CS-349: Networks Lab
Assignment 3: Socket Programming

Submission deadline: 11:55 PM on Tuesday 20 March, 2018 (Hard Deadline)

In this assignment you need to implement an application using socket programming in C. Check the application assigned to each group in the table given below. The group member information can be found in the "Group Information-Spring 2018" file, attached with the mail.

Application No.	Group Numbers
1	1, 10, 19, 28, 37
2	2, 11, 20, 29, 38
3	3, 12, 21, 30, 39
4	4, 13, 22, 31, 40
5	5, 14, 23, 32, 41
6	6, 15, 24, 33, 42
7	7, 16, 25, 34, 43, 46
8	8, 17, 26, 35, 44
9	9, 18, 27, 36, 45

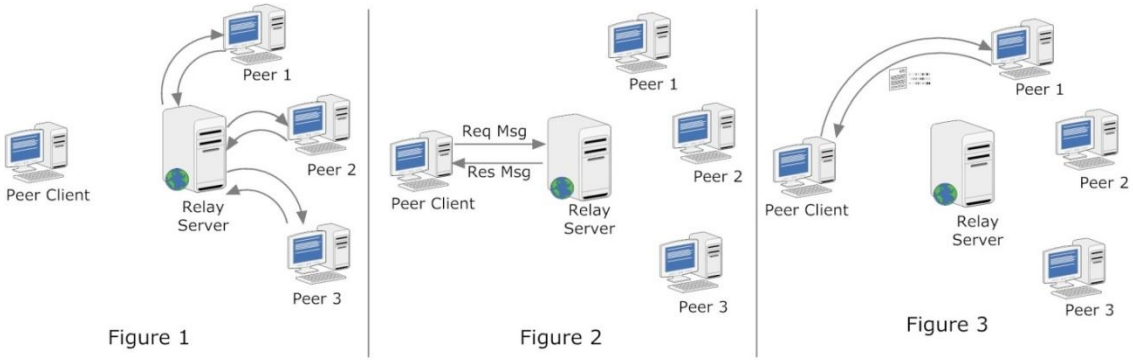
NOTE: Submit your code as a zipped file (max size 1MB) on Moodle by 11:55 PM on Tuesday 20 March, 2018 (hard deadline). The file name should be same as your Group Number, Example, "Group_3.zip", or "Group_3.tar.gz" or "Group_3.rar". Assignment will be evaluated through Viva-voce on Wednesday, 21 March, 2018; during your lab session (evaluation schedule and TA assignment will be notified later).

***Copy cases will be penalized by awarding ZERO marks.**

Useful Resources:
Refer first few chapters of "*Unix. Network Programming*" Volume I, by Richard Stevens

Application #9: *Relay based Peer-to-Peer System using Client-Server socket programming*

In this application, you require implementing three C programs, namely Peer_Client, and Relay_Server and Peer_Nodes, and they communicate with each other based on TCP sockets. The aim is to implement a simple Relay based Peer-to-Peer System.



Initially, the Peer_Nodes (peer 1/2/3 as shown in fig. 1) will connect to the Relay_Server using the TCP port already known to them. After successful connection, all the Peer_Nodes provide their information (IP address and PORT) to the Relay_Server and close the connections (as shown in Figure 1). The Relay_Server actively maintains all the received information with it. Now the Peer_Nodes will act as servers and wait to accept connection from Peer_Clients (refer phase three).

In second phase, the Peer_Client will connect to the Relay_Server using the server's TCP port already known to it. After successful connection; it will request the Relay_Server for active Peer_Nodes information (as shown in Figure 2). The Relay_Server will response to the Peer_Client with the active Peer_Nodes information currently having with it. On receiving the response message from the Relay_Server, the Peer_Client closes the connection gracefully.

In third phase, a set of files (say .txt) are distributed evenly among the three Peer_Nodes. The Peer_Client will take "file_Name" as an input from the user. Then it connects to the Peer_Nodes one at a time using the response information. After successful connection, the Peer_Client tries to fetches the file from the Peer_Node. If the file is present with the Peer_Node, it will provide the file content to the Peer_Client and the Peer_Client will print the file content in its terminal. If not, Peer_Client will connect the next Peer_Node and performs the above action. This will continue till the Peer_Client gets the file content or all the entries in the Relay_Server Response are exhausted (Assume only three/four Peer_Nodes in the system).

Implement the functionalities using appropriate REQUEST and RESPONSE Message formats. After each negotiation phase, the TCP connection on both sides should be closed gracefully releasing the socket resource. You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). Prototype for command line is as follows:

Prototypes for Client and Server
Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

*Please make necessary and valid assumptions whenever required.