

Project 3-- Multithreaded Sorting & Fork-Join Sorting

陈思远 518021910567

Project 3-- Multithreaded Sorting & Fork-Join Sorting

1. 实验内容和目标
2. Multithreaded Sorting Application
 - 2.1 要求与实现
 - 2.2 结果
3. Fork-Join Sorting Application
 - 3.1 要求与实现
 - 3.2 结果
4. 总结与思考

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pthread，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。

```

chesiy@ubuntu:~/osProject/Proj3$ gcc multsort.c -lpthread
chesiy@ubuntu:~/osProject/Proj3$ ./a.out
6
3 5 2 7 1 4
original array:
3 5 2 7 1 4
sorted left half:
2 3 5
sorted right half:
1 4 7
sorted array:
1 2 3 4 5 7
chesiy@ubuntu:~/osProject/Proj3$ ./a.out
7
2 9 70 4 3 56 21
original array:
2 9 70 4 3 56 21
sorted left half:
2 9 70
sorted right half:
3 4 21 56
sorted array:
2 3 4 9 21 56 70

```

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mergesort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();
        rightpart.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();
        rightpart.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。

```

chesty@ubuntu:~/osProject/Proj3$ javac ForkJoinSort.java
chesty@ubuntu:~/osProject/Proj3$ java ForkJoinSort
QuickSort(original array):
25 338 382 458 94 489 994 721 105 733 927 19 224 905 813 948 69 470 180 346 95 569 503 102 132 12 501 386 715 278 338 89 79 159 450 120 584 981 181 7 300 764 989 163 35
5 445 76 91 864 406 59 869 635 922 157 29 531 196 3 905 401 573 296 348 783 616 164 413 40 655 478 83 961 724 947 879 869 469 899 54 382 588 922 762 543 978 500 748 575
8 162 768 375 746 641 700 885 464 361 278

QuickSort(sorted array):
3 7 8 12 19 25 29 40 54 59 69 76 79 83 89 91 94 95 102 105 120 132 157 159 162 163 164 180 181 196 224 278 278 296 300 338 338 346 348 355 361 375 382 382 386 401 406 4
13 445 450 458 464 469 470 478 489 500 501 503 531 543 569 573 575 584 588 616 635 641 655 700 715 721 724 733 746 748 762 764 768 783 813 864 869 869 879 885 899 905 9
05 922 922 927 947 948 961 978 981 989 994

MergeSort(original array):
33 26 63 86 52 52 36 97 97 55 46 88 64 60 86 68 3 78 20 99 27 34 34 61 19 64 28 34 99 76 65 43 68 96 11 67 58 69 65 59 88 21 30 58 25 89 54 41 54 75 17 7 91 71 52 61 79
34 49 73 31 76 79 48 49 33 70 2 86 39 37 90 70 2 25 95 49 12 3 71 17 28 19 15 19 98 24 36 95 98 31 41 33 63 76 21 12 59 80 92

MergeSort(sorted array):
2 2 3 3 7 11 12 12 15 17 17 19 19 19 20 21 21 24 25 25 26 27 28 28 30 31 31 33 33 33 34 34 34 34 36 36 37 39 41 41 43 46 48 49 49 49 52 52 52 54 54 55 58 58 59 59 60 61
61 63 63 64 64 65 65 67 68 68 69 70 70 71 71 73 75 76 76 76 78 79 79 80 86 86 86 88 88 89 90 91 92 95 95 96 97 97 98 98 99 99
chesty@ubuntu:~/osProject/Proj3$

```

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。