CS307 Project3: Multi-threaded Sorting Application & Fork-Join Sorting Application

Junjie Wang 517021910093

May 23, 2019

1 Programming Thoughts

1.1 Part1: Multi-threaded Sorting

In this part we are going to split a list into two halves and create sort threads to sort these two halves. Then we need to use merge thread to merge these two halves. This idea is a bit similiar to the mergeSort algorithm. Another thing worth attention is that to pass the parameters to the threads, we better define a **struct**(in this example, consists of start idx and end idx).

1.2 Part2: Fork-Join Sorting

In this part we have to implement quickSort and mergeSort algorithms using Java multi-threading APIs. Both algorithms here consist of the **dividing stage**(where we use new threads to accomplish the sorting of two halves.) and **conquering stage**(where we set a threshold, smaller than we will simply use sorting algorithm like insertionSort.)

2 Execution Results And Snapshots

The execution results are as follows:

```
dreamboy@Elon_Mask:~/OSProjects/project3$ ./sort
Before sorting, the input array is:
80 79 78 77 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 3
7 36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

After the sorting, the output array is:
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 4
7 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
dreamboy@Elon_Mask:~/OSProjects/project3$
```

Figure 1: Multi-threaded Sorting

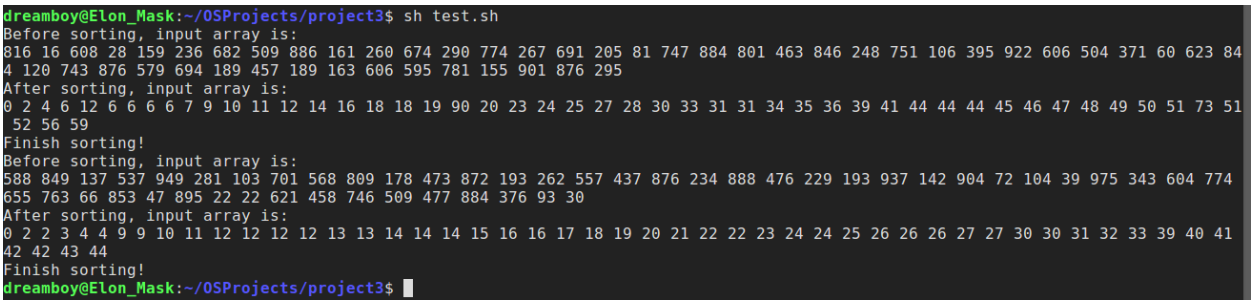

Figure 2: Multi-threaded Sorting

3 Code Explanation

First we show the parameters passing for the first part.

```
/* for multiple arguments to be passed
* we have to define a struct */
typedef struct _param{
   int start;
   int end;
} funcParam;
...
/* setting the parameters */
params[0].start = 0;  params[0].end = mid;
pthread_attr_init(&attr[0]);
pthread_create(&tid[0], &attr[0], sortThread, &params[0]);
...
/* getting the parameters by type conversion
   and code for the sorting thread */
void *sortThread(void *param){
   funcParam arg = *(funcParam *)param;
   int start = arg.start, end = arg.end;
   int min, tmp, idx;
   for(int i=start;i<end;i++){
      min = in_arr[i];
      idx = i;
      for(int j=i;j<end;j++){
         if(in_arr[j] < min){
         min = in_arr[j];
         idx = j;
      }
      tmp = in_arr[i];
   in_arr[i] = in_arr[idx];
   in_arr[idx] = tmp;
   }
   }
}
```

Then we show the **divide** stage of both quickSort and mergeSort algorithms in Java.

```
/* divide stage for the quickSort algorithm */
int low = begin, high = end;
int val = array[low];
do{
while(low < high && array[high] >= val) high--;
if(low < high) { array[low] = array[high]; low++;}
while(low < high && array[low] <= val) low++;
if(low < high) { array[high] = array[low];high--; }
}while(low!=high);

array[low] = val;

/* from the place we put val */
int mid = low + 1;
quickSortTask left = new quickSortTask(begin, mid - 1, array);
quickSortTask right = new quickSortTask(mid + 1, end, array);

/* create new threads to accomplish the sorting of the two halves */
left.fork();
right.fork();

/* wait until the threads exit */
left.join();
right.join();

/* divide stage for the mergeSort, much simpler*/
int mid = (begin + end) / 2;
mergeSortTask left = new mergeSortTask(begin, mid, array);
mergeSortTask right = new mergeSortTask(mid + 1, end, array);
left.fork();
right.fork();
left.join();
right.join();
```