Assignment 7

TCP Client-server II

Due: November 10, 2006

1. In this assignment, you will write a simplified online railway reservation system that will require you to use sockets, shared memory, and semaphores together. In the reservation system, each train is identified by a train no. (an integer), name (string, one word only), an originating station (string, one word only), a terminating station (string, one word only), a departure time (string in format hh:mm where hh is 2 digit hour, mm is 2 digit minutes), and an arrival time (string in time format as before). For each train, a certain maximum number of available seats are defined (assume all seats are of one class only), and users can buy these seats online. There are two types of users. Normal users can see the details of a train (search by train number or search by originating and terminating station pair), see the number of seats currently available on a train, and book a ticket. Administrative users, ***in addition to being able to do everything that a normal user can do***, can also update the maximum number of seats on an existing train. Normal users do not require to login into the system, but administrative users need to login first. For simplicity, assume that you are dealing with reservation for a single date only, so you need not bother about keeping reservation information for different dates.

   A normal user of the system that you design should be able to execute the following commands at the server.

   - **find <train_no>** : see the details of the train identified by **<train_no>**. If there is no such train, server should send an appropriate message.
   - **find_between <originating station> <terminating station> :** Show details of all trains between the two stations.
   - **available <train_no>**: shows the number of seats currently available on the train identified by **<train_no>**
   - **reserve <train_no> <name>**: If seats are available in the train identified by **<train_no>**,, a reservation is made in the name of the person with name **<name>**. Assume that names are one word strings. If the reservation is successful, a PNR no. (an integer) is given to the client. If unsuccessful, an appropriate message is returned.

   An administrative user should **also** be able to do the following:

   - **login <username> <passwd>** : logs the user in. **<username>** and **<passwd>** are strings. The server should check the username-password in a file and returns the string “ok” to the client on a match; if no match is found, the server returns the string “login failed”. Assume that the server has a file with username–passwords of the valid users of the system (prepared manually). If login fails, no further commands are accepted and the socket is closed by the server.
   - **changemax <train_no> <newmax>:** Change the maximum number of seats on the train identified by **<train_no>** to the number identified by **<newmax>**. The operation should fail if **<newmax>** is less than the currently defined max. seats for that train.

Design a client that gives a console based user interface (i.e., no graphical user interface is needed) that will first ask the user whether he/she is a normal user or administrator. If a normal user is chosen, the user can choose or type in any of the above commands (you can print nice names for the commands at the client end if you want.). If an administrative user is chosen, a user-password is asked for and the **login** command is sent to the server. Once a user types in a command and the parameters, the client opens a TCP socket connection to the server and sends the command in the above format to the server. The result, if any, is displayed in a nice manner.

The server is a concurrent server using TCP. The information is stored in several files. The information about the details (name, number, originating and terminating stations, departure and arrival time, max. number of seats) of all the trains is stored in one single file. The current reservation information of a particular train is stored in a file with the train no. as its name. So if you have x trains, you will have a total of (x +1) files. You can design your own format for storing the information in the files.

However, for efficiency reasons, the server, when it starts, loads the details of the trains, and the number of seats currently available for each train in shared memory. Design the data structures needed for maintaining all the information in the shared memory. The current reservation information (who has reserved) of the individual trains is **not** stored in the shared memory. On a client request, a separate process is created to handle the client's request. All searches (**find**, **find_between**, and **available** commands) are answered from the data in the shared memory, the files are not accessed at all for this. A reservation/cancellation or a change in the maximum seats in a train (**changemax** and **reserve** commands) is done directly in the respective file itself; only change in the shared memory in this case is to update the number of available seats in the shared memory. You must design appropriate synchronization mechanisms using semaphore to make sure reads and writes to any shared memory do not happen simultaneously.

For simplicity, you can assume that the maximum number of trains in your system is 25, and create the shared memory accordingly (and hence, you need not worry about dynamic allocation of shared memory for this assignment). The train information to test your program with will be put up on the course website shortly.