IN3030/IN4330 Spring 2021 Exam

Eric Jul, May 2021

Question 1: Caching and the Speed of Light

Question 1.1: Speed of Light (1 point)

What is the *exact* speed of light in vacuum in m/s?

Question 1.2: Caching (9 points)

Explain, in 100 words or less, how the speed of light affects the access time of memory.

Question 2: Variable Cyclic Barriers using Semaphores

Question 2.1 Variable Cyclic Buffer Replacement (10 points)

You are to achieve the effect of a cyclic barrier, but instead of using the Java class CyclicBarrier, you must write a Java class `SemCyclicBarrier` that has the functionality of CyclicBarrier.

Submit the program and a brief explanation of it.

Question 2.2 Test Case (10 points)

You are to write a Java program that demonstrates a single, representative test case for the program that you wrote in 2.1. Explain the test that you chose and why you think it shows that your program from 2.1 works – at least for your chosen test case (it does

not have to be comprehensive – just show a typical case). Each thread could, for illustration, print what it does at each step – be sure to include an id of the thread doing the printing.

Hint: You can "schedule" when threads reach the barrier by delaying them using, *e.g.,* `TimeUnit.SECONDS.sleep(10);`

Submit the program and its output and any comments that you might have.

Question 3: Recursive Mergesort

Mergesort is a sorting algorithm that sorts, *e.g.*, an integer array A, by dividing an array of integer elements to be sorted into two parts: a first part and a second part, recursively sorting each part, and then merging the two parts into one sorted array.

A sketch of a sequential `mergesort` of an integer array is given below:

```
class mergesorting {

  public static void merge(int[] left_arr,int[] right_arr, int[] arr,int left_size, int right_size){

     int i=0;
     int l = 0;
     int r = 0;
     //The while loops check the conditions for merging
     while (l<left_size && r<right_size){

        if(left_arr[l]<right_arr[r]){
           arr[i++] = left_arr[l++];
        }
        else {
           arr[i++] = right_arr[r++];
        }
     }
     while (l<left_size) {
        arr[i++] = left_arr[l++];
     }
     while (r<right_size) {
       arr[i++] = right_arr[r++];
     }
  }

  public static void mergeSort(int [] arr, int len){
     if (len < 2) {return;}

     int mid = len / 2;
     int [] left_arr = new int[mid];
     int [] right_arr = new int[len-mid];

    //Dividing array into two and copying into two separate arrays
     int k = 0;
```

```
    for  (int i = 0;i<len;++i) {
       if (i<mid) {
          left_arr[i] = arr[i];
       }
       else {
          right_arr[k] = arr[i];
          k = k+1;
       }
    }
   // Recursively calling the function to divide the subarrays further
    mergeSort(left_arr,mid);
    mergeSort(right_arr,len-mid);
   // Calling the merge method on each subdivision
    merge(left_arr,right_arr,arr,mid,len-mid);
 }

 public static void main(String args[] ) {
      int [] array = {120,1,101,503,57,158,451};
      mergeSort(array,array.length);
      for (int i =0; i< array.length;++i) {
         System.out.print(array[i]+ " ");
      }
   }
}
```

Question 3.1 Parallelizing MergeSort (30 points)

How can `mergesort` be parallelized? Describe the design of a solution that MUST be loyal to the algorithm, *e.g.*, it must still be recursive.

Hint: spend time on describing the parallelization as this is central to the course.

Submit your description.

Question 3.2: Parallel Mergesort (30 points)

Write a Java program implementing your design of a parallel version of `mergesort` from Question 3.1. Strive to have a speedup over the sequential version. Put any added explanation that you have as comments in the code.
Submit the program and its output.

Question 3.3: Parallel Mergesort test case (10 points)

Write a Java program testing your parallel version of mergesort.
Run the program and document your speedup.
You should sort at least 10 million integers. Generate a random content.
Submit the test program and its output.

END OF EXAM