浙江大学实验报告

专业：混合班
姓名：徐圣泽
学号：3190102721
日期：2020.6.9

课程名称：C 程序设计专题　　指导老师：翁恺　　实验名称：多线程排序

一、实验要求

用`pthread`库做多线程实现的归并排序和快速排序，并比较性能。

二、实验思路

（一）代码思路

本次实验中主要利用双线程和单线程实现归并排序和快速排序，并对比程序在各种实现方式下处理不同数量级的数据时的效率。

代码的主体内容仍然是排序算法中实现功能的几个函数，即：

- 归并排序：

①`void mergeSort(int a[], int len, int start, int end);`

②`void merge(int a[],int len,int left,int mid,int right);`

- 快速排序：

①`int partition( int a[], int left, int right );`

②`void quicksort( int a[], int left, int right );`

这些代码就不再进行解释。

再创建多个线程，将创建的数组`a[]`分成多组分别进行排序，最后再处理一些细节问题即可。

（二）分析思路

用不同数量级的数据，分别对两种排序算法的性能进行测试，包括算法的效率和稳定性；同时将单线程与多线程进行比较。

三、代码解释

（一）归并排序

①结构体

```
typedef struct{
    int first;//第一个数据位置
    int last;//最后一个数据位置
    int length;//待处理数组数据长度
} MY_ARRAY;
```

将结构体作为参数传入定义的函数中，带来了很多便利。

②随机数组

- 原本采用`rand()`函数来产生随机数，但是每次产生的一系列数都是相同的。
- 查阅资料发现，在C语言中`rand()`函数可以用来产生随机数，但是这不是真正意义上的随机数，是一个伪随机数，它是根据一个数，我们可以称它为种子，为基准以某个递推公式推算出来的一系数，但这不是真正的随机数。当计算机正常开机后，这个种子的值是定了的，除非破坏了系统，为了改变这个种子的值。
- 后来发现`srand()`函数可以较好地解决这个问题，C语言中`srand((time(NULL));`语句表示设置一个随机种子，并且保证每次运行随机种子不同，则每次产生的随机数都是不同的。

```
void Create_array(){
    srand((int)time(NULL));
    int i;
    for(i=0;i<array_length;i++) {
        a[i]=rand();
        //printf("%d ",a[i]);
    }
    //printf("\n");
}
```

因此`srand()`解决了`rand()`函数的弊端，每次都可以提供与前一次不同的随机数。

③`void mySort(MY_ARRAY *array)`函数

这个函数的目的是为了更好地使我们所用到的参数被利用到`mergeSort`函数中。

```
void mySort(MY_ARRAY *array){
    mergeSort(a,array->length,array->first,array->last);
}
```

这样我们只需要在`main`函数中创建线程，在每个线程中传入相应范围的需要排序的数组数据即可。

④`main`函数

```
    MY_ARRAY arr[N];
    for(i=0;i<N;i++){
        if(i==0){
            arr[i].first=array_length*(i)/N;
            arr[i].last=array_length*(i+1)/N-1;
        }else if(i<N-1){
            arr[i].first=arr[i-1].last+1;
            arr[i].last=arr[i].first+array_length/N;
        }else{
            arr[i].first=arr[i-1].last+1;
            arr[i].last=array_length-1;
        }
        arr[i].length=arr[i].last-arr[i].first+1;
        //printf("%d %d\n",arr[i].first,arr[i].last);
    }
```

最简单的方式是创建两个线程，分别处理数组的左半部分和右半部分：

```
    pthread_t tid;
    for(i=0;i<N;i++){
        pthread_create(&tid, NULL, (void *)mySort, (void *)&arr[i]);
    }
```

最后再将两个数组合并，这一步需要两两合并，再两两合并，直至合并至一组。

⑤算法效率

利用`struct timeval`结构体。

```
struct timeval{
    long tv_sec;         /* Seconds. */
    long tv_usec;  /* Microseconds. */
};
```

利用函数`gettimeofday`可以获得当前的时间。

函数所需头文件为`#include<sys/time.h>`，原型为`int gettimeofday(struct timeval *tv,struct timezone *tz )`，其中把目前的时间用`tv`结构体返回，当地时区的信息则放到`tz`所指的结构中。

本题我们只需要得到`tv`中的信息即可，所以不需要返回`tz`对应的结构体，第二个参数为`NULL`即可。

```
    struct timeval T1,T2;
    int deltaT;
    gettimeofday(&T1,NULL);
    /*排序部分*/
    gettimeofday(&T2,NULL);
    deltaT=(T2.tv_sec-T1.tv_sec)*1000+(T2.tv_usec - T1.tv_usec)/1000;
    printf("time is %d millisecond\n",deltaT);
    //int uT= (T2.tv_sec-T1.tv_sec)*1000000+(T2.tv_usec - T1.tv_usec);
    //printf("time is %d usecond\n",uT);
```

`deltaT`表示排序部分的运行时间。

（二）快排

快排部分整体代码与归并排序类似，只是简化了`MY_ARRAY`和改变了函数`mySort`。

```
typedef struct{
    int first;
    int last;
} MY_ARRAY;

void mySort(MY_ARRAY *array){
    quicksort(a,array->first,array->last);
}
```

其余的`void quicksort( int a[], int left, int right );`和`int partition( int a[], int left, int right );`都是在之前的课程内容中详细讲解过的。

四、实验结果分析

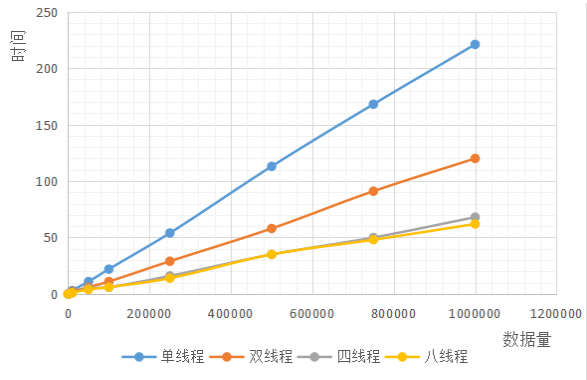
（一）归并排序

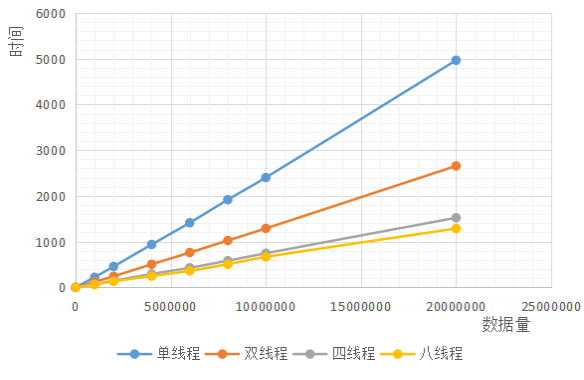
数据量	10	100	1000	10000	100000	1000000	10000000
单线程时间	0	1	1	3	22	227	2386
双线程时间	0	0	1	2	11	127	1333
四线程时间	0	0	0	1	8	74	772
八线程时间	0	0	0	1	6	68	609

先大致取了几个数量级的数据进行测试，发现数据量较小时，单线程和双线程的效率差不多，于是细分各个范围：

数据量	1W	5W	10W	25W	50W	75W	100W
单线程时间	3	11	22	54	113	168	221
双线程时间	2	6	11	29	58	91	120
四线程时间	1	5	6	16	35	50	68
八线程时间	1	4	6	14	35	48	62

数据量	100W	200W	400W	600W	800W	1000W	2000W
单线程时间	222	457	938	1412	1917	2402	4967
双线程时间	119	243	505	761	1025	1289	2656
四线程时间	69	145	293	428	583	746	1522
八线程时间	62	132	247	362	507	666	1288





- 从图中可以看出，双线程的确比单线程的效率更高，运行速度大致是两倍左右；
- 从各组的测试结果可以看出，归并排序较为稳定，随着数据量的增加其性能并未发生大的波动。
-

（二）快速排序

数据量	10	100	1000	10000	100000	1000000	10000000
单线程时间	0	1	1	1~2	12	118	1178
双线程时间	0	1	1	1	2~10	3-120	120-1300
四线程时间	0	1	1	1	2-10	7-56	56-829

- 从测试的结果可以看出，单线程的快排比单线程的归并要快不少，性能大致是归并排序的两倍左右。
- 双线程快排十分不稳定，这也是因为快速排序本身的算法速度和随机产生的数组有着比较大的关系。
- 虽然多线程快排不稳定，波动性比较大，但整体性能仍然优于单线程快排，且远远快于归并排序算法。
- 多线程快排不稳定的原因在于`int pivot = partition(a,0,array_length-1);`找到的`pivot`波动过大
- 如果换一种分划方式，也可以呈现出线性关系。

（三）二者比较

- 从二者最终结果的比较可以看出，一般情况下，快速排序的性能都远远优于归并排序，虽然快排并不十分稳定，但较多时候其速度都比归并算法要快。
- 快排比归并排序快的一个原因便是快排查找的常量比归并小，归并的比较次数虽然小于快排，但是其移动次数多于快排，在C语言的算法中两者的性能大致是两倍的关系。
- 多线程和单线程相比，在数据量较大的情况下，采用多线程便能大大缩短排序所需的时间。
- 本实验中采用的是多线程，并且是在线性的情况下，并没有采用更多线程运行的方法；根据上面的数据进行合理的推测，适当地增加线程数量，可以进一步提高程序的性能，缩短处理时间。

五、实验心得

（一）排序算法

经过本次实验，对归并排序和快速排序这两种算法有了进一步的认识。发现快速排序整体情况下性能优于快速排序，前者的运行速度大致是后者的两倍。

在时间复杂度的分析中，有如下的情况：

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	稳定性
归并排序	O(nlog2n)	O(nlog2n)	O(nlog2n)	稳定
快速排序	O(nlog2n)	O(n^2)	O(nlog2n)	不稳定

归并排序的复杂度总是O（nlog2n），而快速排序虽然平均复杂度是O（nlog2n），其最坏的情况下会达到O(n^2)，但是在绝大多数情况下，快排遇到的都是平均情况，即最佳情况，只有极个别的情况下才会遇到最坏情况，并且快排需要寻找的常量比归并排序小，因此快排更快一些。

随着数据量的增加，归并排序后期需要进行的合并操作所花费的时间越来越大，而合并操作对整体的效率有着很大的影响，包括其所需要进行的大量赋值操作；而快排不需要进行专门的合并，因此数据量越大，快排的优势便越来越明显。

各个算法并无优劣之分，在不同的情况下，选择最合适的算法才是最重要的。

（二）多线程

多线程中包含的内容很多，本次实验只是用了最浅层次的知识，将一个随机数组均分成及部分，相应创建多个线程分别处理这几个部分数据，因此工程量并不大。本次实验我也仅仅考虑了最简单的情况，并没有过多地去根据数组的特征去考虑该如何划分数据，又如何创建线程去处理。

本次实验中我对多线程的学习和掌握只是皮毛罢了，更多的方法还有待挖掘，也需要我们在日常生活中不断尝试和积累，去比较各个方法的优势和不足，才能在遇到问题时选择最合适的方法求解。

在目前这个数据量日益增加的社会，效率和稳定性都是数据处理所必需的，多线程带来了更多更好的解决方法，其优势在本次实验中的体现大概也只是冰山一角，我相信多线程和并行的处理方式在日后一定会发挥更大的作用。