && array[high].compareTo(pivot) >= 0) -- high;
31 if (low < high) array[low ++] = array[high];
32 while (low < high && array[low].compareTo(pivot) <= 0) ++ low;
33 if (low < high) array[high --] = array[low];
34 }
35 array[low] = pivot;
36
37 QuickSort leftTask = new QuickSort(begin, low - 1, array);
38 QuickSort rightTask = new QuickSort(low + 1, end, array);
39
40 leftTask.fork();
41 rightTask.fork();
42
43 leftTask.join();
44 rightTask.join();
45 }
46 }
47
48 public static void main(String[] args) {
49 ForkJoinPool pool = new ForkJoinPool();
50 Scanner sc = new Scanner(System.in);
51
52 System.out.println("Please input the number of elements:");
53 int n=sc.nextInt();
54 Integer[] array = new Integer[n];
55 for (int i = 0; i < n; ++ i)
56 {
57 array[i]=sc.nextInt();
58 }
59
60 System.out.println("Before sorting:");
61 System.out.println(Arrays.toString(array));
62
63 // use fork-join parallelism to sum the array
64 QuickSort task = new QuickSort(0, n-1, array);
```

```
65
66 pool.invoke(task);
67 System.out.println("After sorting:");
68 System.out.println(Arrays.toString(array));
69 }
70 }
```

5 实验结果

Multithreaded Sorting Application & Fork-Join Sorting Application :
图 1中依次为 Multithreaded Sorting Application 排序结果、MergeSort 排序结果和 QuickSort 排序结果。

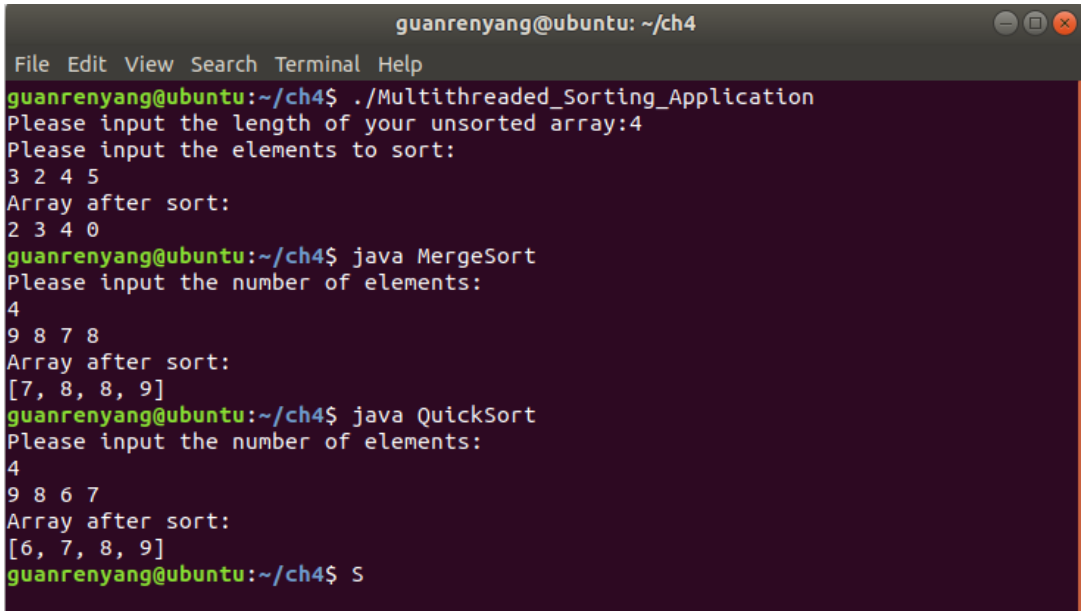

图 1: Result

6 总结与思考

6.1 Multithreaded Sorting Application

通过此次实验我学会了如何使用 pthread API 创建线程、同步线程。此外，我还学会了线程间传递参数的方法。在多线程变成中，使用 gdb 等调试工具调试并行的线程并不十分方便，因此**对多线程操作函数的返回值进行异常处理，使用 stderr 来打印错误信息**对于多线程程序的 debug 来说尤其重要。

6.2 Fork-Join Sorting Application

通过此次实验我学会了如何使用 java 的 ForkJoinPool 来进行多线程排序。Java 的 ForkJoinPool 不需要使用者了解层操作，甚至将每一个子线程封装成了一个类，创建子线程只需要进行类的实例化。这也引发了我对不同编程语言的异同有了更深的理解与思考。