DISTRIBUTED SYSTEMS

Distributed Systems – Assignment 3

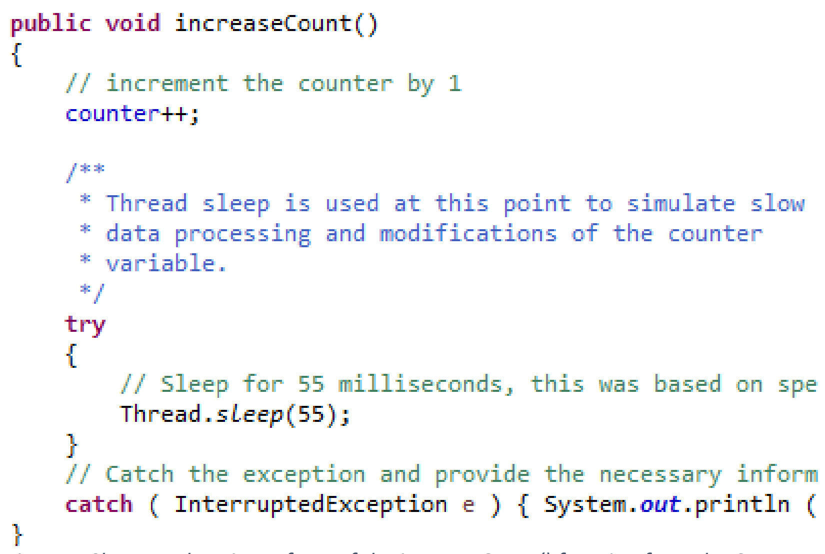
Submitted: November 17, 2014
By: Devan Shah 100428864
Submitted to: Weina ma & Ying Zu

1. A significant concern in distributed systems is conflicts that might arise from unsynchronized requests to a single resource. A very simple example of this concern can be demonstrated by a class that increments a counter through a method. If this class is instantiated by another class that supports multi-threading, and increments the counter N times within its run() method one can demonstrate that when multiple threads of this class are created there are situations when the value of the total count is not what is expected. For example, let's say that the multi-threaded counter object increments count 10 times and 3 threads of this object type are launched. The expected value of count is 30 (10*3) but this will not be the case if the 3 threads are not synchronized.

   For this question create a Counter class with 2 methods *increaseCount()* and *getCount()* that correspondingly increase the value of a counter by 1 and gets the value of the counter. Now define a *CountingThread* class that instantiates a Counter object and implements the Runnable interface (i.e. supports multi-threading). In the *run()* method of *CountingThread* call *increaseCount()* several times. In the main method of *CountingThread* instantiate 3 threads of *CountingThread*. When these threads end get the value of the counter using *getCount()* and print out the result. Note: you might have to put the Counter object to sleep for a few milliseconds in the *increaseCount()* method to get significant synchronization issues. Code this and show that you have issues when you do not use method-level synchronization. Add the method-level synchronization to the code so that the expected counter sum is achieved. [10]

   Answer:

   The implementation of the Counter class is to keep an incrementing count of an integer. On Object creation of Counter the counter (incremental counter) is initialized at 0, which can be incremented by 1 every by calling the function increaseCount() available in the Counter class (or object). The Counter class also contains a getCount() that is used to get the current count value. The Counter class object is shared between all threads in this example, there for any updates to the common variables need to be synchronized to make sure that threads are updating the value correctly. Figure 1 and Figure 3 show code with no use of method-level synchronization and with method-level synchronization.

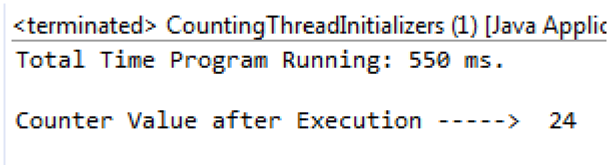


```
public void increaseCount()
{
    // increment the counter by 1
    counter++;

    /**
     * Thread sleep is used at this point to simulate slow
     * data processing and modifications of the counter
     * variable.
     */
    try
    {
        // Sleep for 55 milliseconds, this was based on spe
        Thread.sleep(55);
    }
    // Catch the exception and provide the necessary inform
    catch ( InterruptedException e ) { System.out.println (
}
```

Figure 1 Shows code snippet from of the increaseCount() function from the Counter class without the use of method-level synchronization.

Figure 1 on the left shows the increaseCount() function from the Counter class **without the use of method-level synchronization**, the reason for the thread sleep is to make sure that slow data processing was simulated accurately. With the use of 3 threads and each thread incrementing the shared Counter object the result at the end should be 30 (10*3). But without the use of method-level synchronization this is now the case. Figure 2 shows the out of the code without the use of method-level synchronization.

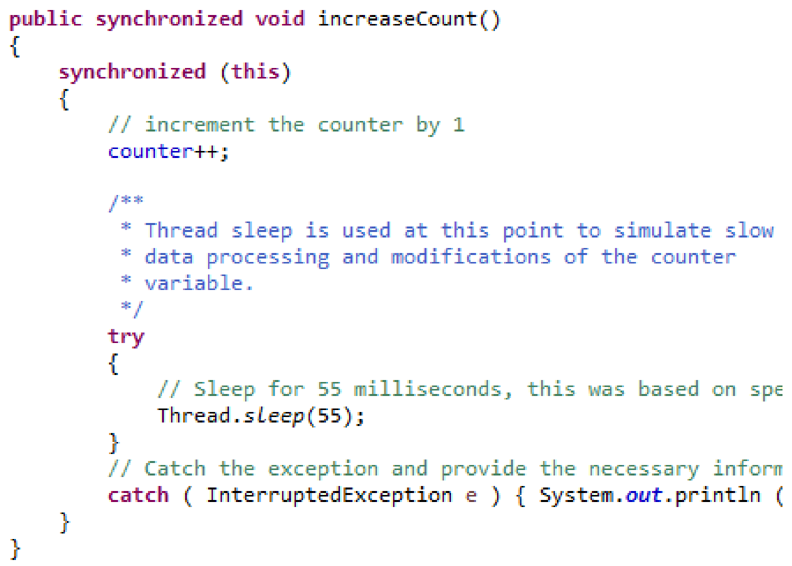


```
<terminated> CountingThreadInitializers (1) [Java Applic
Total Time Program Running: 550 ms.

Counter Value after Execution ----->  24
```

Figure 2 Shows the output from the run of the code without the use of method-level synchronization. The result will always be different, sometimes it may be 30.

Figure 3 on the right shows the increaseCount() function from the Counter class **with the use of method-level synchronization**,



```
public synchronized void increaseCount()
{
    synchronized (this)
    {
        // increment the counter by 1
        counter++;

        /**
         * Thread sleep is used at this point to simulate slow
         * data processing and modifications of the counter
         * variable.
         */
        try
        {
            // Sleep for 55 milliseconds, this was based on spe
            Thread.sleep(55);
        }
        // Catch the exception and provide the necessary inform
        catch ( InterruptedException e ) { System.out.println (
    }
}
```

Figure 3 Shows code snippet from of the increaseCount() function from the Counter class with the use of method-level synchronization.

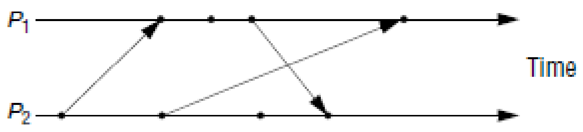
2. A file server uses caching and achieves a hit rate of 80%. File operations in the server cost 5 ms of CPU time when the server finds the request blocked in the cache, and take an additional 15 ms of I/O time otherwise. Explaining any assumptions you make, estimate the server's throughput capacity (average requests/sec) if it is:
   I. Single-threaded
   II. Two-threaded, running on a single processor;
   III. Two-threaded, running on a two-processor computer. [8]

   Answer:

3. A clock is reading 10:27:54:0 (hr:min:sec) when it is discovered to be 4 seconds fast. Explain why it is undesirable to set it back to the right time at that point and show (numerically) how it should be adjusted so as to be correct after 8 seconds has elapsed. [5]

   Answer:

4.



The figure above shows events occurring for each of two processes, p1 and p2. Arrows between processes denote message transmission. Draw and label the lattice of consistent states (p1 state, p2 state), beginning with the initial state (0,0). [15]

Answer:

5. In a certain system, each process typically uses a critical section many times before another process requires it. Explain why Ricart and Agrawala's multicast-based mutual exclusion algorithm is inefficient for this case, and describes how to improve its performance, Does your adaptation satisfy liveness condition ME2? [7]