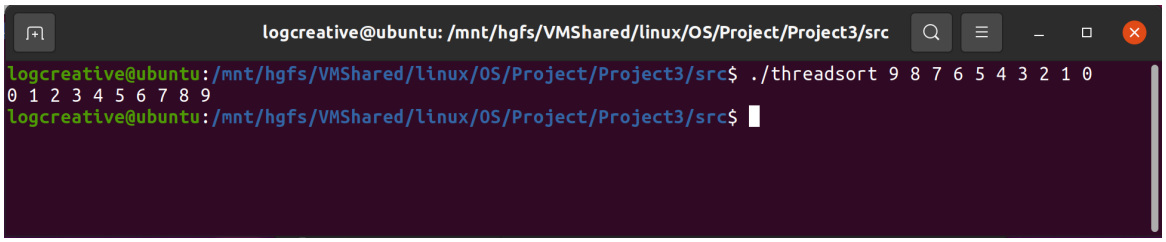
操作系统(D)

项目 3

李子龙 518070910095

2021 年 3 月 20 日

一 多线程排序程序



使用命令行的参数获取需要排序的数组，使用动态内存分配原数组和排序数组。原数组两边分别开一个线程用冒泡排序，最后归并两个数组到排序数组中。定义了一个结构体用于传递参数：

```
typedef struct {
    int start;
    int end;
} sort_param;
```

Listing 1: src/threadsort.c

```
#include<pthread.h>
#include<stdio.h>
#include<stdlib.h>

int* old_list;
int* sort_list;

typedef struct {
    int start;
    int end;
} sort_param;

void bubblesort(sort_param* sp){
    int start = sp->start;
    int end = sp->end;

    int flag = 1;
    for(int i = start + 1; i <= end && flag; ++i){
        flag = 0;
```

```
        for(int j = start; j <= end - i + start; ++j){
            if(old_list[j+1]<old_list[j]){
                int tmp = old_list[j];
                old_list[j] = old_list[j+1];
                old_list[j+1] = tmp;
                flag = 1;
            }
        }
    }

    pthread_exit(0);
}

void mergearray(int mid, int end){
    int left = 0;
    int right = mid + 1;
    int cur = 0;
    while(left<=mid && right <= end)
        if(old_list[left]<=old_list[right])
            sort_list[cur++] = old_list[left++];
        else sort_list[cur++] = old_list[right++];

    while(left<=mid) sort_list[cur++] = old_list[left++];
    while(right<=end) sort_list[cur++] = old_list[right++];
}

int main(int argc, char *argv[]){
    if(argc == 1){
        fprintf(stderr, "Please input the array!\n");
        return 1;
    }

    old_list = (int*)malloc((argc-1)*sizeof(int));
    sort_list = (int*)malloc((argc-1)*sizeof(int));
    for(int i = 1; i < argc; ++i)
        old_list[i-1] = atoi(argv[i]);

    pthread_t sorting_thread[2];
    pthread_attr_t attr[2];

    int mid = (argc-2)/2;

    pthread_attr_init(&attr[0]);
    sort_param *sp0 = (sort_param*) malloc(sizeof(sort_param));
    sp0->start = 0;
    sp0->end = mid;
    pthread_create(&sorting_thread[0],&attr[0],bubblesort,sp0);
    pthread_join(sorting_thread[0],NULL);

    pthread_attr_init(&attr[1]);
    sort_param *sp1 = (sort_param*) malloc(sizeof(sort_param));
    sp1->start = mid + 1;
    sp1->end = argc - 2;
    pthread_create(&sorting_thread[1],&attr[1],bubblesort,sp1);
    pthread_join(sorting_thread[1],NULL);
```

```
    mergearray(mid,argc-2);

    for(int i = 0; i < argc-1; ++i)
        fprintf(stdout, "%d ", sort_list[i]);
    fprintf(stdout, "\n");

    free(old_list);
    free(sort_list);
    return 0;
}
```

二 分离–联合排序程序

使用 java 实现快速排序和归并排序的多线程版本。当需要排序的数组量小于阈值 THRESHOLD 100 时，将会采用选择排序。派生了了 RecursiveAction 类用于多线程运算。使用 Comparable 用于任何可比较类型的比较。

取消 main 函数开始时的注释，可以用于手动输入数据进行排序。这里的测试基于 1000 个 0 9 的随机数。

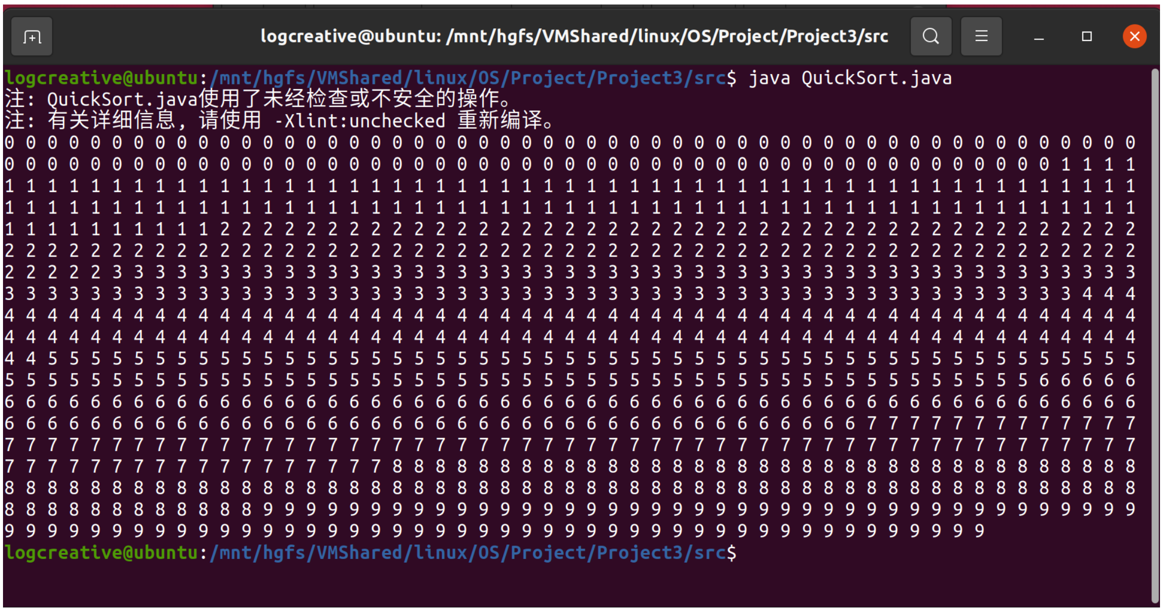
```
        Scanner scanner = new Scanner(System.in);
        String str = scanner.nextLine();
        scanner.close();
        String array[] = str.split(" ");
```

Listing 2: src/QuickSort.java

```
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class QuickSort extends RecursiveAction{

    static final int THRESHOLD = 100;
    static final int SIZE = 1000;
```

```
private int low;
private int high;
private Comparable[] array;

public QuickSort(Comparable[] array, int low, int high){
    this.array = array;
    this.low = low;
    this.high = high;
}

protected void compute(){
    if(high - low < THRESHOLD){
        // Apply Selection Sort
        int i,j,min;
        for(i = 0; i < array.length; i++){
            min = i;
            for(j = i+1; j < array.length; j++)
                if(less(array[j],array[min]))
                    min = j;
            exch(array, i, min);
        }
    } else {
        int mid = divide(array, low, high);

        QuickSort leftAction = new QuickSort(array, low, mid-1);
        QuickSort rightAction = new QuickSort(array, mid+1, high);

        leftAction.fork();
        rightAction.fork();

        rightAction.join();
        leftAction.join();
    }
}

public static int divide(Comparable[] a, int low, int high){
    Comparable k = a[low];
    do{
        while(low<high && less(k, a[high])) --high;
        if(low<high){a[low]=a[high]; ++low;}
        while(low<high && less(a[low], k)) ++low;
        if(low<high){a[high]=a[low]; --high;}
    } while (low!=high);
    a[low] = k;
    return low;
}

public static boolean less(Comparable v, Comparable w){
    return v.compareTo(w) < 0;
}

public static void exch(Comparable[] a, int i, int j){
    Comparable t = a[i];
    a[i] = a[j];
    a[j] = t;
}
```

```
    private static void show(Comparable[] a){
        for (int i = 0; i < a.length; i++)
            System.out.print(a[i] + " ");
        System.out.println();
    }

    public static boolean isSorted(Comparable[] a){
        for(int i = 1; i < a.length; i++)
            if(less(a[i],a[i-1])) return false;
        return true;
    }

    public static void main(String[] args) {
        // Scanner scanner = new Scanner(System.in);
        // String str = scanner.nextLine();
        // scanner.close();
        // String array[] = str.split(" ");

        Comparable[] array = new Comparable[SIZE];

        // create SIZE random integers between 0 and 9
        java.util.Random rand = new java.util.Random();

        for (int i = 0; i < SIZE; i++) {
            array[i] = rand.nextInt(10);
        }

        ForkJoinPool pool = new ForkJoinPool();

        QuickSort action = new QuickSort(array, 0, array.length - 1);

        pool.invoke(action);

        assert isSorted(array);
        show(array);
    }
}
```



Listing 3: src/MergeSort.java

```java
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveAction;

public class MergeSort extends RecursiveAction{

    static final int THRESHOLD = 100;
    static final int SIZE = 1000;

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

    public MergeSort(Comparable[] array, int left, int right){
        this.array = array;
        this.left = left;
        this.right = right;
    }

    protected void compute(){
        if(right - left < THRESHOLD){
            // Apply Selection Sort
            int i,j,min;
            for(i = 0; i < array.length; i++){
                min = i;
                for(j = i+1; j < array.length; j++)
                    if(less(array[j],array[min]))
                        min = j;
                exch(array, i, min);
            }
        } else {
            int mid = (left + right) / 2;

            MergeSort leftAction = new MergeSort(array, left, mid);
            MergeSort rightAction = new MergeSort(array, mid+1, right);

            leftAction.fork();
```

```
        rightAction.fork();

        rightAction.join();
        leftAction.join();

        merge(array, left, mid+1, right);
    }
}

public static void merge(Comparable[] a, int left, int mid, int right){
    Comparable aux[] = new Comparable[right-left+1];
    int i = left;
    int j = mid;
    int k = 0;
    while(i<mid && j<=right){
        if(less(a[i],a[j])) aux[k++] = a[i++];
        else aux[k++] = a[j++];
    }
    while(i<mid) aux[k++] = a[i++];
    while(j<=right) aux[k++] = a[j++];
    for(i = 0, k = left; k<=right;) a[k++] = aux[i++];
}

public static boolean less(Comparable v, Comparable w){
    return v.compareTo(w) < 0;
}

public static void exch(Comparable[] a, int i, int j){
    Comparable t = a[i];
    a[i] = a[j];
    a[j] = t;
}

private static void show(Comparable[] a){
    for (int i = 0; i < a.length; i++)
        System.out.print(a[i] + " ");
    System.out.println();
}

public static boolean isSorted(Comparable[] a){
    for(int i = 1; i < a.length; i++)
        if(less(a[i],a[i-1])) return false;
    return true;
}

public static void main(String[] args) {
    // Scanner scanner = new Scanner(System.in);
    // String str = scanner.nextLine();
    // scanner.close();
    // String array[] = str.split(" ");

    Comparable[] array = new Comparable[SIZE];

    // create SIZE random integers between 0 and 9
    java.util.Random rand = new java.util.Random();
```

```
        for (int i = 0; i < SIZE; i++) {
            array[i] = rand.nextInt(10);
        }

        ForkJoinPool pool = new ForkJoinPool();

        MergeSort action = new MergeSort(array, 0, array.length - 1);

        pool.invoke(action);

        assert isSorted(array);
        show(array);
    }
}
```