操作系统课程设计 Project 3
Multithreaded Sorting Application
& Fork-Join Sorting Application

姓名：郭倩昀
班级：F1903303
学号：519021910095
Email：guoqianyun@sjtu.edu.cn

2021 年 5 月 21 日

目录

1 Multithreaded Sorting Application

1.1 实验内容与目标

本实验需要利用 C 语言设计多线程排序应用程序，主要步骤如下

- 创建两个排序线程完成部分排序
- 创建归并排序线程完成两个排序线程结果的归并

1.2 实验过程及步骤

- 设计排序线程内的排序方法
  本人直接选用 C 语言 <stdlib.h> 带有的快速排序函数 qsort()，不过重新定义了大小比较函数 qsort_compare。
- 设计归并排序线程内的归并方法
  在函数 msorting 里设计归并排序。msorting 根据传入的下标参数定位两段排序结果的下标，然后进行比较和归并。
- 生成待排序数组
  main 函数开始先设置随机数种子，让用户输入数组长度并选择是否随机生成数组，然后按相应方式生成待排序数组。
- 排序线程
  根据数组大小创建相应的下标参数，创建两个排序线程 sorting_thread，用函数 pthread_create 传入线程，参数和排序函数 qsorting，最后用函数 pthread_join 等待线程完成。
- 归并排序线程
  同样根据数组大小创建相应归并的下标参数,创建归并排序线程 merging_thread,用函数 pthread_create 传入线程，参数和排序函数 msorting，最后用函数 pthread_join 等待线程完成。
- 其他注意事项
  对于每次 pthread_create，pthread_join 等关键步骤，用 error 变量记录完成情况，出现异常及时报错。另外，在为数组分配空间后在结束前要及时释放分配空间。

1.3 实验代码

multithreaded_sorting.c

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

int n;
int *array;
int *rst;
//quick sort for sorting_thread
int qsort_compare(const int*a, const int *b)
```

```
{
    return *a-*b;
}

void* qsorting(int *arg)
{
    if(arg[1]-arg[0]<0) return NULL;//invalid index
    qsort(array+arg[0],arg[1]-arg[0]+1, sizeof(int), qsort_compare);
    return NULL;
}
//mergesort for merging_thread
void* msorting(int *arg)
{
    int rst_dex=arg[0];
    int dex0=arg[0];
    int dex1=arg[1]+1;
    while(dex0<=arg[1]&&dex1<=arg[2])
    {
        if(array[dex0]<=array[dex1])
        {
            rst[rst_dex]=array[dex0];
            rst_dex++;
            dex0++;
        }
        else
        {
            rst[rst_dex]=array[dex1];
            rst_dex++;
            dex1++;
        }
    }
    while(dex0<=arg[1])
    {
        rst[rst_dex]=array[dex0];
        rst_dex++;
        dex0++;
    }
    while(dex1<=arg[2])
    {
        rst[rst_dex]=array[dex1];
        rst_dex++;
        dex1++;
    }
}

int main(void)
{
    //for random number
    srand((unsigned)time(NULL));
    int error=0;
    printf("input the array length n (0<=n<=10000): ");
    scanf("%d", &n);

    if(n<0||n>10000)
    {
        printf("ERROR:n is out of range\n");
        exit(1);
```

```
    }


    array=(int *)malloc(n*sizeof(int));
    rst=(int *)malloc(n*sizeof(int));

    char random[2];
    printf("generate the random elements?(Y/N):\n");
    scanf("%s", random);
    if(random[0]=='Y')  //generate random number
    {
        for (int i = 0; i < n; ++ i)
        {
            array[i] = rand() % 1000;
            printf("%d ",array[i]);
        }
        printf("\n");
    }
    else    //read in array
    {
        printf("input the array elements:\n");
        for(int i=0;i<n;i++)
            scanf("%d",&array[i]);
    }

    //Range parameter for sorting
    int parameters0[2];
    int parameters1[2];
    parameters0[0]=0;
    parameters0[1]=n/2;
    parameters1[0]=n/2+1;
    parameters1[1]=n-1;

    //create 2 sorting_thread
    pthread_t sorting_thread[2];
    error=pthread_create(&sorting_thread[0],NULL,qsorting,&parameters0);
    if(error)
    {
        printf("ERROR:create thread failed\n");
        exit(1);
    }
    error=pthread_create(&sorting_thread[1],NULL,qsorting,&parameters1);
    if(error)
    {
        printf("ERROR:create thread failed\n");
        exit(1);
    }
    error=pthread_join(sorting_thread[0],NULL);
    if(error)
    {
        printf("ERROR:thread join failed\n");
        exit(1);
    }
    error=pthread_join(sorting_thread[1],NULL);
    if(error)
    {
        printf("ERROR:thread join failed\n");
```

```
        exit(1);
    }


    //Range parameter for merging
    int mergeparam[3];
    mergeparam[0]=0;
    mergeparam[1]=n/2;
    mergeparam[2]=n-1;

    //create merging_thread
    pthread_t merging_thread;
    error=pthread_create(&merging_thread,NULL,msorting,&mergeparam);
    if(error)
    {
        printf("ERROR:create thread failed\n");
        exit(1);
    }
    error=pthread_join(merging_thread,NULL);
    if(error)
    {
        printf("ERROR:thread join failed\n");
        exit(1);
    }

    printf("after sorting: \n");
    for(int i=0;i<n;i++)
        printf("%d ",rst[i]);
    printf("\n");

    //free
    free(array);
    free(rst);

    return 0;

}
```

1.4 实验测试

- Multithreaded Sorting 测试 (图 1)

  测试指令如下

```
gcc multithreaded_sorting.c -o ./sort -g -lpthread
./sort
100
Y
```

  首先用 gcc 编译 multithreaded_sorting.c 文件，生成 sort 可执行文件，加上-g -lpthread 表示使用 pthread API。输入./sort 开始执行，输入数组长度为 100，然后输入 Y 选择随机生成待排序数组，排序完成结果如图 1。

```
gqy@gqy-VirtualBox:~/os_proj3$ ./sort
input the array length n (0<=n<=10000): 100
generate the random elements?(Y/N):
Y
566 873 337 157 453 721 67 464 398 975 267 323 808 437 888 241 283 596 501 901 299 94
2 70 75 653 78 949 384 611 604 794 178 829 131 687 282 852 754 99 251 729 366 574 889
 156 814 131 791 762 984 693 414 927 115 489 580 193 438 316 804 394 110 334 223 241
21 858 445 775 957 696 857 675 622 98 183 436 581 975 551 566 20 965 493 135 454 73 3
28 892 741 484 638 851 171 213 444 544 71 889 672
after sorting:
20 21 67 70 71 73 75 78 98 99 110 115 131 131 135 156 157 171 178 183 193 213 223 241
 241 251 267 282 283 299 316 323 328 334 337 366 384 394 398 414 436 437 438 444 445
453 454 464 484 489 493 501 544 551 566 566 574 580 581 596 604 611 622 638 653 672 6
75 687 693 696 721 729 741 754 762 775 791 794 804 808 814 829 851 852 857 858 873 88
8 889 889 892 901 927 942 949 957 965 975 975 984
gqy@gqy-VirtualBox:~/os_proj3$
```

图 1: Multithreaded Sorting 测试

2 Fork-Join Sorting Application

2.1 实验内容与目标

本实验需要利用 java 语言设计 Fork-Join 排序应用程序

- 设计归并排序版本的分治排序应用程序
- 设计快速排序版本的分治排序应用程序

2.2 实验过程及步骤

归并排序

- 创造 Mergesort 类
  参考书本 4.5.2 中 Figure4.18 示例创造一个延伸 RecursiveAction 的 Mergesort 类，这里选择将 THRESHOLD 设置为 10。
- 修改 compute() 函数
  当数组大小小于 THRESHOLD 时候，直接采用冒泡排序完成排序工作。否则，创建新的 Mergesort 对象 leftTask 和 rightTask 传入相应的参数，分别调用 fork()，join()，然后将 leftTask 和 rightTask 排序好的结果进行比较与归并，完成整体数组的排序。
- 生成待排序数组
  main 函数中让用户输入数组长度并选择是否随机生成数组，然后按相应方式生成待排序数组。
- 创建线程池执行任务
  创建 ForkJoinPool 类型的 pool，Mergesort 类型的 task 对待排序数组创建任务，然后 pool 调用 invoke 执行任务对数组进行排序然后输出排序后数组。

快速排序（与归并排序类似）

- 创造 Quicksort 类
  参考书本 4.5.2 中 Figure4.18 示例创造一个延伸 RecursiveAction 的 Quicksort 类，这里选择将 THRESHOLD 设置为 10。

- 修改 compute() 函数
  当数组大小小于 THRESHOLD 时候，直接采用冒泡排序完成排序工作。否则，选中数组第一个元素作为关键值用快速排序的方式将数组分成两部分，创建新的 Quicksort 对象 leftTask 和 rightTask 传入相应的参数，分别调用 fork()，join()。

- 生成待排序数组
  main 函数中让用户输入数组长度并选择是否随机生成数组，然后按相应方式生成待排序数组。

- 创建线程池执行任务
  创建 ForkJoinPool 类型的 pool，Quicksort 类型的 task 对待排序数组创建任务，然后 pool 调用 invoke 执行任务对数组进行排序然后输出排序后数组。

2.3 实验代码

Mergesort.java

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.*;

public class Mergesort extends RecursiveAction {
    static final int THRESHOLD = 10;

    private int begin;
    private int end;
    private int[] array;

    public Mergesort(int begin, int end, int[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }

    protected void compute()
    {
        if (end - begin < THRESHOLD) {
            //Bubble Sort
            for (int i = end; i >= begin + 1; -- i)
            {
                for (int j = begin; j < i; ++ j)
                {
                    if (array[j]>(array[j + 1])) {
                        int tmp = array[j];
                        array[j] = array[j + 1];
                        array[j + 1] = tmp;
                    }
                }
            }
        }
        else
        {
            int mid = begin + (end - begin) / 2;
            //new task
            Mergesort leftTask = new Mergesort(begin, mid, array);
```

```
            Mergesort rightTask = new Mergesort(mid + 1, end, array);
            //fork and join
            leftTask.fork();
            rightTask.fork();

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

            int[] tmp = new int [end - begin + 1];
            //merge the 2 halves
            int dex1 = begin, dex2 = mid + 1, newdex = 0;
            while (dex1 <= mid && dex2 <= end) {
                if (array[dex1]<=array[dex2]) tmp[newdex++] = array[dex1++];
                else tmp[newdex++] = array[dex2++];
            }
            while (dex1 <= mid) tmp[newdex++] = array[dex1++];
            while (dex2 <= end) tmp[newdex++] = array[dex2++];

            for (int i = 0; i < newdex; ++ i)
                array[i + begin] = tmp[i];
        }
    }

    public static void main(String[] args)
    {
        ForkJoinPool pool = new ForkJoinPool();
        Scanner sc = new Scanner(System.in);

        System.out.print("input the array length n (0<=n<=10000): ");
        int n = sc.nextInt();
        if (n < 0 || n > 10000)
        {
            System.out.println("ERROR:n is out of range");
            System.exit(1);
        }

        int[] array = new int[n];

        System.out.print("generate the random elements?(Y/N): ");
        char opt = sc.next().charAt(0);
        if (opt == 'Y')
        {
            java.util.Random rand = new java.util.Random();
            for (int i = 0; i < n; ++ i)
                array[i] = rand.nextInt(1000);
            System.out.print("The original array: ");
            System.out.println(Arrays.toString(array));
        }
        else
        {
            System.out.println("input the array elements: ");
            for (int i = 0; i < n; ++ i)
                array[i] = sc.nextInt();
        }


        Mergesort task = new Mergesort(0, n - 1, array);
```

```
        pool.invoke(task);
        System.out.print("after sorting: ");
        System.out.println(Arrays.toString(array));
    }
}
```

Quicksort.java

```
import java.util.Arrays;
import java.util.Scanner;
import java.util.concurrent.*;

public class Quicksort extends RecursiveAction {
    static final int THRESHOLD = 10;

    private int begin;
    private int end;
    private int[] array;

    public Quicksort(int begin, int end, int[] array) {
        this.begin = begin;
        this.end = end;
        this.array = array;
    }

    protected void compute() {
        if (end - begin < THRESHOLD) {
            //Bubble Sort
            for (int i = end; i >= begin + 1; -- i)
            {
                for (int j = begin; j < i; ++ j)
                {
                    if (array[j]>(array[j + 1])) {
                        int tmp = array[j];
                        array[j] = array[j + 1];
                        array[j + 1] = tmp;
                    }
                }
            }
        }
        else
        {
            //quick sort
            int pivot = array[begin];
            int low = begin, high = end;
            while (low < high) {
                while (low < high && array[high]>=pivot) -- high;
                if (low < high) array[low ++] = array[high];
                while (low < high && array[low]<=pivot) ++ low;
                if (low < high) array[high --] = array[low];
            }
            array[low] = pivot;
            //new task
            Quicksort leftTask = new Quicksort(begin, low - 1, array);
            Quicksort rightTask = new Quicksort(low + 1, end, array);
            //fork and join
```

```
            leftTask.fork();
            rightTask.fork();

            leftTask.join();
            rightTask.join();
        }
    }

    public static void main(String[] args) {
        ForkJoinPool pool = new ForkJoinPool();
        Scanner sc = new Scanner(System.in);

        System.out.print("input the array length n (0<=n<=10000): ");
        int n = sc.nextInt();
        if (n < 0 || n > 10000)
        {
            System.out.println("ERROR:n is out of range");
            System.exit(1);
        }

        int[] array = new int[n];

        System.out.print("generate the random elements?(Y/N): ");
        char opt = sc.next().charAt(0);
        if (opt == 'Y')
        {
            java.util.Random rand = new java.util.Random();
            for (int i = 0; i < n; ++ i)
                array[i] = rand.nextInt(1000);
            System.out.print("The original array: ");
            System.out.println(Arrays.toString(array));
        }
        else
        {
            System.out.println("input the array elements: ");
            for (int i = 0; i < n; ++ i)
                array[i] = sc.nextInt();
        }


        Quicksort task = new Quicksort(0, n - 1, array);
        pool.invoke(task);
        System.out.print("after sorting: ");
        System.out.println(Arrays.toString(array));
    }
}
```

2.4 实验测试

- Mergesort.java 测试 (图 2) 测试指令如下

```
javac Mergesort.java
java Mergesort
100
Y
```

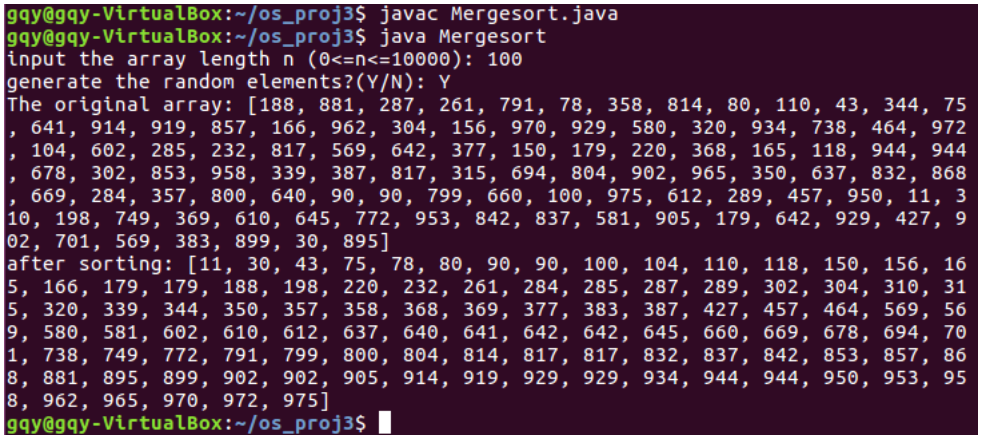

图 2: Mergesort.java 测试

首先 javac Mergesort.java 编译文件，输入 java Mergesort 执行，输入数组元素个数 100，输入 Y 自动生成待排序数组，测试结果如图 2。

- Quicksort.java 测试 (图 3)

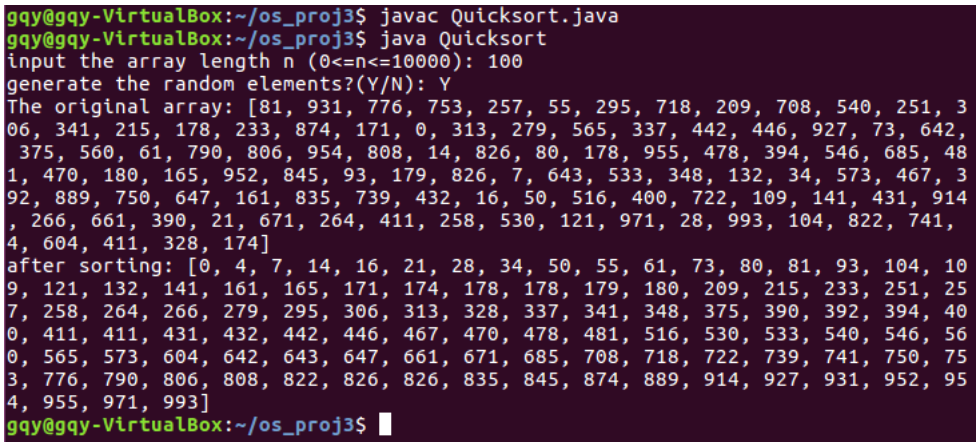

图 3: Quicksort.java 测试

测试指令如下

```
javac Quicksort.java
java Quicksort
100
Y
```

首先 javac Quicksort.java 编译文件，输入 java Quicksort 执行，输入数组元素个数 100，输入 Y 自动生成待排序数组，测试结果如图 3。

3 Conclusion

3.1 问题与解决方案

本次 project3 的 Multithreaded Sorting Application 部分比较顺利，主要是掌握 pthread_create 和 pthread_thread 函数的使用，另外在分多线程的时候需要注意传入分任务的数组下标。

project3 的另一个部分用 java 设计 Fork-Join Sorting Application 的时候，由于是新的虚拟机，首先需要配置 java 环境。另外由于先前对 java 语句没有过多的了解，在进行 project 的时候首先要熟悉一些必要的 java 语句比如输入输出生成随机数等，并且仔细参考书本的示例，进行多次尝试才最后完成了该部分的实验。

3.2 实验心得

本次 project3 让我深入体验了多线程编程，第一次使用 java 语言完成项目，还接触了许多新的函数工具，将多线程编程的理论知识很好地应用到实践中，在问题探索中一步步完成整个实验。总的来说本次 project 很好地锻炼了动手能力，并从中习得面对新事物需要勇于尝试，敢于试错，才会不断有新的收获。