EE 382N: Distributed Systems
Homework 1

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)
TA: Changyong Hu (email: colinhu9@utexas.edu)

Deadline: Sep. 13^{th}, 2019

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. 13^{th}). The assignment should be done in teams of two. You should use the templates downloaded from the course github site. 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)** 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.
3. **(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.
4. **(70 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 an online store system. The system should function with both TCP as well as UDP connections. The server has an input file which represents all products of the store. There is a single server, but multiple clients may access the server concurrently. The server must be multithreaded. Assume that a person can purchase only one type of product at any given time. Every client accepts only the following commands from standard input:
   (a) `setmode T|U` – sets the protocol for communication with the server. The protocol is specified by the letter U ot T where U stands for UDP and T stands for TCP. The default mode of communication is TCP.

(b) `purchase <user-name> <product-name> <quantity>` – inputs the name of a customer, the name of the product, and the number of quantity that the user wants to purchase. The client sends this command to the server using the current mode of the appropriate protocol. If the store does not have enough items, the server responds with message: 'Not Available - Not enough items'. If the store does not have the product, the server responds with message: 'Not Available - We do not sell this product'. Otherwise, an order is placed and the server replies a message: 'You order has been placed, `<order-id> <user-name> <product-name> <quantity>`'. Note that, the order-id is unique and automatically generated by the server. You can assume that the order-id starts with 1. The server should also update the inventory.

(c) `cancel <order-id>` – cancels the order with the `<order-id>`. If there is no existing order with the id, the response is: '`<order-id>` not found, no such order'. Otherwise, the server replies: 'Order `<order-id>` is canceled' and updates the inventory.

(d) `search <user-name>` – returns all orders for the user. If no order is found for the user, the system responds with a message: 'No order found for `<user-name>`'. Otherwise, list all orders of the users as `<order-id>, <product-name>, <quantity>`. Note that, you should print one line per order.

(e) `list` – lists all available products with quantities of the store. For each product, you should show '`<product-name> <quantity>`'. Note that, even if the product is sold out, you should also print the product with quantity `0`. In addition, you should print one line per product. The products should be listed in the sorted order.

You can assume that the servers and clients always receive consistent and valid commands from users. You can also assume that the server is running before the client is run. The server reads the initial inventory from the input file. The following shows an example of the input file. Each line shows the name of the product and its quantity. You can assume that the product name does not contains any space.

---

input file:
phone 10
laptop 20
camera 15
ps4 14
xbox 7