Report for OS Project 3

Multithreaded Sorting Application & Fork-Join Sorting Application

陈文迪 519021910071

I. 实验任务

1. 编写一个多线程排序程序，采用分治法的思想。将一个无序数组分成两部分用两个线程分别排序，再利用第三个线程将两个已排序的数组归并成一个大的有序数组。
2. 和任务1相似的思想，使用Java的fork-join API，分别采用快速排序和归并排序作为基础排序算法来实现一个多线程排序程序。

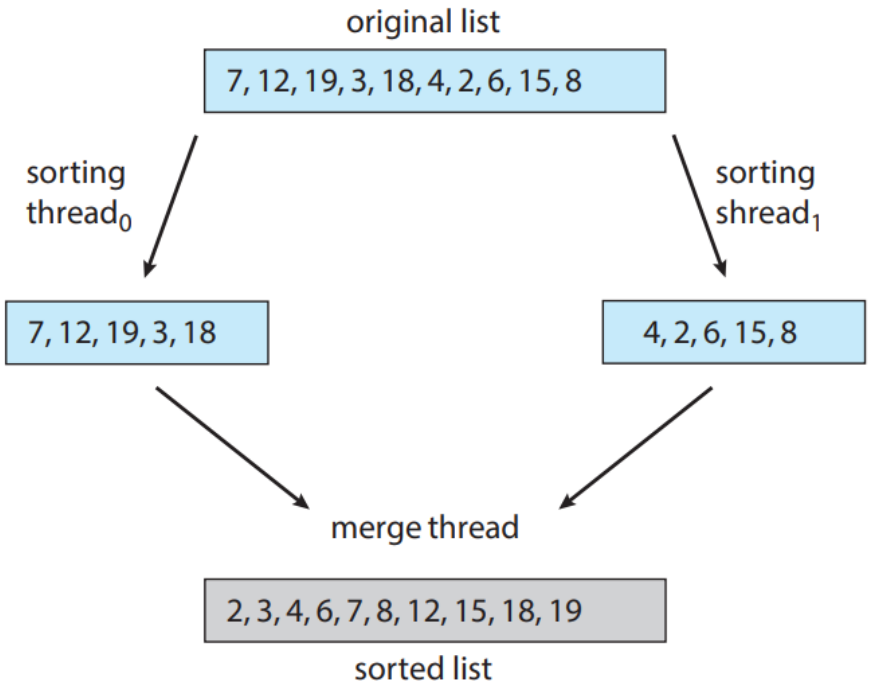
II. 实验思路

1. Multithreaded Sorting Application

在本项目中，我们需要熟悉`Pthread`库中与多线程编程有关的函数。由于在本次项目中采取的是fork-join策略，并且几个并发线程中并不会同时访问同一数据，不存在同步问题，因此我们所涉及到的函数如下。

```
pthread_attr_init(&first_attr);                                  //线程属性的初始化
pthread_create(&first_tid,&first_attr,sorter,left_data);         //线程创建
pthread_join(first_tid,NULL);                                    //线程合并
```

在具备了以上的准备知识的基础上，我们便可以开展我们的项目设计。依据书上给出的提示，我们的程序流程图如下：



具体来说：我们首先将原数据划分为两部分，接着我们创建两个排序线程分别将这两个子数组进行排序，使用`pthread_join()`函数等待这两个排序线程完成后，我们采用归并排序将已经有序的子数组归并到一个与原数组大小相同的数组中。在本项目中，基础排序算法为快速排序。

`pthread_create()`的第三个参数为一个函数指针，创建的子线程将执行这个函数。因此我们定义了两个工作函数分辨进行排序和归并。

```
void *sorter(void * param){
    parameters* p = (parameters *) param;
    quick_sort(unsorted,p->l,p->r);
}

void *merger(void * param){
    parameters* p = (parameters *) param;
    int mid = (p->l+p->r+1)>>1;
    int i = 0,j = mid+1,k = 0;
    while(i<=mid && j<=p->r){
        if(unsorted[i]<unsorted[j]){
            sorted[k++] = unsorted[i++];
        }
        else{
            sorted[k++] = unsorted[j++];
        }
    }
    while(i<=mid){
        sorted[k++] = unsorted[i++];
    }
    while(j<=p->r){
        sorted[k++] = unsorted[j++];
    }
}
```

`pthread_create()`的第四个参数为一个`void`指针，表示工作函数所需要的参数，因此我们定义了一个参数结构体来方便参数的传递。

```
typedef struct{
    int l;
    int r;
}parameters;
```

具体实现的核心代码如下：

```
int main(){
    srand((unsigned)time(NULL));
    init();
    pthread_t first_tid, second_tid, merge_tid;
    pthread_attr_t first_attr, second_attr, merge_attr;

    parameters *left_data = (parameters*) malloc(sizeof(parameters));
    parameters *right_data = (parameters*) malloc(sizeof(parameters));
    parameters *merge_data = (parameters*) malloc(sizeof(parameters));
    left_data->l = 0;
    left_data->r = (SIZE>>1);
    right_data->l = left_data->r+1;
    right_data->r = SIZE-1;
    merge_data->l = 0;
    merge_data->r = SIZE-1;

    pthread_attr_init(&first_attr);
    pthread_attr_init(&second_attr);
```

```
    pthread_attr_init(&merge_attr);

    pthread_create(&first_tid,&first_attr,sorter,left_data);
    pthread_create(&second_tid,&second_attr,sorter,right_data);
    pthread_join(first_tid,NULL);
    pthread_join(second_tid,NULL);

    pthread_create(&merge_tid,&merge_attr,merger,merge_data);
    pthread_join(merge_tid,NULL);

    free(left_data);
    free(right_data);
    free(merge_data);

    for(int i =0;i<SIZE;++i){
        printf("%d ", sorted[i]);
        if(i%10==0) printf("\n");
    }

    return 0;
}
```

2. Fork-Join Sorting Application

Quicksort Implementation

本项目与项目1最大的不同在于，由于我们可以继承Java的 `RecursiveAction` 类来实现线程池的功能，我们可以将fork-join的思想进一步发展。每一次递归都可以利用快排的特性通过 `pivot` 将父数组划分为两个较小的子数组，对于子数组，若其规模已经足够小（小于某一个阈值），则我们通过简单的插入排序即可使子数组有序，不然则继续递归，将子数组进行划分。这种设计可以更好地利用多线程结构来加速排序过程。

具体实现的核心代码如下：

```
protected void compute() {
    if (end - begin < THRESHOLD) {
        // conquer stage
        for (int i = begin+1; i <= end; i++){
            int p = array[i];
            int j = i-1;
            while(j>=begin && array[j]>p){
                array[j+1] = array[j];
                j--;
            }
            array[j+1] = p;
        }
    }
    else {
        // divide stage
        int r = begin, s = end, pivot = array[begin];
        while (r < s)
        {
            while(r < s && array[s] >= pivot){
                s--;
            }
            if(r < s)
                array[r++] = array[s];
```

```
            while(r < s && array[r] < pivot){
                r++;
            }
            if(r < s)
                array[s--] = array[r];
        }
        array[r] = pivot;

        int mid = r;

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

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

        int[] temp = new int[SIZE];

        int i = begin,j = mid+1,k = begin;
        while(i<=mid && j<=end){
            if(array[i]<array[j])
                temp[k++] = array[i++];
            else
                temp[k++] = array[j++];
        }
        while(i<=mid)
            temp[k++] = array[i++];

        while(j<=end)
            temp[k++] = array[j++];

        int p = begin;
        while(p<=end){
            array[p] = temp[p];
            p++;
        }
    }
}
```

Mergesort Implementation

采用归并排序的实现是类似的，不同的是我们不需要通过`pivot`来划分数组，而是直接将数组二分成两个子数组。其余的实现与`Quicksort`版本类似。

```
protected void compute() {
    if (end - begin < THRESHOLD) {
        // conquer stage
        for (int i = begin+1; i <= end; i++){
            int p = array[i];
            int j = i-1;
            while(j>=begin && array[j]>p){
                array[j+1] = array[j];
                j--;
            }
```

```
            array[j+1] = p;
        }
    }
    else {
        // divide stage
        int mid = begin + (end - begin) / 2;

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

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

        int[] temp = new int[SIZE];

        int i = begin,j = mid+1,k = begin;
        while(i<=mid && j<=end){
            if(array[i]<array[j])
                temp[k++] = array[i++];
            else
                temp[k++] = array[j++];
        }
        while(i<=mid)
            temp[k++] = array[i++];

        while(j<=end)
            temp[k++] = array[j++];

        int p = begin;
        //System.out.println(Arrays.toString(temp));
        while(p<=end){
            array[p] = temp[p];
            p++;
        }
    }
}
```

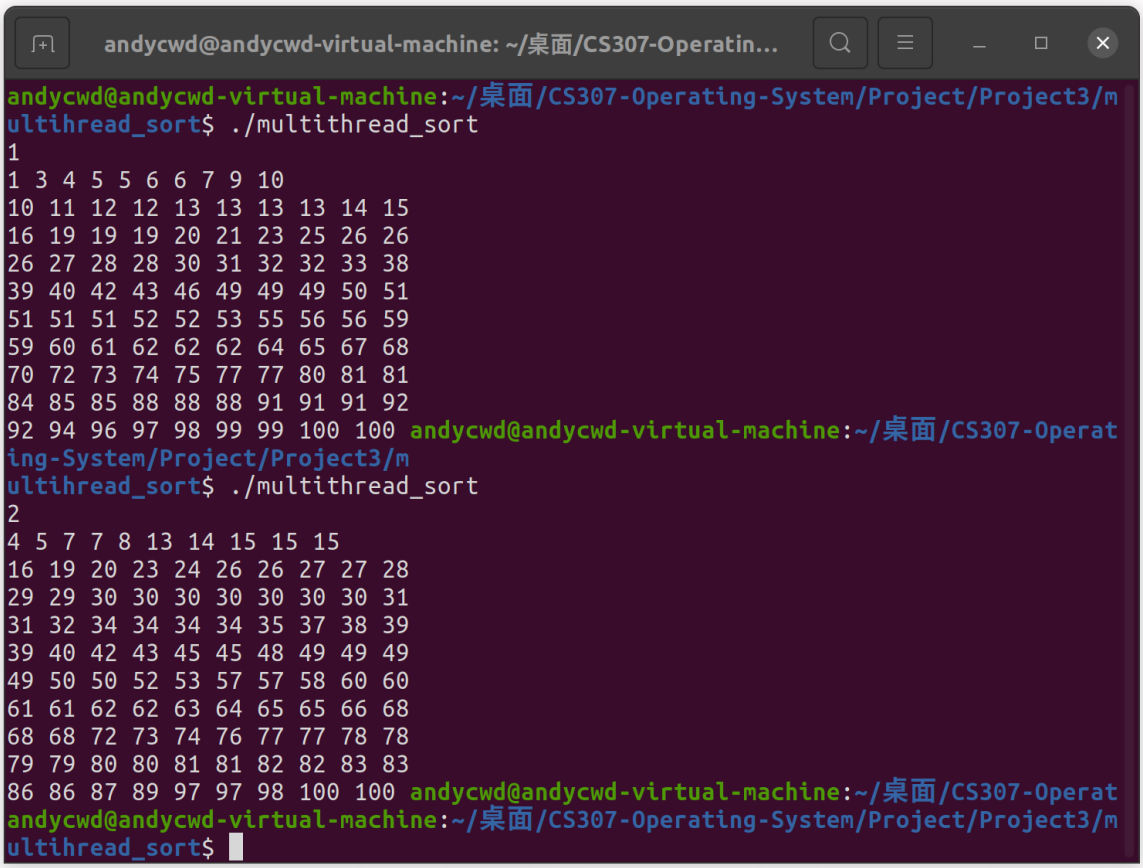
III. 实验过程

1. Multithreaded Sorting Application

我们使用随机数生成一个大小为100，元素范围为1~100的原始数组用作测试。

```
void init(){
    for(int i = 0; i < SIZE;++i)
        unsorted[i] = rand()%100+1;
}
```

测试结果如下：



符合我们的设计预期。

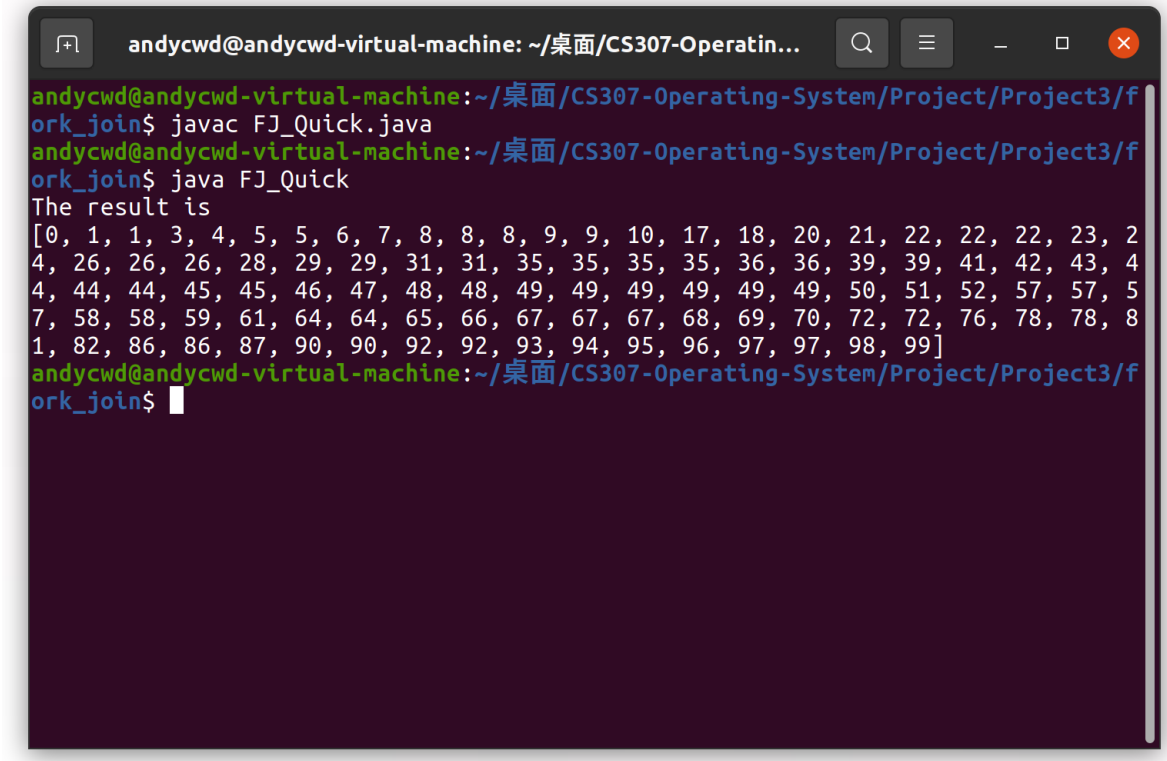
2. Fork-Join Sorting Application

我们采用类似与项目1的测试方法，并把阈值设置为5。

```
for (int i = 0; i < SIZE; i++) {
    array[i] = rand.nextInt(100);
}
```

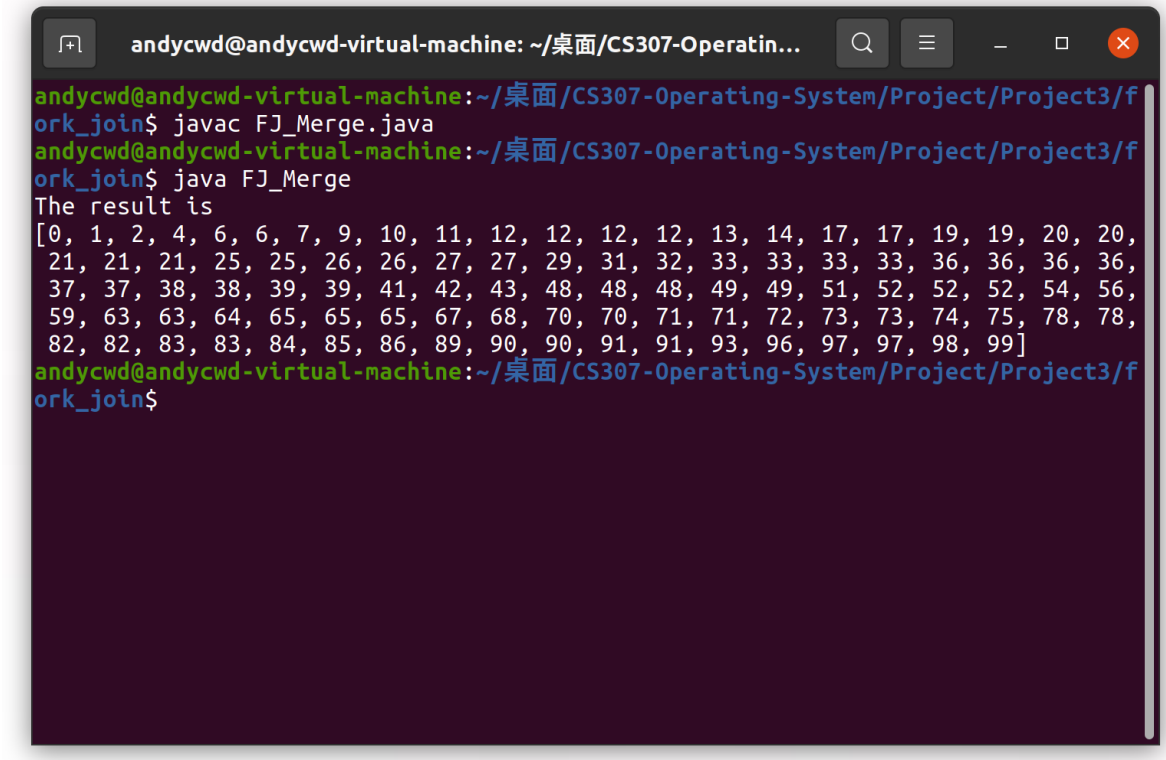
Quicksort Implementation

测试结果如下：



Mergesort Implementation

测试结果如下：



均与我们的设计预期相符合。

IV. 遇到的问题

1. 工作函数的参数解析问题

由于我们向工作函数传递的是一个`void`指针，这类似于一个打包过程，因此在工作函数内部需要通过类型转换来进行解包，而不能直接通过`void`指针传递。

```
arameters* p = (parameters *) param;
```

2. Java程序编写问题

在开发本项目之前，我对Java的语法并不是特别熟悉。好在Java与C语言家族有许多语法是类似的，对于像`RecursiveAction`这样具体的类，我们可以通过查看官方文档来使用。

V. 参考资料

[1] Operating System Concept 10th Edition

[2] The Pthreads Library - Multithreaded Programming Guide (oracle.com)

[3] RecursiveAction (Java Platform SE 8) (oracle.com)