EE 360P: Concurrent and Distributed Systems
Assignment 3

Instructor: Professor Vijay Garg (email: garg@ece.utexas.edu)

Deadline: February 28, 2017

This homework contains a theory part (Q1-Q2) and a programming part (Q3). The theory part should be written or typed on a paper and submitted at the beginning of the class. The source code (Java files) of the programming part must be uploaded through the canvas before the end of the due date (i.e., 11:59pm on February 28). The assignment should be done in teams of two. You should use the templates downloaded from the course github (https://github.com/vijaygarg1/EE-360P.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)** 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.

2. **(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.

3. **(80 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 book system of our library. The system should function with both TCP as well as UDP connections. The server has an input file which represents all books this library has. There is a single server, but multiple clients may access the server concurrently. The server must be multithreaded. Assume that a person can borrow only one book at any given time. Every client accepts only the following commands:

   (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 UDP.

   (b) `borrow <student-name> <book-name>` – inputs the name of a student, the name of the book. The client sends this command to the server using the current mode of the appropriate protocol. If the library has all this book lent out, the server responds with message: 'Request Failed - Book not available'. If the library does not have the book, the server responds with message: 'Request Failed - We do not have this book'. Otherwise, the borrow request succeeds and the server replies a message: 'You request has been approved, `<record-id> <student-name> <book-name>`'. Note that, the record-id is unique and automatically generated by the server.

You can assume that the request-id starts with 1. The server should also update the library inventory.

(c) `return <record-id>` – return the book associated with the `<record-id>`. If there is no existing borrow record with the id, the response is: '`<record-id>` not found, no such borrow record'. Otherwise, the server replies: '`<record-id>` is returned' and updates the inventory.

(d) `list <student-name>` – list all books borrowed by the student. If no borrow record is found for the student, the system responds with a message: 'No record found for `<student-name>`'. Otherwise, list all records of the student as `<record-id> <book-name>`. Note that, you should print one line per borrow record.

(e) `inventory` – lists all available books with quantities of the library. For each book, you should show '`<book-name> <quantity>`'. Note that, even if there is no such book left, you should also print the book with quantity `0`. In addition, you should print one line per book.

(f) `exit` – inform server to stop processing commands from this client and print inventory to inventory file named as inventory.txt.

You can assume that the servers and clients always receive consistent and valid commands from users. The server reads the initial book inventory from the input file. Also, the book name has the is sufficient to uniquely represent a book. Each line shows the name of the book and its quantity. You are provided with a command file in which each line contains a command needs to be executed. All the return messages after each command should be written into the file named **out_clientID.txt** in the current directory, e.g, client 1 should write to out_1.txt. If there is no return message from server, don't output anything. The following shows an example of the input file, command file and output file

---

input file:
"The Letter" 1
"The Great Gatsby" 15
"Divergent" 10
"The Count of Monte Cristo" 17

command file:
setmode T
borrow Mike "The Letter"
borrow Jack "The Letter"
borrow Lucy "Divergent"
list Mike
return 1
inventory
exit

output file:
You request has been approved, 1 Mike "The Letter"
Request Failed - Book not available
You request has been approved, 2 Lucy "Divergent"
1 "The Letter"
1 is returned

"The Letter" 0
"The Great Gatsby" 15
"Divergent" 9
"The Count of Monte Cristo" 17

inventory file:
"The Letter" 0
"The Great Gatsby" 15
"Divergent" 9
"The Count of Monte Cristo" 17

Note: the book name includes quotation marks. Your program should run in this way: in one terminal, run **java server-program input-file inventory.txt**, in other terminals, run client side program, e.g, for client 1, run **java client-program command-file1 out_1.txt**, for client 2, run **java client-program command-file2 out_2.txt**. After executing your server and clients program, in the current directory the following files should be generated: **one inventory.txt file and out_clientId.txt** file for each client. There will be two types of test cases: sequential and concurrent. For sequential test cases, only one client(client 1) tries to executes commands, we will compare your out_clientId.txt file(out_1.txt for client 1) with our expected one. For concurrent test cases, two or more clients will try to execute commands simultaneously, we will only compare your final inventory.txt with our expected one. We will just use diff command to compare your output file and the expected one, so careful with your output file format. Please zip and name the source file as **EID1_EID2.zip**. No package name in your source file. Make sure you test your program on a Linux machine.