

Project 3

秦啸涵 5210121910604

多线程校验数独

在这部分中，要求使用 `pthread` 库创建多线程校验一个 9×9 的数独是否正确。我为每一行，每一列和每一个 3×3 的格子分别创建一个子线程，共创建27个线程用于校验正确性。

首先是用校验的函数(提前封装好方便调用):

```
1  bool checkrow(int row)
2  {
3      int check[9] = {0};
4      for (int i = 0; i < 9; i++)
5      {
6          if (check[A[row][i] - 1] == 1)
7              {
8                  flag = false;
9                  return false;
10             }
11             else
12             {
13                 check[A[row][i] - 1] = 1;
14             }
15         }
16         return true;
17     }
18     bool checkcol(int col)
19     {
20         int check[9] = {0};
21         for (int i = 0; i < 9; i++)
22         {
23             if (check[A[i][col] - 1] == 1)
24                 {
25                     flag = false;
26                     return false;
27                 }
```

```

28         else
29         {
30             check[A[i][col] - 1] = 1;
31         }
32     }
33     return true;
34 }
35
36 bool checkgrid(int row, int col)
37 {
38     int check[9] = {0};
39     for (int i = row; i < row + 3; i++)
40     {
41         for (int j = col; j < col + 3; j++)
42         {
43             if (check[A[i][j] - 1] == 1)
44             {
45                 flag = false;
46                 return false;
47             }
48             else
49             {
50                 check[A[i][j] - 1] = 1;
51             }
52         }
53     }
54     return true;
55 }

```

其次是用于传递给线程的函数指针：

```

1 // 检查数独的所有行的线程函数
2 void *checkrow_thread(void *param)
3 {
4     int row = (int)param;
5     if (checkrow(row))
6     {
7         printf("\033[1;32mrow %d is legal\033[0m\n", row);
8     }
9     else

```

```

10     {
11         printf("\033[1;31mrow %d is illegal\033[0m\n", row);
12     }
13     pthread_exit(0);
14 }
15 //检查数独的所有列的线程函数
16 void *checkcol_thread(void *param)
17 {
18     int col = (int)param;
19     if (checkcol(col))
20     {
21         printf("\033[1;32mcol %d is legal\033[0m\n", col);
22     }
23     else
24     {
25         printf("\033[1;31mcol %d is illegal\033[0m\n", col);
26     }
27     pthread_exit(0);
28 }
29 //检查数独的所有九宫格的线程函数
30 void *checkgrid_thread(void *param)
31 {
32     int grid = (int)param;
33     int row = grid / 3 * 3;
34     int col = grid % 3 * 3;
35     if (checkgrid(row, col))
36     {
37         printf("\033[1;32mgrid %d is legal\033[0m\n", grid);
38     }
39     else
40     {
41         printf("\033[1;31mgrid %d is illegal\033[0m\n", grid);
42     }
43     pthread_exit(0);
44 }

```

在 `main` 函数中，使用 `pthread_create()` 创建子线程并调用 `pthread_join()` 等待线程结束：

```

1  pthread_t tid[27];
2  pthread_attr_t attr;
3  pthread_attr_init(&attr);
4  //创建检查数独的所有行的线程
5  for (int i = 0; i < 9; i++)
6  {
7      pthread_create(&tid[i], &attr, checkrow_thread, (void *)i);
8  }
9  //创建检查数独的所有列的线程
10 for (int i = 0; i < 9; i++)
11 {
12     pthread_create(&tid[i + 9], &attr, checkcol_thread, (void
13 *)i);
14 }
15 //创建检查数独的所有九宫格的线程
16 for (int i = 0; i < 9; i++)
17 {
18     pthread_create(&tid[i + 18], &attr, checkgrid_thread, (void
19 *)i);
20 }
21 //等待所有线程结束
22 for (int i = 0; i < 27; i++)
23 {
24     pthread_join(tid[i], NULL);
25 }

```

运行结果如下：

```
in1.txt X
ch4 > in1.txt
1 6 2 4 5 3 9 1 8 7
2 5 1 9 7 2 8 6 3 4
3 8 3 7 6 1 4 2 9 5
4 1 4 3 8 6 5 7 2 9
5 9 5 8 2 4 7 3 6 1
6 7 6 2 3 9 1 4 5 8
7 3 7 1 9 5 6 8 4 2
8 4 9 6 1 8 2 5 7 3
9 2 8 5 4 7 3 9 1 6
```

```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ ./sudoku < in1.txt
row 2 is legal
row 3 is legal
row 5 is legal
row 4 is legal
row 6 is legal
row 7 is legal
col 1 is legal
col 2 is legal
col 3 is legal
col 4 is legal
col 5 is legal
col 6 is legal
col 7 is legal
col 8 is legal
row 1 is legal
row 8 is legal
grid 1 is legal
grid 6 is legal
grid 7 is legal
grid 8 is legal
grid 5 is legal
row 0 is legal
grid 2 is legal
grid 4 is legal
grid 3 is legal
grid 0 is legal
col 8 is legal
This sudoku is legal
```

```
in2.txt X
ch4 > in2.txt
1 1 2 3 4 5 6 7 8 9
2 1 2 3 4 5 6 7 8 9
3 1 2 3 4 5 6 7 8 9
4 1 2 3 4 5 6 7 8 9
5 1 2 3 4 5 6 7 8 9
6 1 2 3 4 5 6 7 8 9
7 1 2 3 4 5 6 7 8 9
8 1 2 3 4 5 6 7 8 9
9 1 2 3 4 5 6 7 8 9
```

```
jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ ./sudoku < in2.txt
row 0 is legal
row 3 is legal
row 4 is legal
col 1 is illegal
col 3 is illegal
row 8 is legal
col 4 is illegal
col 5 is illegal
col 6 is illegal
col 7 is illegal
grid 1 is illegal
row 5 is legal
row 2 is legal
grid 6 is illegal
grid 8 is illegal
grid 3 is illegal
grid 0 is illegal
col 8 is illegal
col 2 is illegal
grid 2 is illegal
grid 7 is illegal
col 0 is illegal
row 7 is legal
row 6 is legal
row 1 is legal
grid 4 is illegal
grid 5 is illegal
This sudoku is illegal
```

使用pthread库进行多线程排序

定义两个函数指针 `sort()` 和 `merge()`，分别用于做子数组的排序和最终的归并

```
1 void *sort(void *args)
2 {
3     struct pthread_sort *arg = (struct pthread_sort *)args;
4     int start = arg->start;
5     int end = arg->end;
6     qsort(arr + start, end - start + 1, sizeof(int), compare);
7     // for (int i = start; i <= end; i++)
8     // {
9     //     printf("%d ", arr[i]);
10    // }
11    // printf("\n");
```

```

12     pthread_exit(NULL);
13 }
14
15 void *merge(void *args)
16 {
17     struct pthread_sort *arg = (struct pthread_sort *)args;
18     int start = arg->start;
19     int end = arg->end;
20     int mid = (start + end + 1) / 2;
21     int i = start, j = mid + 1, k = start;
22     while (i ≤ mid && j ≤ end)
23     {
24         if (arr[i] < arr[j])
25         {
26             sorted[k++] = arr[i++];
27         }
28         else
29         {
30             sorted[k++] = arr[j++];
31         }
32     }
33     while (i ≤ mid)
34     {
35         sorted[k++] = arr[i++];
36     }
37     while (j ≤ end)
38     {
39         sorted[k++] = arr[j++];
40     }
41     pthread_exit(NULL);
42 }

```

使用 `pthread_create()` 和 `pthread_join()` 完成多线程排序，在这里由于 `merge` 需要等待两个排序的子线程完成后才能进行，因此需要先执行排序线程的 `join` 再进行 `merge`

```

1  pthread_t tid[3];
2  for (int i = 0; i < 2; i++)
3  {
4      pthread_create(&tid[i], NULL, sort, &args[i]);
5  }
6  for (int i = 0; i < 2; i++)
7  {
8      pthread_join(tid[i], NULL);
9  }
10 pthread_create(&tid[2], NULL, merge, &args[2]);
11 pthread_join(tid[2], NULL);

```

运行结果:

```

● jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ ./mergesort
Enter the number of elements:
6
Enter the elements:
1 5 2 4 3 7
After multithreaded sort, the sorted array is:
1 2 3 4 5 7

```

java fork-join API实现mergesort和quicksort

quicksort

使用 `java` 实现 `quicksort` 类。在fork/join框架中实现排序，任务被分割成更小的子任务，每个子任务都需要实现 `compute()` 方法来执行具体的任务。在 `quicksort` 类中，我们定义的 `compute()` 为:

```

1  protected void compute() {
2      if (left < right) {
3          int pivotIndex = left + (right - left) / 2;
4          pivotIndex = partition(pivotIndex);
5          invokeAll(new quicksort(data, left, pivotIndex - 1),
6                  new quicksort(data, pivotIndex + 1, right));
7      }
8  }

```

按照快速排序的逻辑，选定分割点后执行 `partition()` 将数组分为两个子数组，之后调用 `invokeAll()` 并行地执行左右分区的快速排序。

在 `main` 函数中，我们新建一个线程池，之后调用 `invoke()` 进行并行排序即可：

```
1 ForkJoinPool pool = new ForkJoinPool();
2 quicksort sorter = new quicksort(data, 0, data.length - 1);
3 pool.invoke(sorter);
```

运行结果如下：

```
• jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ javac quicksort.java
• jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ java quicksort
Enter the size of the array:
7
Enter the elements of the array:
1 7 6 4 5 3 9
The sorted array is:
1 3 4 5 6 7 9
```

mergesort

与 `quicksort` 方法基本相同，只需更改实现逻辑

```
1 protected void compute() {
2     if (left < right) {
3         if (right - left < threshold) {
4             insertionsort(data, left, right);
5         } else {
6             int mid = left + (right - left) / 2;
7             invokeAll(new mergesort(data, left, mid,
8                                     threshold),
9                       new mergesort(data, mid + 1, right,
10                                    threshold));
11             merge(data, left, mid, right);
12         }
13     }
14 }
```


对于 `compute()` 函数，我们定义一个常量 `threshold`，当要排序的数组小于这个常量时，不再继续分割子任务而是调用 `insertionsort()` 直接进行插入排序，否则就将数组分割为两部分，调用 `invokeAll()` 并行执行归并排序，并将排序好的数组进行 `merge` 得到最终结果。

`main()` 函数中实现如下：

```
1 ForkJoinPool pool = new ForkJoinPool();
2 mergesort sorter = new mergesort(data, 0, data.length - 1, 4);
3 pool.invoke(sorter);
```

运行结果如下：

```
● jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ javac mergesort.java
● jianke@ubuntu:~/Desktop/final-src-osc10e/ch4$ java mergesort
Enter the size of the array:
7
Enter the elements of the array:
1 5 7 4 2 6 3
The sorted array is:
1 2 3 4 5 6 7
```