CSCI 53700 – Fall 2016

Assignment Number 1

Due Date: October 5, 2016

This assignment is based on the principles of clock consistency and associated drifts in a distributed system. You have to create a simulation, running on a single machine, of a simple distributed system involving four process objects (PO). Each PO will contain a *logical clock*, a concept first proposed by Lamport.[1] The concept of the logical clocks and associated synchronization, based on the following rules, will attempt to resolve the clock consistency in this system:

1. Each event (internal, send, or receive) in the system is associated with a time-stamp, based on logical clocks.

2. Each PO, P_i, will have an associated logical clock, LC_i. This logical clock is implemented as a simple counter that is incremented whenever an event takes place within that PO. Since a logical clock has a monotonically increasing value, it assigns a unique number to every event. The time stamp of an event is the value of the logical clock for that event. Hence, if an event A occurs before an event B in P_i, then $LC_i(A) < LC_i(B)$.

3. A P_i will randomly decide if it wants to carry out an internal event or send a message to some other P_j. If it decides to send a message to a randomly selected P_j then it will attach the value of its logical clock to that message. A P_i may also decide to receive a message from another P_j if that P_j has sent a message to the P_i earlier.

4. Whenever a P_j receives a message from another P_i, it will advance its logical clock if the time stamp associated with the incoming message is greater than or equal to the current value of its logical clock, i.e., if P_j, receives a message (event B) from P_i with a time stamp t and $LC_j(B) \leq t$ then P_j should advance its logical clock such that $LC_j(B)$ is equal to $t + 1$.

5. Any P_i, in the system, may exhibit Byzantine or arbitrary failures when it receives a message – e.g., it may choose not to advance its logical clock at all.

Your task is to:

1. Propose the interaction and failure models for this system. Discuss the pros and cons of your design.

2. Simulate this entire environment in C++ or Java using threads. Allow the simulation to reach a *steady-state*, i.e., run the program for a large number of iterations. During each iteration and at the end of the simulation compare the values of the logical clocks of POs – this comparison should indicate the clock drifts and their synchronizations. Repeat the simulation with different probabilities and access their effects on the clock drifts.

3. Create a brief report that indicates the aforementioned models and the analyses of the results of your simulation.

---

[1] http://research.microsoft.com/en-us/um/people/lamport/pubs/time-clocks.pdf

Please employ good software engineering principles in your design and implementation. The program must run on any machine from the following list:

```
in-csci-rrpc01.cs.iupui.edu 10.234.136.55
in-csci-rrpc02.cs.iupui.edu 10.234.136.56
in-csci-rrpc03.cs.iupui.edu 10.234.136.57
in-csci-rrpc04.cs.iupui.edu 10.234.136.58
in-csci-rrpc05.cs.iupui.edu 10.234.136.59
in-csci-rrpc06.cs.iupui.edu 10.234.136.60
```

Provide adequate documentation of your programs. Create a *makefile* for your program. Submit all the source files (including the readme, input/output and make files) by using *submitd* command on Pegasus/Tesla. Also, hand-in a hard-copy of the report at the beginning of the class on the due date.