Project 3-- Multithreaded Sorting & Fork-Join Sorting

陈思远 518021910567

1. 实验内容和目标

1. 用C写一个多线程的排序程序，用两个线程各排序一半，第三个线程合并两个已排序的数组。
2. 用Java的fork-join并行API实现多线程的快排和归并排序。

2. Multithreaded Sorting Application

2.1 要求与实现

这部分要求我们用多线程的方式对输入的数组进行排序。使用两个线程分别排序（可以自选排序算法）前一半和后一半数组，再创建第三个线程将两个已排序的数组进行归并，从而完成整个数组的排序。需要注意的是，第三个线程必须等到前两个线程全部完成后才可以开始执行，因此执行`pthread_join`函数，用阻塞的方式等待前两个线程执行完成，再执行第三个线程。关键代码如下。

```
    //排序线程
    pthread_t tid1,tid2;
    parameters *data1=(parameters *)malloc(sizeof(parameters));
    parameters *data2=(parameters *)malloc(sizeof(parameters));
    //给各个参数赋值
    data1->num=total_num/2;
    data2->num=total_num-data1->num;
    data1->arr=(int *)malloc(sizeof(int)*data1->num);
    data2->arr=(int *)malloc(sizeof(int)*data2->num);

    for(int i=0;i<data1->num;i++){
        data1->arr[i]=sort_arr[i];
    }
    for(int i=0;i<data2->num;i++){
        data2->arr[i]=sort_arr[i+total_num/2];
    }
   //创建新线程，指定线程函数和传入参数
    pthread_create(&tid1,NULL,sortit,(void*)data1);//排序左半边
    pthread_create(&tid2,NULL,sortit,(void*)data2);//排序右半边
   //用阻塞的方式等待线程结束
    pthread_join(tid1,NULL);
    pthread_join(tid2,NULL);
```

```
    //归并线程
    pthread_t tid3;
    merge_para *mp=(merge_para *)malloc(sizeof(merge_para));
    mp->d1=data1;
    mp->d2=data2;
    //创建新线程，指定线程函数和传入参数
    pthread_create(&tid3,NULL,mergeit,(void*)mp);//将排序后的数组归并
    //用阻塞的方式等待线程结束
    pthread_join(tid3,NULL);

    printf("sorted array:\n");  //输出已排序的数组
    for(int i=0;i<total_num;i++){
        printf("%d ",sort_arr[i]);
    }
    //释放空间
    free(data1);
    free(data2);
    free(mp);
```

接下来介绍如何向线程传递参数，以及线程函数的具体实现。根据书本中介绍的方法，可以将要传递给线程的参数用结构体包装起来，作为`pthread_create`函数的第四个参数传入即可。`pthread_create`的第三个参数则表示使用的线程函数，分别为`sortit`和`mergeit`，具体代码如下。

```
typedef struct{
    int num;
    int *arr;
}parameters;    //传入排序线程的参数
void *sortit(void *param){              //排序线程
    parameters *tmp;
    tmp=(parameters*)param;
    int i,j,t;
    for(i=0;i<tmp->num-1;i++){
        for(j=0;j<tmp->num-i-1;j++){
            if(tmp->arr[j]>tmp->arr[j+1]){
                t=tmp->arr[j+1];
                tmp->arr[j+1]=tmp->arr[j];
                tmp->arr[j]=t;
            }
        }
    }
}
```

```
typedef struct{
    parameters* d1;
    parameters* d2;
}merge_para;    //传入归并线程的参数

void *mergeit(void *param){             //归并线程
    merge_para* tmp;
    tmp=(merge_para*)param;
    int i=0,j=0,k=0;
    while(i<tmp->d1->num && j<tmp->d2->num){
        if(tmp->d1->arr[i]<=tmp->d2->arr[j]){
            sort_arr[k++]=tmp->d1->arr[i++];
        }else{
```

```
            sort_arr[k++]=tmp->d2->arr[j++];
        }
    }
    while(i<tmp->d1->num){
        sort_arr[k++]=tmp->d1->arr[i++];
    }
    while(j<tmp->d2->num){
        sort_arr[k++]=tmp->d2->arr[j++];
    }
}
```

2.2 结果

gcc在编译多线程程序时需要额外链接lpthread，若不加，就会出现编译错误。成功编译后，运行程序。首先输入待排序的数组元素个数和各个元素，输出排序后的结果。运行截图如下。

3. Fork-Join Sorting Application

3.1 要求与实现

这部分要求使用Java的fork-join并行API，实现多线程的快速排序和归并排序。这两种排序都使用了分治法，因此非常适合多线程。我将两种排序在同一个程序中实现。在main函数中进行下列步骤。

1. 创建一个`ForkJoinPool`类型的实例`pool`作为线程池。
2. 随机生成100个0到999的整数作为待排序的数组，用于快速排序。
3. 调用`QuickSort`实现快速排序。`QuickSort`是一个可以并行、合并，但没有返回值的任务，继承`RecursiveAction`类。
4. 调用`invoke`，等待`quicksort`任务完成，才能执行后面的代码。
5. 打印快排结果。
6. 随机生成100个0到99的整数作为待排序数组，用于归并排序。
7. 调用`MergeSort`实现归并排序。`MergeSort`是一个可以并行、合并，但没有返回值的任务，继承`RecursiveAction`类。
8. 调用`invoke`，等待`mergeksort`任务完成，才能执行后面的代码。

9. 打印归并排序结果。

具体代码和注释如下。

```
public class ForkJoinSort {

    static final int SIZE=100;
    private int[] array;

    public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool();
        int[] array = new int[SIZE];

        java.util.Random rand = new java.util.Random();
        //随机生成100个0到999的整数，赋给array
        for (int i = 0; i < SIZE; i++) {
            array[i] = rand.nextInt(1000);
        }
        //快速排序
        QuickSort quicksort=new QuickSort(array,0,array.length-1);
        //调用invoke，调用后等待quicksort任务完成，才能执行后面的代码
        pool.invoke(quicksort);
        //打印排序结果
        System.out.print("QuickSort(sorted array):\n");
            for (int i=0; i<array.length; i++){
            System.out.printf("%d ", array[i]);
        }
        System.out.print("\n\n\n");

        //随机生成100个0到99的整数，赋给array
        for (int i = 0; i < SIZE; i++) {
            array[i] = rand.nextInt(100);
        }
        //归并排序
        MergeSort mergesort=new MergeSort(array,0,array.length-1);
        //调用invoke，调用后等待mergesort任务完成，才能执行后面的代码
        pool.invoke(mergesort);
        //打印排序结果
        System.out.print("MergeSort(sorted array):\n");
            for (int i=0; i<array.length; i++)
            System.out.printf("%d ", array[i]);
        System.out.print("\n");
    }
}
```

接下来，介绍如何实现快速排序。快排的基本思想是选择一个元素作为`pivot`（此处选择首个元素），遍历数组使这个元素左边的元素都比它小，右边的都比它大。然后对左右两边的数组递归操作，最终实现整个数组的排序。

将阈值设为10，当数组长度小于10时，直接采用插入排序进行排序，否则继续分为左右两个线程，分别排序左边的数组和右边的数组。

```
class QuickSort extends RecursiveAction
{
    private int left;
    private int right;
    private int[] array;
```

```
    public QuickSort(int array[],int left, int right) {
        this.left = left;
        this.right = right;
        this.array = array;
    }
    protected void insertSort(){
        for(int i=left+1;i<right+1;i++){
            int tmp=array[i];
            int j=i-1;
            while(j>=left&&array[j]>tmp){
                array[j+1]=array[j];
                j--;
            }
            array[j+1]=tmp;
        }
    }

    protected int divide(int left,int right){
        int pivot=array[left];
        do{
            while(left<right&&array[right]>=pivot)--right;
            if(left<right){
                array[left]=array[right];
                ++left;
            }
            while(left<right&&array[left]<=pivot)++left;
            if(left<right){
                array[right]=array[left];
                --right;
            }
        }while(left!=right);
        array[left]=pivot;
        return left;
    }
    @override
    protected void compute() {
        if(right-left>10){
            int mid=divide(left,right);
            RecursiveAction leftpart=new QuickSort(array,left,mid-1);
            RecursiveAction rightpart=new QuickSort(array,mid+1,right);
            leftpart.fork();
            rightpart.fork();
            leftpart.join();
            leftpart.join();
            invokeAll(leftpart,rightpart);
        }else{
            insertSort();
        }
    }
}
```

接着，实现归并排序。递归地对左右两边调用`MergeSort`，再将两个已排序的数组归并为一个数组。同样的，将阈值设为10，当数组长度小于10时，直接采用插入排序进行排序。具体代码如下。

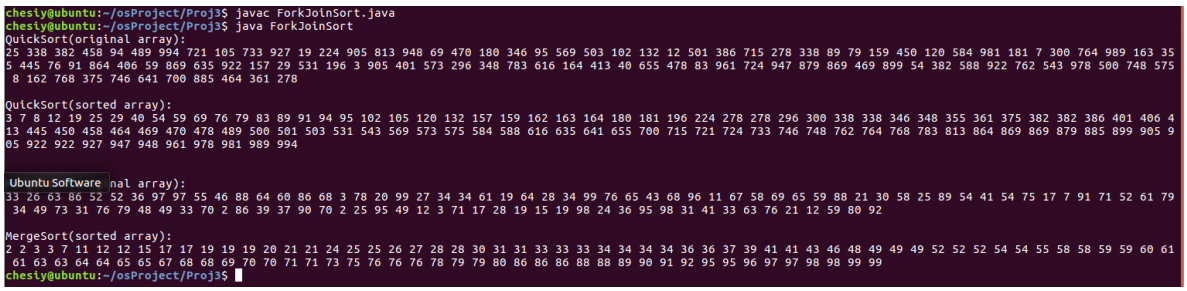
```
class MergeSort extends RecursiveAction
{
```

```
    private int left;
    private int right;
    private int[] array;

    public MergeSort( int[] array,int left, int right) {
        this.left = left;
        this.right = right;
        this.array = array;
    }
    protected void insertSort(){
        for(int i=left+1;i<right+1;i++){
            int tmp=array[i];
            int j=i-1;
            while(j>=left&&array[j]>tmp){
                array[j+1]=array[j];
                j--;
            }
            array[j+1]=tmp;
        }
    }
    protected void Merge(int left,int mid,int right){
        int len=right-left+1;
        int tmp[]=new int[right-left+1];
        int i=left,j=mid+1,k=0;
        while(i<=mid&&j<=right){
            tmp[k++]=array[i]<=array[j]?array[i++]:array[j++];
        }
        while(i<=mid){
            tmp[k++]=array[i++];
        }
        while(j<=right){
            tmp[k++]=array[j++];
        }
        for(int t=0;t<len;t++){
            array[left++]=tmp[t];
        }
    }
    @override
    protected void compute() {
        if(right-left>10){
            int mid=left+(right-left)/2;
            RecursiveAction leftpart=new MergeSort(array,left,mid);
            RecursiveAction rightpart=new MergeSort(array,mid+1,right);
            leftpart.fork();
            rightpart.fork();
            leftpart.join();
            leftpart.join();
            invokeAll(leftpart,rightpart);
            Merge(left,mid,right);
        }else{
            insertSort();
        }
    }
}
```

3.2 结果

编译并运行后，得到的运行结果如下。



4. 总结与思考

这次project中，我实现多种排序算法的多线程版本，对如何用C和Java进行多线程编程有了初步的了解。我认为相比于C，Java的fork-join并行API运用起来更加简洁易懂，再加上可重载函数，灵活度也很高。上个暑假，我曾经用Java写过一个多线程的小游戏。当时对线程、同步等等概念非常模糊，所以小游戏中虽然用到了多线程，但各个线程之间彼此独立，不需要相互传递参数，也没有执行的先后顺序要求。而这次project在多线程的实现中用到了线程池、`invokeAll`等，线程之间也需要考虑执行的先后顺序，更为复杂，让我学到很多。