EE 382N: Distributed Systems
Homework 1

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)
TA: Wei-Lun Hung (email: wlhung@utexas.edu)

Deadline: Sep. 17^{th}, 2015

This homework contains a theory part (Q1-Q4) and a programming part (Q5). The theory part should be written or typed on a paper and submitted at the beginning of the class. The source code must be uploaded through Canvas before the end of the due date (i.e., 11:59pm in Sep. 17^{th}). The assignment should be done in teams of two. You should use the templates downloaded from the course github (https://github.com/kinmener/UT-Garg-Distributed.git). You should not change the file names and function signatures. In addition, you should not use package for encapsulation. Please zip and name the source code as [EID1_EID2].zip.

1. **(10 pts)**
   (a) Prove or disprove that every symmetric and transitive relation is reflexive.
   (b) Prove or disprove that every irreflexive and transitive relation is asymmetric.

2. **(10 pts)**
   Prove the following for vector clocks: $s \rightarrow t$ iff

$$(s.v[s.p] \leq t.v[s.p]) \wedge (s.v[t.p] < t.v[t.p])$$

3. **(10 pts)** Some applications require two types of accesses to the critical section – *read* access and *write* access. For these applications, it is reasonable for multiple read accesses to happen concurrently. However, a *write* access cannot happen concurrently with either a *read* access or a write access. Modify Lamport's mutex algorithm for such applications.

4. **(10 pts)**
   (a) Extend Lamport's mutex algorithm to solve k-mutual exclusion problem which allows at most k processes to be in the critical section concurrently.
   (b) Extend Ricart and Agrawala's mutex algorithm to solve the k-mutual exclusion problem.

5. **(60 pts)** The goal of this assignment is to learn client server programming with TCP and UDP sockets. You are required to implement a server and a client for a ticket reservation system for a movie. The system should function with both TCP as well as UDP connections. Assume that the movie theater has c total seats. There is a single server, but multiple clients may access the server concurrently. Assume that a person can reserve only one seat at any given time. Every client accepts only the following commands from stand input:

(a) `reserve <name> T|U` – inputs the name of a person and reserves a seat against this name. The client sends this command to the server using the appropriate protocol indicated by U or T (U = UDP, T = TCP). If the theater does not have enough seats(completely booked), no seat is assigned and the command responds with message: 'Sold out - No seat available'. If a reservation has already been made under that name, then the command responds with message: 'Seat already booked against the name provided'. Otherwise, a seat is reserved against the name provided and the client is relayed a message: 'Seat assigned to you is `<seat-number>`'.

(b) `bookSeat <name> <seatNum> T|U` – behaves similar to reserve command, but imposes an additional constraint that a seat is reserved if and only if there is no existing reservation against name and the seat having the number `<seatNum>` is available. If there is no existing reservation but `<seatNum>` is not available, the response is: '`<seatNum>` is not available'.

(c) `search <name> T|U` – returns the seat number reserved for name. If no reservation is found for name the system responds with a message: 'No reservation found for `<name>`'.

(d) `delete <name> T|U` – frees up the seat allocated to that person. The command returns the seat number that was released. If no existing reservation was found, responds with: 'No reservation found for `<name>`'.