Lab Report of Project 3
Multi-threaded Sorting Application & Fork-Join Sorting Application

Name: Li Dongyue Student Number: 516030910502
Computer System Engineering (Undergraduate), Fall 2018

November 4, 2018

1 Multi-threaded Sorting Application

In this program, it is required to write a multi-threaded sorting program that works as follows:

- A list of integers is divided into two smaller lists of equal size.
- Two separate threads sort each sublist using a sorting algorithm (here we use **insert sort**).

  The code to start two threads:

```
    char *thread_no = "first";

    pthread_attr_init(&attr); /* get the default attributes */
    pthread_create(&tid,&attr,runner, thread_no); /* create the thread */


    char *thread2 = "second";
    pthread_attr_init(&attr2);
    pthread_create(&tid2,&attr2, runner, thread2);
    /* wait for the thread to exit */
```

MultithreadSort.c

  In each thread, A number `curThread` identifies its sorting range. Each thread sort sublist using a simple sorting algorithm:

```
void sort(int low, int high)
{
    for (int i=low; i<high; i++)
    {
        int tmp = i;
        for (int j=i+1; j<high+1; j++)
        {
            if (array[j] < array[tmp])
                tmp = j;
        }
        int tmpVal = array[tmp];
        array[tmp] = array[i];
        array[i] = tmpVal;
    }
}

void *sortAlgorithm(void *arg)
{
    int curThread = count++;

    int low = curThread*(n/sortingThreadNum);
    int high = (curThread+1)*(n/sortingThreadNum) - 1;
    sort(low, high);
}
```

MultithreadSort.c

- The two sublists are then merged by a third thread into a single sorted list:

```
1
2  /* This is in main() function */
3
4      pthread_t mergingThread;
5      pthread_create(&mergingThread, NULL, merge, NULL);
6      pthread_join(mergingThread, NULL);
7
8  /* This is in main() function */
9
10 void *merge(void *arg)
11 {
12     int index1 = 0, index2 = n/sortingThreadNum;
13     for (int i=0; i<n; i++)
14     {
15         if ( index1 < n/sortingThreadNum && index2 < n)
16         {
17             if (array[index1] < array[index2]) resArray[i] = array[index1++];
18             else resArray[i] = array[index2++];
19         }
20         else
21         {
22             if (index1 < n/sortingThreadNum) resArray[i] = array[index1++];
23             if (index2 < n) resArray[i] = array[index2++];
24         }
25     }
26 }
```

MultithreadSort.c

2 Fork-Join Sorting Application

In this program, it is required to implement the preceding project using Java's fork-join parallelism API. This project will be developed in two different versions. Each version will implement a different divide-and-conquer sorting algorithm:

- Quicksort
- Mergesort

2.1 Fork-Join API

Implementing Java Fork-Join API, we extends the `RecursiveAction` class and redefine its `compute()` method:

```
1  /**
2   * Extends RecursiveAction.
3   * Notice that the compute method does not return anything.
4   */
5  class MergeSort extends RecursiveAction {
6
7      /**
8       * Inherited from RecursiveAction.
9       * Compare it with the run method of a Thread.
10      */
11     @Override
12     protected void compute() {
13         /* The override function */
14     }
15 }
```

ForkJoinArraySort.java

In `main()` function, we instantiate a `ForkJoinPool`. When we need to run a new thread, we use the function `pool.invoke(RecursiveAction)` to start the running:

```
1
2  /* In main() */
3  ForkJoinPool pool = new ForkJoinPool();
4
5  QuickSort quick = new QuickSort(array1, 0, array1.length - 1);
6
7  pool.invoke(quick); // Start execution and wait for result/return
8
```

```
/* In main() */
```

ForkJoinArraySort.java

2.2 Mergesort

To implement mergesort, we just need to override the `compute()` function in a `RecursiveAction` class. Basically, we are splitting the array and run merge sort in each subarray:

```
/**
 * Extends RecursiveAction.
 * Notice that the compute method does not return anything.
 */
class MergeSort extends RecursiveAction {
    private int array[];
    private int left;
    private int right;

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

    /**
     * Inherited from RecursiveAction.
     * Compare it with the run method of a Thread.
     */
    @Override
    protected void compute() {
        if (left < right) {
            int mid = (left + right) / 2;
            RecursiveAction leftSort = new MergeSort(array, left, mid);
            RecursiveAction rightSort = new MergeSort(array, mid + 1, right);
            invokeAll(leftSort, rightSort);
            merge(left, mid, right);
        }
    }

    /**
     * Merge two parts of an array in sorted manner.
     * @param left  Left side of left array.
     * @param mid   Middle of separation.
     * @param right Right side of right array.
     */
    private void merge(int left, int mid, int right) {
        int temp [] = new int[right - left + 1];
        int x = left;
        int y = mid + 1;
        int z = 0;
        while (x <= mid && y <= right) {
            if (array[x] <= array[y]) {
                temp[z] = array[x];
                z++;
                x++;
            } else {
                temp[z] = array[y];
                z++;
                y++;
            }
        }
        while (y <= right) {
            temp[z++] = array[y++];
        }
        while (x <= mid) {
            temp[z++] = array[x++];
        }

        for (z = 0; z < temp.length; z++) {
            array[left + z] = temp[z];
        }
    }
}
```

ForkJoinArraySort.java

2.3 Quicksort

To implement quicksort, we just need to override the `compute()` function in a `RecursiveAction` class. Basically, we choose a pivot, and we put the number smaller than the pivot to its left and larger ones to its right. With this, we recursively run the left part of the pivot and right part of the pivot on quicksort:

```
1
2  /**
3   * Extends RecursiveAction.
4   * Notice that the compute method does not return anything.
5   */
6  class QuickSort extends RecursiveAction {
7      private int array[];
8      private int left;
9      private int right;
10
11     public QuickSort(int[] array, int left, int right) {
12         this.array = array;
13         this.left = left;
14         this.right = right;
15     }
16
17     /**
18      * Inherited from RecursiveAction.
19      * Compare it with the run method of a Thread.
20      */
21     @Override
22     protected void compute() {
23         if (left < right) {
24             int pivot = array[left];
25             int ll = left+1;
26             for (int i=left + 1; i<=right; i++){
27                 if (array[i] < array[left]){
28                     int tmp = array[i];
29                     array[i] = array[ll];
30                     array[ll] = tmp;
31                     ll+=1;
32                 }
33             }
34             int tmp = array[left];
35             array[left] = array[ll-1];
36             array[ll-1] = tmp;
37             RecursiveAction leftSort = new QuickSort(array, left, ll-2);
38             RecursiveAction rightSort = new QuickSort(array, ll, right);
39             invokeAll(leftSort, rightSort);
40         }
41     }
42
43 }
```

ForkJoinArraySort.java

Whole code is attached to this file.