CS 342: Computer Networks Lab

(July-November 2023)

Assignment – 2

This assignment is a programming assignment where you need to implement an application using socket programming in C/C++/java programming language. This is a Group **assignment; each group needs to create and** submit the assignment. The applications' description is given in this document.

Instructions:

- The application should be implemented in C/C++/java programming language only. No other programming language other than C/C++/java will be accepted.
- Submit the set of source code files of the application and report as a zipped file on MSTeams (maximum file size is 1 MB) by the deadline of **8:55 am on < 11.09.2023 >** (hard deadline). The ZIP file's name should be the **group number**.
- Only one student per group needs to upload the assignment on Teams with the group name.
- The assignment will be evaluated offline/through viva-voce during your lab session on Monday, **<11.09.2023>** where you will need to explain your source codes and execute them before the evaluator. The details of the venue and time will be communicated to you later, on Teams.
- Write your own source code and do not copy from any source. 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).

Question1: Implementing Basic DNS Lookup **(25)**

Write a C/C++/java program tool create a simple DNS lookup tool that converts domain names into their corresponding IP addresses. Students are not allowed to use any external libraries or modules that directly provide DNS functionality. You'll need to use the basic concepts of networking and socket programming.

Your task is to write a program that does the following:

1) Takes a domain name as input from the user.

2) Establishes a UDP socket connection to a DNS server (use Google's public DNS server: 8.8.8.8, port 53).

3) Sends a DNS query packet to the server following the DNS message format.

4) Parses the DNS response packet and extracts the IP address(es) associated with the provided domain name.

5) Prints the IP address(es) to the console.

Note:

The DNS message format consists of various sections, including the Header, Question, Answer, Authority, and Additional sections. You'll need to create and format these sections correctly in your DNS query packet and also implement caching of DNS responses to improve the lookup speed for frequently visited domain names. Test your program with various domain names and verify that it returns the correct IP addresses.

Question 2: Implement a basic HTTP, Web cache that stores recently accessed web pages to improve retrieval speed for subsequent requests. The cache should follow a least-recently-used (LRU) eviction policy. The cache will store web pages retrieved using HTTP GET requests.

(25)

Requirements:

- Implement a cache data structure that can store a fixed number of web pages (e.g., 5 pages) in memory.
- Implement a function to retrieve a web page from the cache. If the requested page is present in the cache, return the cached content. If not, fetch the page using an HTTP GET request, store it in the cache (evicting the least recently used page if necessary), and return the content.
- Use sockets to perform the HTTP GET request and receive the web page content.
- Use a linked list to keep track of the order of page accesses. When a page is accessed, move it to the front of the list to represent it as the most recently used page.
- Implement a function to display the contents of the cache, showing the order of page accesses from most to least recently used.

Hints:

- Use socket programming to establish a connection with the web server, send an HTTP GET request, and receive the web page content.
- Use a linked list to represent the cache, where each node contains the URL and content of a web page.
- Example webpage url that you can provide for testing: http://quietsilverfreshmelody.neverssl.com/online/

Example Output:

```
Cache size: 3
Enter URL (or 'exit' to quit): http://www.example.com/page1
Page content:
... (content of page1) ...

Cache Contents (Most to Least Recently Used):
1. http://www.example.com/page1

Enter URL (or 'exit' to quit): http://www.example.com/page2
Page content:
... (content of page2) ...

Cache Contents (Most to Least Recently Used):
1. http://www.example.com/page2
2. http://www.example.com/page1

Enter URL (or 'exit' to quit): http://www.example.com/page1
Page content:
... (content of page1) ...

Cache Contents (Most to Least Recently Used):
1. http://www.example.com/page1
2. http://www.example.com/page2

Enter URL (or 'exit' to quit): exit
```

Provide a well-commented C/C++/java code file containing the implementation of the HTTP, Web cache as per the given requirements.

Question 3: In the last assignment, you have created a program to implement a chat server that supports client-server communication. **(25)**

In this assignment, create a chat server to support **client-client communication** using TCP protocol. The following are the details of the application (some of the instructions are same in previous assignment):

1) Set up the Server:

Write a program to create a simple chat server that binds to a specific IP address and port. The server should listen for incoming connections from clients. When a client connects, the server should display a message indicating the successful connection.

2) Handle Multiple Clients:

Modify the server program to handle multiple clients simultaneously. Server should support at least 10 clients at any time.

3)Implement Client Communication:

Write the program for the client application. The client should connect to the server and display a message indicating a successful connection.

4) Client-Client Message Exchange (chat room):

Implement a simple chat mechanism between multiple clients. The client should be able to send messages to all other connected clients (**like broadcasting**), and the other clients should display those messages. The client-to-client communication does not need to be sent to the server. There is also no need to send messages from the server to clients.

5) Client-Client Message Exchange (Messaging):

Implement a simple chat mechanism between multiple clients. The client should be able to send messages to any other connected client (**like private chat**). All the clients will be connected in pairs which can only talk to each other and should not be able to see the communication between other clients.

5) Graceful Exit:

Implement a command in the client and server applications to allow users to exit the chat gracefully.

For example,

typing "/exit" in the client should terminate the connection and close the client application.

typing "/exit" in the server should inform all connected clients of closing the server before terminating the connection and closing the client application.

As with any other program, error handling is expected to deal with any unexpected situations gracefully. For instance, handling cases where the server or client disconnects unexpectedly.

Question 4. Wireshark Analysis of HTTP, Web Cache, DNS, and Transport Layer Protocols **(25)**

Use a Wireshark tool to capture and analyze network traffic related to HTTP, web cache, DNS, and transport layer protocols (TCP and UDP). Install Wireshark (download from www.wireshark.org), and learn how to capture packets and filter the required content. Your task is to perform the specified network activities, capture relevant packets using Wireshark, and then write a comprehensive report documenting your observations, findings, and insights. Each group can take any applications like YouTube, Online Gaming, Live sports streaming etc and needs to perform various activities according to functionalities available in the application and collect the traces for the application using Wireshark. You should carry out your experiments across different network conditions including different time(s) of the day and locations (e.g., lab or hostel, etc.)

Instructions:

- Use Wireshark to capture network traffic as you perform the following tasks.
- Organize your captured packets in separate sections for each task in your report.
- Analyze the captured packets, extract relevant information, and answer the below questions
- Compile your findings into a well-structured report. You can use screenshots, diagrams, and tables to support your analysis.

Tasks:

1) List out all the protocols used by the application at different layers (only those which you can figure out from traces). Study and briefly describe their packet formats.

2) Highlight and explain the observed values for various fields of the protocols. Example: Source or destination IP address and port number, Ethernet address, protocol number, etc.

3) Explain the sequence of messages exchanged by the application for using the available functionalities in the application. For example: upload, download, play, pause, etc. Check whether there are any handshaking sequences in the application. Briefly explain the handshaking message sequence, if any.

4) Explain how the particular protocol(s) used by the application is relevant for functioning of the application.

5) Did you observe any caching mechanisms in the captured packets? Explain.

6) Calculate the following statistics from your traces while performing experiments at different time of the day: Throughput, RTT, Packet size, Number of packets lost, Number of UDP & TCP packets, Number of responses received with respect to one request sent. Report the observed values in your answer, preferably using tables.