**Project 3:** Token ring bulletin board

*Protocol document*

Student name: Probal Chandra Dhar

***Server***: when you start my server at first you’ve to pass the number of peer you’re expecting to contact the server as par project description. Also, the port number to bind the socket so that every peer can contact the server. After receiving a message from a peer it’s sends back a message to the peer and that contains “connected” so that peer knows I’m now connected to server. After getting all the response from the number of peers the server sends IP address and port (struct sockaddr\_in) to the peers. A peer will receive its next peer’s sockaddr\_in. So if there’s three peer, peer 1 will get peer 2’s sockaddr\_in, peer 2 will get peer 3’s sockaddr\_in and peer 3 will get peer 1’s sockaddr\_in. Now the ring is formed. Server will send the first peer a message “start”, that peer will start the game. Server will exit here.

***Peer:*** At first, peer will contact to the server so that a ring can form with the message “new”, in response it will get response message “connected” from the server. After that it will wait for the destination IP & port from the server. After it gets the struct sockaddr\_in from the server, all the peers will play a game. Every peer will select a number for the game. I’m choosing the number by port number % 60000. So, after selecting the number every peer will 0 to its next peer. After it’ll wait for a number from previous peer. Then after receiving the number it’ll compare the number with the number for its own(port number % 60000), if the received number is not the number the peer selected then it’ll increase the received the number by 1 and then again send the number to the next peer. Again, after receiving it’ll compare the number with the received number. If the received number is the same as the calculated number then it’ll sent -1 to the next peer so that all the peers can stop the while loop. Which peer wins first it’ll start passing the token to the next peer. For my project the TOKEN is “T”.

After sending the token from the first peer I’m creating the thread to send and receive the token for each peer.

In the thread when a message received then I’m checking if the message is “T” or not. If it’s the token, then I’m checking if user needs to hold the token or not by global variable doWeNeedToken. If the value of it 0 then we need token so that I’m passing the token to the next peer immediately. If the value of it 1 then we need to write the file. I’m doing the file write by doFileWrite() which uses the writeFile() to complete writing a file. If the value of doWeNeedToken is 2 then we need to read the file. If the value is 3 then we need to show the list to screen. I’m using semaphore when I’m doing these operations so that no other thread can do the same operation at a time.

***Join procedure:*** When a new peer is joining the ring then I’m seeing if the received message is “new”. If it’s new, then a new peer is trying to join. So, I’m giving response “connected” to the newly joined. Then I’m sending the dest of the current peer to the newly joined peer. Then I’m sending another message “join” to the newly joined peer so that it can set the prev address for its own. And then replace the value of the dest by prev. For me dest and prev are global struct sockaddr\_in, they store the destination and previous peer’s information.

***Exit procedure:*** When a peer tries to exit then I’m sending the previous peer a message which contains “IPchange” and then the destination peer’s information (struct sockaddr\_in). When a peer gets, a message containing “IPchange” (in the thread) then it’ll again wait for the destination information from that peer. In this way, the previous peer can set up its new next peer by that information. After that I’m sending the token for the next peer.

***Format of messages:*** For my project format of message is fixed. It can only contain 31 characters. This can be extended. This is the format of a message. You can try with it. There’s a constant defined in the file peer.h “MESSAGE”, right now it’s 54 (including header, footer and body of a message).

This is an interesting project.

This is an interesting project.

This is the body of a message. Header and footer will be added when you write the message in a file.

For my project a file must be created before you can do some operation on the file. I attached a file named “test.bin”. Or just create a file and test with it.

***P.S:*** in the SSH server sometimes I face that join or exit fails to do operation. But after I do second run then it works fine. I don’t know what’s the problem. I ran the program in EC2 server with the same environment (centos 7, gcc) but I didn’t face any such issues in there.