CS 558: Computer Systems Lab
(January-May 2024)

Assignment – 1: Network Diagnostic Commands & Socket Programming

Submission deadline: 11:55 PM, Wednesday, 24th January,2024

Instructions:

- Questions (Q1-Q7) are mandatory for all the groups. Submit a soft copy of the report, preferably PDF, on all these experiments. From the rest of the questions, each group needs to implement one application assigned to you (see the Task Allocation Table at the end of the document). Each group then needs to make one single submission containing its solution to Q1-Q7 and the other assigned problem.
- The application should be implemented with socket programming in C/C++ programming language only. No other programming language other than C/C++ will be accepted.
- Submit the set of source code files of the application as a zipped file (maximum file size is 1 MB) by the deadline. The ZIP file's name should be the same as your group number - for example, "Group_4.zip", "Group_4.rar", or "Group_4.tar.gz".
- Only one member from a group needs to submit this form by uploading the file: **https://forms.gle/9KiDTEq5bsHm4zeEA**.
- The assignment will be evaluated offline/through viva voce during your lab session. where you will need to explain your source codes and execute them before the evaluator.
- Write your own source codes and do not copy from any source. A plagiarism detection tool will be used and any detection of unfair means will be penalized by awarding NEGATIVE marks (equal to the maximum marks for the assignment).
- Marking will be done for the group as a whole.

References: https://www.geeksforgeeks.org/socket-programming-cc/

Questions:

Q1. The Internet Ping command bounces a small packet(s) to test network communications, and then shows how long this packet(s) took to make the round trip. The Internet Ping program works much like a sonar echo-location, sending a small packet of information containing an ICMP ECHO_REQUEST to a specified computer, which then sends an ECHO_REPLY packet in return. Explore more about the ping command and answer the following questions (Unix or GNU/Linux version only):

a) What is the option required to specify the number of echo requests to send with ping command?
b) What is the option required to set time interval (in seconds), rather than the default one second interval, between two successive ping ECHO_REQUESTs?
c) What is the command to send ECHO_REQUEST packets to the destination one after another without waiting for a reply? What is the limit for sending such ECHO_REQUEST packets by normal users (not super user)?
d) What is the command to set the ECHO_REQUEST packet size (in bytes)? If the PacketSize is set to 32 bytes, what will be the total packet size?

Q2. Select six hosts of your choice in the Internet (mention the list in your report) and experiment with pinging each host 25 times at three different hours of the day. Check if there exist cases, which show packet loss greater than 0% and provide reasoning. Find out average RTT for each host and explain whether measured RTTs are strongly or weakly correlated with the geographical distance of the hosts. Pick one of the above used hosts and repeat the experiment with different packet sizes ranging from 64 bytes to 2048 bytes. Plot the average RTT, and explain how change in packet size and time of the day impact RTT. You can use the following online tools for this experiment:

i) http://www.spfld.com/ping.html
ii) https://www.subnetonline.com/pages/network-tools/online-ping-ipv4.php

Q3. With regard to ifconfig and route commands, answer the following questions:

a) Run ifconfig command and describe its output (identify and explain as much of what is printed on the screen as you can).
b) What options can be provided with the ifconfig command? Mention and explain at least four options.
c) Explain the output of route command.
d) Mention and explain at least four options of the route command. Execute the route command with these four options and show the output.

Q4. Answer the following questions related to netstat command.

a) What is the command netstat used for?
b) What parameters for netstat should you use to show all the established TCP connections? Include a screenshot of this list for your computer and explain all the fields of the table in the output.

c) What does "netstat –r" show? Explain all the fields of the output.
d) What option of netstat can be used to display the status of all network interfaces? By using netstat, figure out the number of interfaces on your computer.
e) What option of netstat can be used to show the statistics of all UDP connections? Run the command for this purpose on your computer and show the output.
f) Show and explain the function of loopback interface.

Q5. What is a traceroute tool used for? Perform a traceroute experiment (with same hosts used in Q2) at three different hours of the day, and then answer the questions below. Use any one of the following online tools for this experiment:

- http://ping.eu;
- http://www.cogentco.com/en/network/looking-glass;
- http://network-tools.com;

a) List out the hop counts for each host in each time slot. Determine the common hops between two routes if they exist.
b) Check and explain the reason if route to same host changes at different times of the day.
c) Inspect the cases when traceroute does not find complete paths to some hosts and provide reasoning.
d) Is it possible to find the route to certain hosts which fail to respond with ping experiment? Give reasoning.

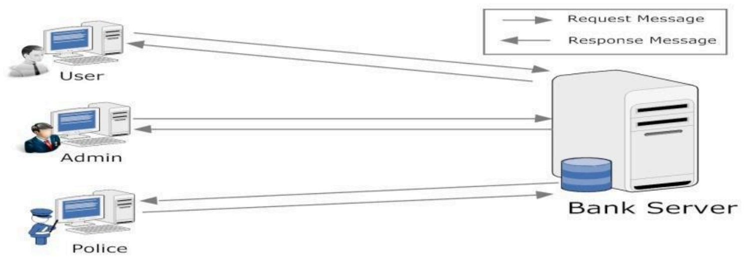
Q6. Answer the following questions with regard to network addresses.

a) How do you show the full ARP table for your machine? Explain each column of the ARP table.
b) Check and explain what happens if you try and use the arp command to add or delete an entry to the ARP table. Find out how to add, delete or change entries in the ARP table. Use this mechanism to add at least four new hosts to the ARP table and include a printout.
c) What are the parameters that determine how long the entries in the cache of the ARP module of the kernel remain valid and when they get deleted from the cache? Describe a trial-and-error method to discover the timeout value for the ARP cache entries.
d) What will happen if two IP addresses map to the same Ethernet address? Be specific on how all hosts on the subnet operate.

Q7. Local network analysis: Query your LAN using the nmap command to discover which hosts are online. Use a command such as: *nmap –n –sP <Subnet Range> (e.g., 172.16.112.0/26)*
You can choose a different LAN subnet address as well (make sure you report the same in your report explicitly).
Now run the command repeatedly at different times of the day, and find the number of hosts online. Do it for at least 6 times with sufficient time gap. Plot a graph against time to see if there are any hourly trends for when computers are switched ON or OFF in your LAN.

Application #1: *Banking System using Client-Server socket programming*

In this application, you require implementing two C programs, namely Client and Bank Server, and they communicate with each other based on TCP sockets. The goal is to implement a simple Banking System.



Initially, the client will connect to the bank server using the server's TCP port already known to the client. After successful connection, the client sends a Login Message (containing the Username and password) to the bank server. The client side, we can have three different types of user modes namely, Bank_Customer, Bank_Admin and Police. The bank server has the following files with him: Login_file (contains the login entries, assume limited number of static entries only), Customer_Account_files (Assume the bank has 10 Bank_Customers only and one file for each customer, which maintains the transaction history. Refer to Login_file and Customer_Account_files formats for more details). Once the Bank server receives the login request, it validates the information and performs the functionalities according to the user mode type. The system must provide the following functionalities to the following users:

- **Bank_Customer:** The customer should able to see AVAILABLE BALANCE in his/her account and MINI STATEMENT of his/her account.
- **Bank_Admin:** The admin should be able to CREDIT/DEBIT the certain amount of money from any Bank_Customer ACCOUNT (as we do it in a SBI single window counter.☺). The admin must update the respective "Customer_Account_file" by appending the new information. Handel the Customer account balance underflow cases carefully.
- **Police:** The police should only be able to see the available balance of all customers. He is allowed to view any Customers MINI STATEMENT by quoting the Customer _ID (i.e. User_ID with user_type as 'C').

Login_file entry format:

User_ID	Password	User_Type (C/A/P)

Customer _Account_files entry format:

Transaction_Date	Transaction Type (Credit/Debit)	Available Account_Balance

Implement the functionalities using proper 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>

NB: Please make necessary and valid assumptions whenever required.

Application #2: *File Transfer Protocol (FTP) using Client-Server socket programming*

In this assignment, you require to implement two C programs, namely server and client to communicate with each other based on TCP sockets. The goal is to implement a simple File Transfer Protocol (FTP).

Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client should be able to perform the following functionalities:

> **PUT:** Client should transfer the file specified by the user to the server. On receiving the file, server stores the file in its disk. If the file is already exists in the server disk, it communicates with the client to inform it. The client should ask the user whether to overwrite the file or not and based on the user choice the server should perform the needful action.
>
> **GET:** Client should fetch the file specified by the user from the server. On receiving the file, client stores the file in its disk. If the file is already exists in the client disk, it should ask the user whether to overwrite the file or not and based on the user choice require to perform the needful action.
>
> **MPUT and MGET:** MPUT and MGET are quite similar to PUT and GET respectively except they are used to fetch all the files with a particular extension (e.g. .c, .txt etc.). To perform these functions both the client and server require to maintain the list of files they have in their disk. Also implement the file overwriting case for these two commands as well.

Use appropriate message types to implement the aforesaid functionalities. For simplicity assume only .txt and .c file(s) for transfer.

You should accept the IP Address and Port number from the command line (Don't use a hard-coded port number). *Please make necessary and valid assumptions whenever required.

Prototype for command line is as follows:

<u>Prototypes for Client and Server</u>

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

Application #3: *Base64 encoding system using Client-Server socket programming*

In this application, you require to implement two C programs, namely server and client to communicate with each other based on TCP sockets. The aim is to implement simple Base64 encoding communication protocol.

Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client accepts the text input from the user and encodes the input using Base64 encoding system. Once encoded message is computed the client sends the Message (Type 1 message) to the server via TCP port. After receiving the Message, server should print the received and original message by decoding the received message, and sends an ACK (Type 2 message) to the client. The client and server should remain in a loop to communicate any number of messages. Once the client wants to close the communication, it should send a Message (Type 3 Message) to the server and the TCP connection on both the server and client should be closed gracefully by releasing the socket resource.

The messages used to communicate contain the following fields:

Message_ Type	**Message**

1. Message_type: integer
2. Message: Character [MSG_LEN], where MSG_LEN is an integer constant
3. <Message> content of the message in Type 3 message can be anything.

You also require implementing a "***Concurrent Server***", i.e., a server that accepts connections from multiple clients and serves all of them *concurrently*.

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>

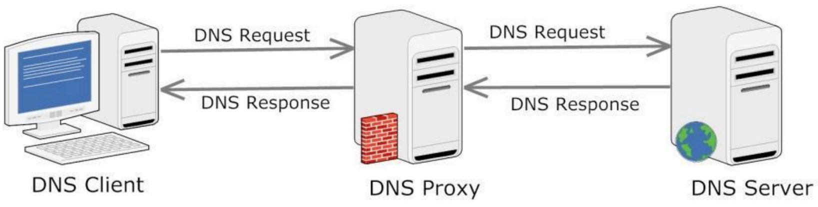
NB: Please make necessary and valid assumptions whenever required.

Base64 Encoding System Description:

Base64 encoding is used for sending a binary message over the net. In this scheme, groups of 24bit are broken into four 6 bit groups and each group is encoded with an ASCII character. For binary values 0 to 25 ASCII character 'A' to 'Z' are used followed by lower case letters and the digits for binary values 26 to 51 & 52 to 61 respectively. Character '+' and '/' are used for binary value 62 & 63 respectively. In case the last group contains only 8 & 16 bits, then "==" & "=" sequence are appended to the end.

Application #4: *Multi-stage DNS Resolving System using Client-Server socket programming*

In this application, you require implementing three C programs, namely Client, Proxy Server (which will act both as client and server) and DNS Server, and they communicate with each other based on TCP sockets. The aim is to implement a simple 2 stage DNS Resolver System.



Initially, the client will connect to the proxy server using the server's TCP port already known to the client. After successful connection, the client sends a Request Message (Type 1/Type 2) to the proxy server. The proxy server has a limited cache (assume a cache with three IP to Domain_Name mapping entries only). After receiving the Request Message, proxy server based on the Request Type (Type 1/Type 2) searches its cache for corresponding match. If match is successful, it will send the response to the client using a Response Message. Otherwise, the proxy server will connect to the DNS Server using a TCP port already known to the Proxy server and send a Request Message (same as the client). The DNS server has a database (say .txt file) with it containing set of Domain_name to IP_Address mappings. Once the DNS Server receives the Request Message from proxy server, it searches in its file for possible match and sends a Response Message (Type 3/Type 4) to the proxy server. On receiving the Response Message from DNS Server, the proxy server forwards the response back to the client. If the Response Message type is 3, then the proxy server must update its cache with the fresh information using FIFO scheme. After each negotiation phase, the TCP connection on both sides should be closed gracefully releasing the socket resource.

Request Message Format:

Request_Type	Message

- Type 1: Message field contains Domain Name and requests for corresponding IP address.
- Type 2: Message field contains IP address and request for the corresponding Domain Name.

Response Message Format:

Response_Type	Message

- Type 3: Message field contains Domain Name/IP address.
- Type 4: Message field contains error message "entry not found in the database".

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:
<u>Prototypes for Client and Server</u>

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>
NB: Please make necessary and valid assumptions whenever required.

Application #5: *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.

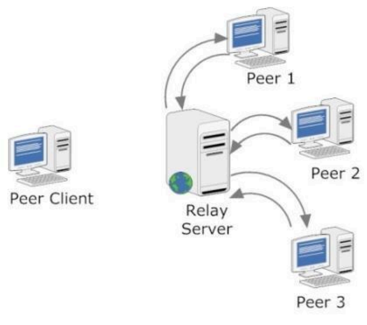


Figure 1

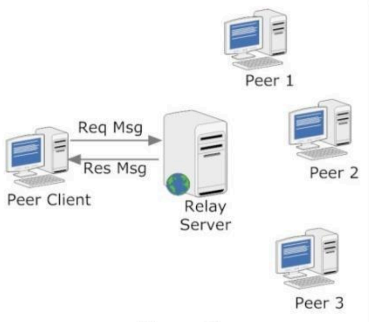


Figure 2

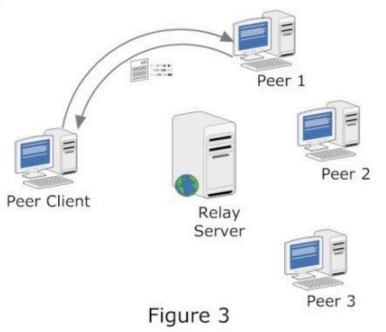


Figure 3

Initially, the Peer_Nodes (peer 1/2/3 as shown in Figure 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>

NB: Please make necessary and valid assumptions whenever required.

Application #6: *Client-Server programming using both TCP and UDP sockets*

In this assignment, you require to implement two C programs, namely server and client to communicate with each other based on both TCP and UDP sockets. The aim is to implement a simple 2 stage communication protocol.

Initially, server will be waiting for a TCP connection from the client. Then, client will connect to the server using server's TCP port already known to the client. After successful connection, the client sends a Request Message (Type 1 message) to the server via TCP port to request a UDP port from server for future communication. After receiving the Request Message, server selects a UDP port number and sends this port number back to the client as a Response Message (Type 2 Message) over the TCP connection. After this negotiation phase, the TCP connection on both the server and client should be closed gracefully releasing the socket resource.

In the second phase, the client transmits a short Data Message (Type 3 message) over the earlier negotiated UDP port. The server will display the received Data Message and sends a Data Response (type 4 message) to indicate the successful reception. After this data transfer phase, both sides close their UDP sockets.

The messages used to communicate contain the following fields:

Message_ Type	**Message_Length**	**Message**

1. Message_type : integer
2. Message_length : integer
3. Message : Character [MSG_LEN], where MSG_LEN is an integer constant

<Data Message> in **Client** will be a **Type 3** message with some content in its message section.

You also require implementing a **"Concurrent Server"**, i.e., a server that accepts connections from multiple clients and serves all of them concurrently.

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:

Prototype for command line is as follows:
<u>Prototypes for Client and Server</u>

Client: <executable code><Server IP Address><Server Port number>
Server: <executable code><Server Port number>

NB: Please make necessary and valid assumptions whenever required.

Task Allocation Table

Questions	Groups
Q1-Q7	All Groups
Application-1	1, 7, 13
Application-2	2, 8, 14
Application-3	3, 9, 15
Application-4	4, 10, 16
Application-5	5, 11, 17
Application-6	6, 12