

上海交通大學

SHANGHAI JIAOTONG UNIVERSITY



操作系统课程设计报告 - Project3  
Multithreaded Sorting Application &  
Fork-Join Sorting Application

姓名：薛春宇

学号：518021910698

完成时间：2020/5/29

## 一、实验目的

1. 通过实现多线程排序算法，加深对多线程编程的理解和使用；
2. 通过在 Java 语言下实现多线程下的归并、快速排序，掌握 Java 语言中多线程的相关原理和操作。

## 二、实验内容

1. 用 C 语言实现一个多线程的归并排序操作，即通过创建新的线程来实现分治，再通过合并线程来实现归并；
2. 用 Java 分别实现快速排序和归并排序的两个多线程版本。

## 三、内容一：Multithreaded Sorting Application

### 1. 实验原理：

该部分的核心思想是分治法，因此我们选用最有代表性的归并排序来实现这个项目。同时，分治的核心部分在于引入了<pthread.h>头文件之后，通过以下函数来实现多线程操作：

- (1) *pthread\_attr\_t*: 用来初始化线程属性；
- (2) *pthread\_create()*: 用来根据几个参数创建相应的线程；
- (3) *pthread\_join()*: 用来将已经完成的线程合并回主线程中；

正式通过这些创建的多线程，我们实现了分治法中最为重要的分治步骤。同时，在将线程合并之后，我们会再次调用自己定义的 *merge()* 函数来完成整个任务的合并，进而将分治操作完成。

具体到用户交互层面，我们实现了一个有选择性的输入方式。你可以选择依据提示输入“Y”，并自定义数组的大小，之后我们的程序会迭代调用随机函数，生成随机输入并将排序好的结果输出在命令行中。我们利用宏定义设置了迭代次数为 500000 次，由于 C 语言中的 `srand()` + `rand()` 函数组合生成随机数组的形式每隔一秒钟才会刷新一次，这样多次的迭代次数是合理的。

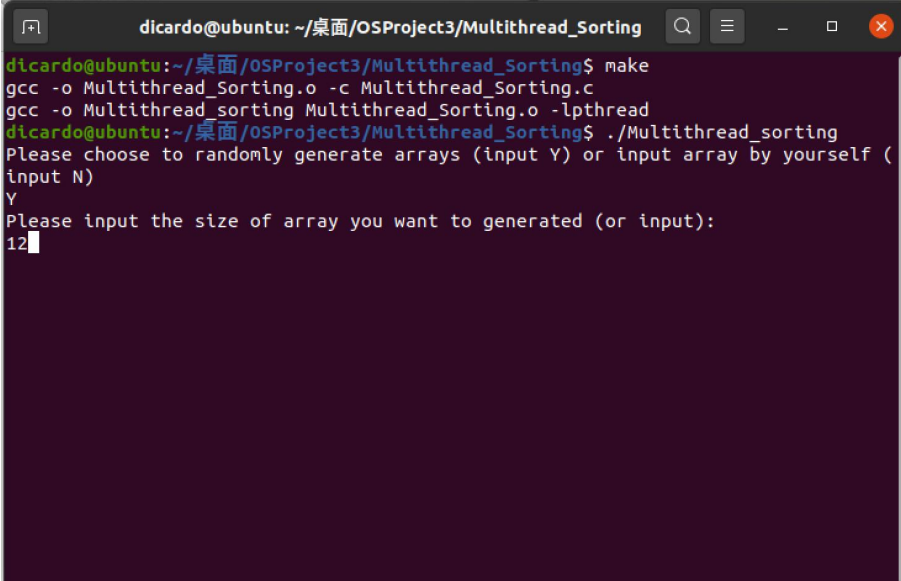
同时，我们会对每一次的排序结果进行检测，若正确则会输出“Success! ”，否则会输出“Failed! ”并直接终止整个程序。

如果你输入的是“N”，则你可以在自定义数组大小之后，输入自己想要排序的序列。排序结果也会被输出在命令行中。

## 2. 实验代码：见附件/Part A/Multithread\_Sorting.c

## 3. 实验结果截图：

### (1) 随机生成输入：



```
dicardo@ubuntu: ~/桌面/OSProject3/Multithread_Sorting
dicardo@ubuntu:~/桌面/OSProject3/Multithread_Sorting$ make
gcc -o Multithread_Sorting.o -c Multithread_Sorting.c
gcc -o Multithread_sorting Multithread_Sorting.o -lpthread
dicardo@ubuntu:~/桌面/OSProject3/Multithread_Sorting$ ./Multithread_sorting
Please choose to randomly generate arrays (input Y) or input array by yourself (
input N)
Y
Please input the size of array you want to generated (or input):
12
```

```
dicardo@ubuntu: ~/桌面/OSProject3/Multithread_Sorting
The sorted array is:
-446 -404 -340 -166 -80 6 28 96 146 176 210 272
-----Round 499996 Success!-----
The initial array is:
-404 -166 272 -80 146 210 96 28 6 176 -446 -340
The sorted array is:
-446 -404 -340 -166 -80 6 28 96 146 176 210 272
-----Round 499997 Success!-----
The initial array is:
-404 -166 272 -80 146 210 96 28 6 176 -446 -340
The sorted array is:
-446 -404 -340 -166 -80 6 28 96 146 176 210 272
-----Round 499998 Success!-----
The initial array is:
-404 -166 272 -80 146 210 96 28 6 176 -446 -340
The sorted array is:
-446 -404 -340 -166 -80 6 28 96 146 176 210 272
-----Round 499999 Success!-----
The initial array is:
-404 -166 272 -80 146 210 96 28 6 176 -446 -340
The sorted array is:
-446 -404 -340 -166 -80 6 28 96 146 176 210 272
-----Round 500000 Success!-----
dicardo@ubuntu:~/桌面/OSProject3/Multithread_Sorting$
```

(2) 自定义输入:

```
dicardo@ubuntu:~/桌面/OSProject3/Multithread_Sorting$ ./Multithread_sorting
Please choose to randomly generate arrays (input Y) or input array by yourself (
input N)
N
Please input the size of array you want to generated (or input):
6
Please input the array:
-5 0 3 9 21142 -15434
The sorted array is:
-15434 -5 0 3 9 21142
-----Success!-----
dicardo@ubuntu:~/桌面/OSProject3/Multithread_Sorting$
```

## 四、内容二：Fork-Join Sorting Application

### 1. 实验原理:

该部分利用了 Java 语言中的一些多线程操作函数来实现快速排序、归并排序中的分治思想。以归并排序为例，我们在 Java 中定义了一个大的结构体 *MergeSort*，进而在需要分治的时候分别设置两个子任务结构体，并调用 *subtask.fork()* 和 *subtask.join()* 这两个函数来实现子任务（子线程）的并行。

在用户交互层面，我们也实现了选择性的输入方式，即“Y”可以随机迭代生成数组并检测排序结果是否正确，“N”可以输入用户自己的待排序序列。值得注意的是，Java 中的随机函数工具效果比 C 中要好，可以做到每一次模

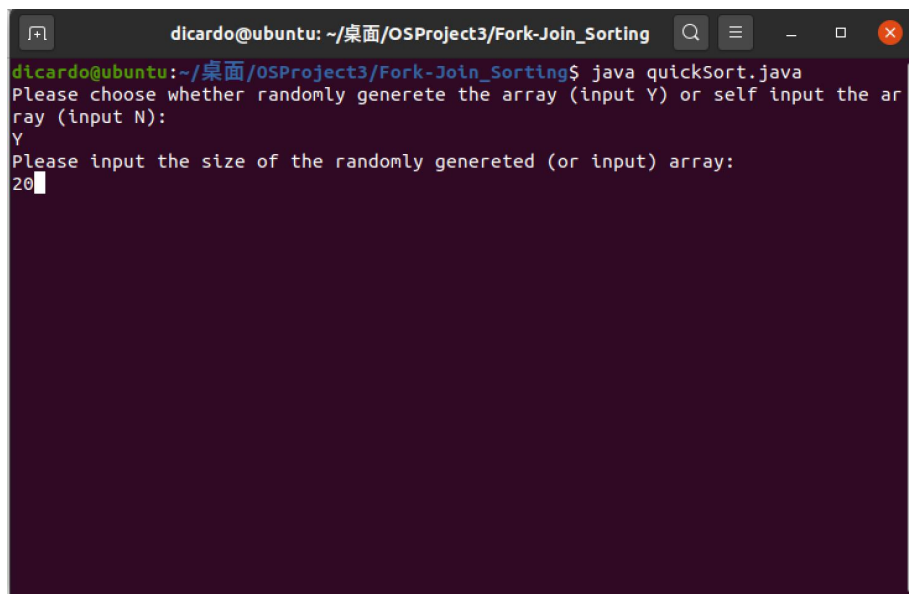
拟都会产生不同的序列，而不是每一秒钟才会变化一次，因此这里我们设置的迭代次数为 100000 次。

快速排序的实现与归并排序相似。

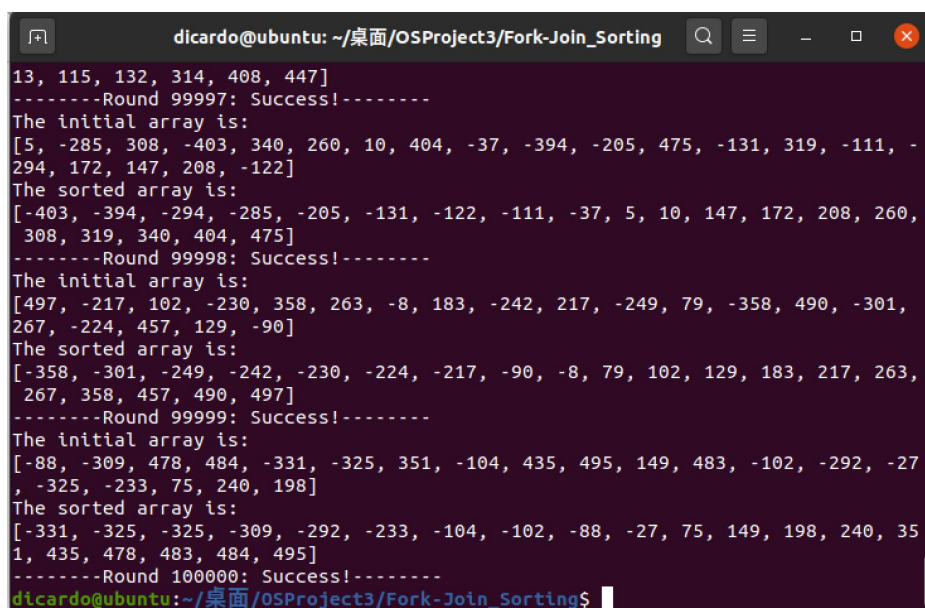
2. 实验代码：见附件/Part B/mergeSort.java & quickSort.java

3. 实验结果截图：

(1) 快速排序：



```
dicardo@ubuntu: ~/桌面/OSProject3/Fork-Join_Sorting
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$ java quickSort.java
Please choose whether randomly generate the array (input Y) or self input the array (input N):
Y
Please input the size of the randomly generated (or input) array:
20
```



```
dicardo@ubuntu: ~/桌面/OSProject3/Fork-Join_Sorting
13, 115, 132, 314, 408, 447]
-----Round 99997: Success!-----
The initial array is:
[5, -285, 308, -403, 340, 260, 10, 404, -37, -394, -205, 475, -131, 319, -111, -294, 172, 147, 208, -122]
The sorted array is:
[-403, -394, -294, -285, -205, -131, -122, -111, -37, 5, 10, 147, 172, 208, 260, 308, 319, 340, 404, 475]
-----Round 99998: Success!-----
The initial array is:
[497, -217, 102, -230, 358, 263, -8, 183, -242, 217, -249, 79, -358, 490, -301, 267, -224, 457, 129, -90]
The sorted array is:
[-358, -301, -249, -242, -230, -224, -217, -90, -8, 79, 102, 129, 183, 217, 263, 267, 358, 457, 490, 497]
-----Round 99999: Success!-----
The initial array is:
[-88, -309, 478, 484, -331, -325, 351, -104, 435, 495, 149, 483, -102, -292, -27, -325, -233, 75, 240, 198]
The sorted array is:
[-331, -325, -325, -309, -292, -233, -104, -102, -88, -27, 75, 149, 198, 240, 351, 435, 478, 483, 484, 495]
-----Round 100000: Success!-----
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$
```

```
dicardo@ubuntu: ~/桌面/OSProject3/Fork-Join_Sorting
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$ java quickSort.java
Please choose whether randomly generate the array (input Y) or self input the array (input N):
N
Please input the size of the randomly generated (or input) array:
12
Please input the array:
6 7 -12135 124 0 0 1351351 24525 124135 5655 6 -1
The sorted array is:
[-12135, -1, 0, 0, 6, 6, 7, 124, 5655, 24525, 124135, 1351351]
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$
```

(2) 归并排序:

```
dicardo@ubuntu: ~/桌面/OSProject3/Fork-Join_Sorting
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$ java mergeSort.java
Please choose whether randomly generate the array (input Y) or self input the array (input N):
Y
Please input the size of the randomly generated (or input) array :
12
```



```
dicardo@ubuntu: ~/桌面/OSProject3/Fork-Join_Sorting
The sorted array is:
[-448, -431, -154, -144, -103, -65, 178, 190, 195, 402, 455, 477]
-----Round 99996: Success!-----
The initial array is:
[-51, -447, -193, 309, 346, -404, -236, -263, -240, 422, 42, -143]
The sorted array is:
[-447, -404, -263, -240, -236, -193, -143, -51, 42, 309, 346, 422]
-----Round 99997: Success!-----
The initial array is:
[-448, 130, -325, 34, 345, 253, 144, 317, 416, 360, -114, 496]
The sorted array is:
[-448, -325, -114, 34, 130, 144, 253, 317, 345, 360, 416, 496]
-----Round 99998: Success!-----
The initial array is:
[307, -4, -125, 475, 182, -77, 115, 390, -378, -456, -104, 372]
The sorted array is:
[-456, -378, -125, -104, -77, -4, 115, 182, 307, 372, 390, 475]
-----Round 99999: Success!-----
The initial array is:
[238, -37, -222, 493, 326, 12, 50, -262, -362, -500, 352, 4]
The sorted array is:
[-500, -362, -262, -222, -37, 4, 12, 50, 238, 326, 352, 493]
-----Round 100000: Success!-----
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$
```

```
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$ java mergeSort.java
Please choose whether randomly generate the array (input Y) or self input the array (input N):
N
Please input the size of the randomly generated (or input) array :
5
Please input the array:
-124 0 0 23525 125352352
The sorted array is:
[-124, 0, 0, 23525, 125352352]
dicardo@ubuntu:~/桌面/OSProject3/Fork-Join_Sorting$
```

## 五、实验结果

在经过一天的 coding 之后完成该项目。综合几个部分代码的运行及测试结果来看，此次实验取得成功。

## 六、实验反思

个人感觉这次项目中的核心思路类似，即都是利用多线程实现排序中的分治

思想。因此，此次实验的难点就落在了如何在 C 语言及 Java 语言中利用库函数实现多线程的操作，以及如何将这些操作同其他部分（如排序、I/O 等）进行一个合理的结构设计。总的来说，本次实验是成功的。在查阅一定资料之后，我对本次项目涉及到的多线程函数操作有了一个基本的理解，因此能够成功完成本次实验。

不过，本次实验中也能体现出我尚未不足的地方，具体表现在对多线程操作的不完全熟悉，以及 debug 能力的有待提高。相信在下一轮的实验中我会做的更好。