IN3030/IN4330 Spring 2020 Training Exam

Eric Jul, May 2020

Question 1.1: Java Threads Startup

In a Java, there are two main ways to start a thread. Provide sample code of them both. The code must be your own. You may use code that you have used in the obligs as a basis.

Question 1.2: Java Threads Execution on Multicore

Explain why multiple threads executing on a machine with eight cores may be executing at *different speed* despite all cores running at the same clock frequency.

Question 1.3 Java Synchronized

Explain an alternative to using the Java keyword `synchronized`. That is an alternative way to synchronize that achieves the same effect. Give a *short* explanation of how your alternative works.

Question 2: Join using Cyclic Barriers

Question 2.1 Join Replacement

You are to achieve the effect of `join`, but instead of using `join`, you should use a CyclicBarrier. The idea is to modify the `JoinP` Java program given below to NOT use `join`, but instead use a single CyclicBarrier. Below is a program, `JoinP`, and a modified version of the program, called `CyclicBarrierJoinP-XXX`, that achieves this, albeit there are parts missing - indicated by XXXX in the program. Your task is now to write a version of `CyclicBarrierJoinPXXX` where you have replaced all the XXXX with some Java code that makes the program work like the original `JoinP` program. You are welcome to copy the `CyclicBarrierJoinP-XXX` program and then modify it directly.

Provide the resulting program.
Here is the `JoinP` program:

```
import java.util.concurrent.*;
class JoinP {

   public static void main(String[] args) {
      int numberofthreads = 10;
      Thread[] t = new Thread[numberofthreads];

      for (int j = 0; j < numberofthreads; j++) {
         (t[j] = new Thread( new ExThread() )).start();
      }

      try {
         for (int k = 0; k < numberofthreads; k++) t[k].join();
      } catch (Exception e) { return; }

   }

   static class ExThread implements Runnable {

      public void run() {
         try {
            TimeUnit.SECONDS.sleep(10);
         } catch (Exception e) { return;};
      }

   }

}
```

Here is the `CyclicBarrierJoinP-XXX` program:

```
import java.util.concurrent.*;
class CyclicBarrierJoinP {
   static CyclicBarrier cb;
      public static void done() {
         XXXX
      }
   public static void main(String[] args) {
      int numberofthreads = 10;
      Thread[] t = new Thread[numberofthreads];
      cb = XXXX;
      for (int j = 0; j < numberofthreads; j++) {
         (t[j] = new Thread( new ExThread() )).start();
      }
      // try {
      // for (int k = 0; k < numberofthreads; k++) t[k].join();
      // } catch (Exception e) { return; }
      XXXX;
   }
   static class ExThread implements Runnable {
      public void run() {
         try {
            TimeUnit.SECONDS.sleep(10);
         } catch (Exception e) { return;};
         done();
      }
   }
}
```

Question 2.2 Test Case

You are to write a Java program that demonstrates a 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);`

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

Question 3: Insertion Sort

Insertion Sort 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 sorted part and an unsorted part. Initially, the sorted part is merely composed of the first element in the array and the remaining elements of the array are the unsorted part. The algorithm sorts by repeatedly taking the first element of the unsorted part and then inserting that element into its place in the sorted part – thus making room for it by shifting the elements larger than or equal to the chosen element one position.

A sequential `insertionsort` of an integer array is given below:

```
public static void insertsort (int a[], int left, int right) {
   int i,k,t;
   for (k = left+1; k <= right; k++) {
     t = a[k];
     i = k;
     while (a[i-1] > t) {
        a[i] = a[i-1];
        if (--i == left) break;
     }
     a[i] = t;
   } // end k
} // end insertsort
```

Question 3.1 Parallelizing Insertion

How can `insertionsort` be parallelized? Describe the design of a solution that MUST be loyal to the algorithm, *i.e.*, splitting the array into *k* parts that are then merge sorted is not loyal as much of the speedup is gained by using merging, which is much more efficient than insertion sort for large arrays.

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

Question 3.2: Parallel Insertionsort

Write a Java program implementing your design of a parallel version of `insertionsort` 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.
Run the program, and document your speedup.
Submit the program and its output.

Question 4.1 The Speed of Light in Vacuum

The speed of light in vacuum is:
Select one alternative:

1. Approximately 300,000 km/s
2. Approximately 300,000 km/h
3. Approximately 300,000 m/s
4. Approximately 300,000,000 km/s

Question 4.2 The Speed of Electricity in Copper

The speed of electricity in copper cables is related to the speed of light in vacuum. What is the approximate speed of electricity in copper cables?
Select one alternative:

1. The same as the speed of light in vacuum
2. Roughly 60% of the speed of light in vacuum
3. Roughly 80% of the speed of light in vacuum

Question 5.1 Caching

Forty years ago, computers had a single CPU and a physically separate memory. Why was a cache subsequently introduced between the CPU and the memory?
Hint: Include Moore's law in your answer.

END OF EXAM