

Projects Description:

Common Rules:

- ❖ Groups: 3-4 students from the same lab.
- ❖ Discussion will be during the lab time.
- ❖ Delivery and discussion of the project will be held in the week starting from 15th of December.
- ❖ Delivery on the 22nd of December will be considered late. **[-2 marks]**
- ❖ Using arguments and running from cmd is mandatory. **[1 mark]**
- ❖ Must use Threads (connect multiple clients to the same server at same time). **[2 marks]**
- ❖ Handling exceptions. **[1 mark]**
- ❖ For each project there is a protocol (some essential functions) that should be applied in the application. (You will find them below). **[6 marks]**
- ❖ Any extra function can be considered bonus according to the TA's opinion. **[+1 mark]**
- ❖ Any student can be asked about any part in the code.
- ❖ Programming language: Java or C++
 - node.js is not acceptable.
- ❖ Each TA will choose the best project and this project will get bonus grades. **[+2 marks]**

1. SMTP: (TA. Heba)

- **You can choose to implement one of the following:**

1. **CHOICE 1:** Implement multiple SMTP servers that run on multiple devices connected to each other.
 - 1) Multiple clients connected to multiple SMTP servers. **[1 mark]**
 - 2) Each client should login with email and password when connected to its mail server. **[1 mark]**
 - 3) A client can send a new email with content (MAIL FROM, RCPT TO, DATA) to its mail server, and then its mail server forwards the email to the recipients' servers. **[2 marks]**
 - Forward to only one server **[1 mark]**
 - 4) The recipient's server places any email come in the recipient's mailbox (Save in files). **[2 marks]**
 - 5) A client can choose QUIT to close the connection at any time.
2. **CHOICE 2:** Implement one SMTP server but retrieve mailbox using POP3 or IMAP.
 - 1) Multiple clients connected to one SMTP server. **[1 mark]**
 - 2) Each client should login with email and password when connected to its mail server. **[1 mark]**
 - 3) A client can send a new email with content (MAIL FROM, RCPT TO, DATA) to the mail server. **[1 mark]**
 - 4) The server places any email come in the recipient's mailbox (Save in files). **[1 mark]**
 - 5) The server forwards the email content to all the intended clients using POP3 or IMAP. Then client can open any email from its mailbox. **[2 marks]**
 - Forward to only one client **[1 mark]**
 - 6) A client can choose QUIT to close the connection at any time.

- MIME Type is optional with extra grade.
- **Saving mailboxes:** Each mail server has a folder for each client (client's mailbox), and this folder contains many files where each file represents an email sent to this client before.
Another option: each mail server has many files, a file for each client. This file contains all the emails (separated by any delimiter) sent to this client before.
- **Example of sending new email using SMTP:** the client sends a message ("Do you like ketchup? How about pickles?") from mail server crepes.fr to mail server hamburger.edu.
 - Server: [220 hamburger.edu](http://220.hamburger.edu)

- Client: **HELO** crepes.fr
- Server: **250** Hello crepes.fr, pleased to meet you
- Client: **MAIL FROM:** <alice@crepes.fr>
- Server: **250** alice@crepes.fr... Sender ok
- Client: **RCPT TO:** <bob@hamburger.edu>
- Server: **250** bob@hamburger.edu ... Recipient ok
- Client: **DATA**
- Server: **354** Enter mail, end with "." on a line by itself
- Client: Do you like ketchup?
- Client: How about pickles?
- Client: .
- Server: **250** Message accepted for delivery
- Client: **QUIT**
- Server: **221** hamburger.edu closing connection

- **Example for implementing the sending of a new email.**

- Server: **OPTIONS: SEND NEW, SHOW MAILBOX, SHOW MAIL *mail_id*, QUIT**
- Client: **SEND NEW**
- Client: **MAIL FROM:** alice@crepes.fr
- Server: **250** // When the client receives 250, it means server confirms the sender.
- Client: **RCPT TO:** bob@hamburger.edu
- Server: **250** // When the client receives 250, it means server confirms the recipient.
- Client: **DATA**
- Server: **354** // When the client receives 354, it means that server is waiting for the body of the message which will be end by writing '.' on a separated line.
- Client: Body line 1.
- Client: Body line 2.
- Client: Body line 3.
- Client: . // This means the end of the body.
- Server: **250** // When the client receives 250, it means that server accepted the message for delivery to the intended mail server and then to the intended client.
- Client: **SHOW MAILBOX**
- Server: List of all emails of the client.
- Client: **SHOW MAIL 1**
- Server: The content of the first email.
- Client: **QUIT**
- Server: **221** // Means server close the connection with this client.

2. HTTP: (TA Farah)

- Create an HTTP server and several HTTP clients
- Login **[1 mark]**
 - Number of clients must be dynamic
 - Enter their usernames and passwords
 - Must handle in case a wrong username or password is entered
- Website saves multiples files in a folder.
- Client should send the request line in a format similar to the http request in real life. The request should contain the headers: **[2 marks]**
 - GET or POST (post data on server or update file) Request
 - Path of requested file (ex: file.html or img.jpeg)
 - Host
 - HTTP version

```
GET /ethereal-labs/lab2-1.html HTTP/1.1\r\n
Host: gaia.cs.umass.edu\r\n
```

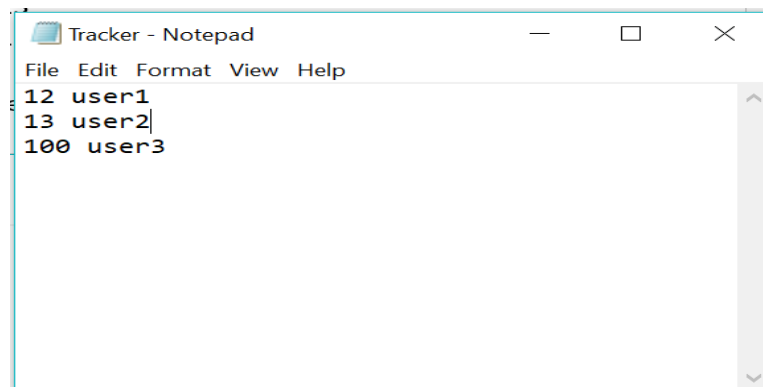
- Request is parsed to the server
- Server will extract the needed information **[1 mark]**
 - GET or POST
 - Path of requested file and this is the root which the server will browse for it and will display its content
- Reply will include **[2 marks]**
 - HTTP version
 - File content
 - Date
 - code
 - 200 OK
 - 400 error (wrong URL)

```
HTTP/1.1 200 OK\r\n
Date: Tue, 23 Sep 2003 05:29:50 GMT\r\n
```

- Note: opening web browser from client is Bonus. **[+1 mark]**
- Apply Caching is Bonus. **[+1 mark]**
- Note: It is forbidden to use WebSockets, you have to write the request and handle the response on your own using Sockets. The project will not be delivered if you use it.

3. Peer-to-Peer File Distribution: (TA Ahmed Alaa)

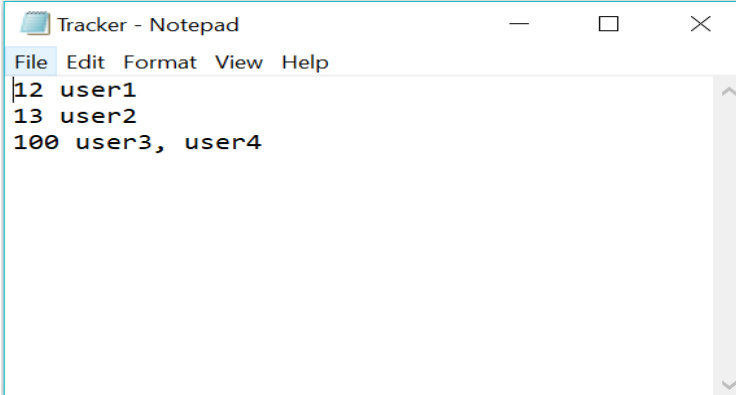
- Using peer classes (i.e. each class is both a server and a client).
- Design a protocol for peer-to-peer file request. **[1 mark]**
 - Design format of request
 - Design format of reply (must include 200 ok – 400 error)
- Number of peers and files must be dynamic (minimum 3 files and 4 peers). **[1 mark]**
- Each peer is directed to a folder and reads files from it.
- Peer has a hash function that is used to convert filename(s) to a hash key. Then the value of hash key and content of file are saved in a table of Key-Value pairs that refers to the files of this peer. **[1 mark] [Note you can use any hash function]**
 - Key is a hashed filename.
 - Value is the file content.
 - Example:
 - User 1 has File1.txt and it is hashed to 12
 - User2 has File2.txt and it is hashed to 13
 - User3 has File3.txt and it is hashed to 100
- Use of Tracker.txt **[1 mark]**
 - It saves the hashed filename and the peers/ users who downloaded it.
 - Available at all peers



- Login **[1 mark]**
 - Each peer enters its username & password and send them to tracker server
 - Tracker server must handle the case if a wrong username or password are entered
 - Once a peer is logged in. Tracker server save its connection.
 - It sends a message to all other peers stating the file(s) that it has so they will update their Tracker.txt.
 - Finally peer will download the recent copy of Tracker.txt

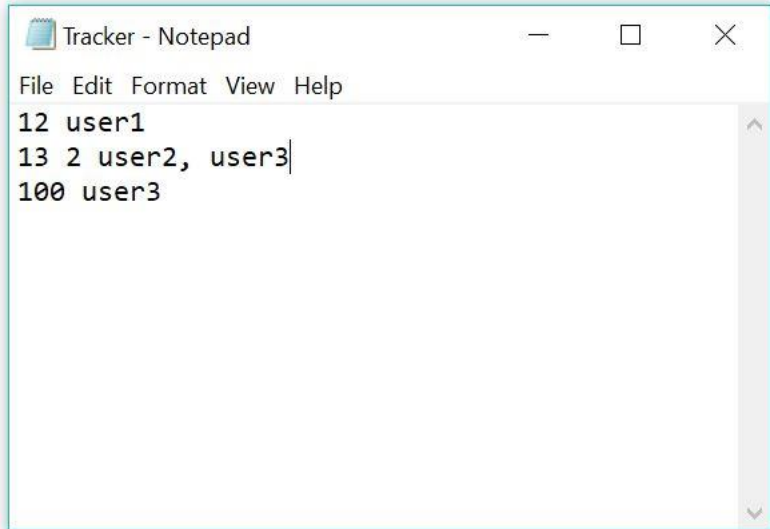
➤ File download and update **[2 marks]**

- Ex: user4 wants to download File3
 - User4 will use its hash function and convert name to hash key 100
 - User4 will send to all peers until it download File3
 - User4 will send a message to all the other peers "User4 has downloaded File3"
 - Tracker.txt will be updated (100 user3, user4)
 - User4 will update File3
 - Changes made to File3 will be applied to File3 at any peer (not only user4)



```
Tracker - Notepad
File Edit Format View Help
12 user1
13 user2
100 user3, user4
```

➤ If the file is divided into to several parts and the new user wants to download the whole file (bonus). **[+1 mark]**



```
Tracker - Notepad
File Edit Format View Help
12 user1
13 2 user2, user3
100 user3
```

- Ex: File2.txt is hashed to 13 and is divided to two parts one in user2 and one in user3 and user1 wants to download file
- User1 will use hash function and convert File2.txt to hash key 13
- It will use Tracker.txt and it find key 13 and then 2 which mean that file is divided to two parts.
- User1 will download two files and append them to each other to make full version of the file
- User1 will update Tracker.txt with new row "13 user1". Which means that user1 has the full version of file.

4. Distributed Password Cracker: (TA Salah)

- The distributed password cracker will simulate a server that has multiple computers helping him in cracking passwords of different encryption / hashing methods
- The application will have 3 parts
 - The server
 - cracker PCs
 - clients
- The application should have a protocol for communication and handling errors in sent commands. **[1 mark]**
- Any built in hashing method (ex: MD5) can be used, but must know its name and type. **[1 mark]**
- All cases/examples in cracker method must be handled. **[2 marks]**
- Client and server communication. **[2 mark]**
- All groups should have 3 test cases prepared for hashing cracker.
- Function to generate ciphers to test them **[+1 mark]**

Cracker methods (choose one of the following 3)

1- Hash cracker by brute force:

- a. The client will send a hash of a word and expected length of the word
- b. The server will run through all possible combination (within letter length) and try to find a matching hash.
- c. The test will use letters that include alphanumeric values (0-9, a-z, A-Z).
- d. Example1:
 - i. 1 cracker PC, hash sent is "098F6BCD4621D373CADE4E832627B4F6" and the expected length sent is 5 letters.
 - ii. The cracker PC tests from one letter '0' to 'Z' and then adds one letter and tests from word "00" to "ZZ until it reaches a match, or the end (5 letters "ZZZZZ").
 - iii. In this case the matching word is "test", so the server should return something like "Match found, the word is "test"". (the cracker PC will inform the server and the server will inform the client)
- e. Example 2:
 - i. 1 cracker PC, hash sent is "8DBDDA48FB8748D6746F1965824E966A" and the expected length sent is 4.

- ii. PC will test from "0" to "ZZZZ" and return result. In this case there will be no match as the original word was "simple" and it is 6 letters long.
 - iii. The server should return something like "No match found". (the cracker PC will inform the server and the server will inform the client)
- f. Example 3:
 - i. 2+ cracker PC, hash sent is
"9C8D9E1D80FC923CA7D532E46A3D606F" and the expected length is 5.
 - ii. The Server should distribute the load on all cracker PCs.
 - iii. In case there are 2:
 - 1. The first one will start from 1 letter "0" to 5 letter word "uZZZZ"
 - 2. The 2nd cracker PC will test from 5 letter "v0000" to "ZZZZZ"
 - 3. If anyone finds a match, it should send it to the server.
 - 4. If the server receives one match, he should inform all the cracker PC's that the specific task of cracking the hash
"8DBDDA48FB8748D6746F1965824E966A" was found so that they move on to the next task
 - 5. The server should inform the client of the original word.
 - 6. Otherwise if all cracker PCs found that there is no hashing method. The server will inform the client that the task was completed and no matching words was found.

2- Substitution Cipher breaker:

- a. The client will send a long paragraph and then the server will send it to one (only one) cracker PC.
- b. The cracker PC will count all the letters and use statistical knowledge based only on letter count, to try to translate all the letters to their possible match.
- c. The cracking here is not 100% accurate (it may even be less than 80% accurate). The important thing here is the method.
- d. You can find the statistics to compare the occurrence of each letter online, use it to translate each letter to what the statistics say it should be.
- e. Then apply that on the whole paragraph that was sent by the client.
- f. Then return both the translation of "A-Z" and the paragraph to the server which will return it to the client.
- g. Example:
 - i. "EDHZ WUYUCYUWD LYHEEAJ DAYA HJ EDS OHUQKCFA VKM HZ EK EAZE
HB EDA CYUS VQUNP BKM IFRWAO KGAY EDA BAJNA HJ EDA VQUNP

JHCDE KB IFRWHJC LDHQA U PJHCDE LUZ YHOHJC VS" this is the cipher text (it is just all capitalized)

- ii. The key you will try to find "EFGHTUVIJNOWXCDKLMYZABPQRS"
- iii. The original text after decoding: "this paragraph written here in thy dialogue box is to test if the gray black fox jumped over the fence in the black night of jumping while a knight was riding by"
- iv. Ignore upper case letters, make them all lower case and remove all spaces.
- v. The cracking here will not be 100% accurate.
- vi. Bigger test cases result in better %.

3- Integer factorization through brute force

- a. (use a long data type here don't use int)
- b. The client sends an integer of 16 digits
- c. The client should find and return all the **prime** factors (all the factors should be prime numbers) of that number. This will work on **1 random cracker PC**.
 - i. (Remember don't use any fancy mathematical models for integer factorization, just **test every possible number** and test if the number is prime or not through checking if it can be divided by other numbers or not.)
- d. Example 1 (smaller case):
 - i. User sends 210, client tests values from 2 to $\sqrt{210} = 15$
 - ii. It will find '2' divides $210/2 = 105$, then finds '3' divides $105/3 = 35$, then it finds 5,7
 - iii. It should return "2,3,5,7" to the server to return to the client
- e. Example 2:
 - i. User sends 96: pc tests, and finds "2" which makes 96 becomes 48, then it tests again and finds that 2 can work again making 48 become 24, it will find that 2 works 5 times, and the 3 works 1 times.
 - ii. So it will return "2^5, 3".
 - iii. Note that you cannot return 4,8,16,32 as they are not prime numbers. Only use prime numbers.

The Client and server communication:

- The client will connect to the server using the client port in the server.
- After the connection is successful the following should be displayed during the communication:

1- Welcome Screen: The server should show a welcome message and a help command to the user (h or help to show help)

a. If the user enters the command asking for help

- i. The server should return a guide on the commands available at this current point.
- ii. In this step the user can choose which type of cipher he wants, so the server should tell him what to enter to use that cipher.
- iii. Example for calling for help function:
USE <cracker name> ON <cracker parameter 1> <cracker parameter 2>
...
Crackers names available:
1- Factorizer: Takes 1 parameter, the integer to be factorized
2- Hash_Collider: Takes 2 parameters, the hash code and the original word expected length
Substitution_Cracker: Takes 1 parameter, a long paragraph. Indicate the start and end of the paragraph with '#'
iv. **An example on input:** USE Fact ON 153432
Server replies: Invalid command "Fact", use "help" to display guide or enter another command
USE Substitution_Cracker ON #EDHZ WUYUCYUWD LYHEE AJ DAYA HJ EDS OHUQKCFA VKM HZ EK EAZE HB EDA CYUS VQUNP BKM IFRWAO KGAY EDA BAJNA HJ EDA VQUNP JHCDE KB IFRWHJC LDHQA U PJHCDE LUZ YHOHJC VS#
Server replies: your request was successfully received, your request ID is 107
Server replies: For the request with ID 107, your substitution key approximation is "EFSHTUVIJROWXCMKLPYZABGQDN" and the original dialogue approximation is "THIN GADASDAGH WDITTER HEDE IR THY MIALOSUE BOX IN TO TENT IF THE SDAY BLACK FOX JUPGEM OVED THE FERCE IR THE BLACK RISHT OF JUPGIRS WHILE A KRISHT WAN DIMIRS BY"

b. The user inputs an invalid command, return a message indicating invalid command or syntax error in command and tell the user he can use the help keyword to display help/guide.

- 2- **Valid Command Entered:** The server returns a message telling the user that the request was successfully added to queue, and gives him a request ID.
- 3- **Results arrived at server:** If the server gets the result of the request, he should send them to the client alongside the request ID, since the client can send multiple request and will want to know which request was finished.

Server side view:

- The server will give each valid request from the client a request ID
- The server should associate the ID with the client who sent it so that he can deliver the result to that same client. Also it will be used alongside with the crackers to know what result was finished.
- The server will need to keep track also of the request type. (because the MD5 hash crack method uses more than 1 client and Also in the methods for substitution cipher cracking and integer factorization)

Server and cracker communication:

- The communication between the cracker and the server should be automatic.
- When a cracker starts, it will connect to the server on the port specified for crackers only.
- The server will ask for the password, and the cracker will reply with that password to confirm.
- If the password is false the server will disconnect the cracker PC.
- Otherwise the cracker PC will be on standby incase a requests comes in.
- The server will send cracking request ids alongside the parameters to the cracker(s) and reply to the user with the result when they arrive.
- The server should keep track of the cracker PCs that he is connected to, so he can send the requests to them.
- There are 2 cases for cracking requests.
 - cracking method
 - substitution and integer factorization.
- These two only use one cracker PC, so the results that will return from that one cracker will be send immediately to the client.
- The other case is the MD5.
 - The server will need to divide the range of the crack
 - The server will send each range to the cracker PCs and only reply to the client when all cracker PCs have returned no match, or if one PC returned match correct.
 - It is also preferable that you send to all cracker PCs to stop processing request 'x' for MD5 hash, if one PC found the answer to save computing power.

Cracker side view:

- The cracker will receive a request ID along with the request from the server and parameters of the request.
- These parameters may require to be different than what the client sent (since MD5 requests sent to crackers contains start and end range).
- The cracker should have a queue of the requests that he has, and that he goes through them and finishes them one by one.
- When the request is finished he should send the result alongside the request ID back to the server so that the server knows to whom to give those results.
- The cracker will be in a loop of waiting for requests and adding them to queue, and a loop of taking requests that are in queue (if there is any) and finishing them one by one.

5. DNS: (TA Mohamed Ashraf)

- UDP is used to handle queries
- In DNS implementations you can use **iterated DNS or recursive DNS**. [1 Mark]
 - Implementing a code using both is bonus. [+1 mark]
- In the project we will have client and multiple DNS servers [local DNS, root DNS, TLD DNS and authoritative DNS]. [1 Mark]
- The DNS which will reply back with the query answer, should tell the client which DNS server it is.
- Project needs to be implemented with **two DNS query types Query A and CNAME**.

Project Design Options:

[Option1]: If the client and the servers will be implemented on the same PC then each server will have a different port number to be reached through it.

[Option2]: If the client and the servers will be implemented on different PCs then each server will have a different IP address to be reached through it

- **Query type A: [2 marks]**
 - Client sends a host name to the server.
 - Server replies to client with: [1 mark]
 - IP
 - Query type which is "A".
 - Client should reach the local DNS server using either recursive or iterated method.
 - Local DNS server uses a file called "local_dns_table.txt" to get the IP information.
 - If local server didn't find the IP in local_dns_table.txt file, then we need to reach root DNS server that uses a new file called "root_dns_table.txt" and replies back with the IP address we are looking for.
 - If root server didn't find the IP in local_dns_table.txt file nor root_dns_table.txt then the server should reach another server called TLD DNS that uses file called "TLD_dns_table.txt" in case, he didn't find it on TLD then he need to reach authoritative DNS server that uses "authoritative_dns_table.txt" to find the IP for the host name.
 - In case the IP found on authoritative then the response should include the following: [1 mark]
 - IP address that we are looking for.
 - Query type "A" and type "NS" because the response came from authoritative.
 - Host name for the authoritative DNS servers.
 - IP address for the authoritative DNS servers.

➤ **Query type CNAME:[2 marks]**

- Name is alias name for some “canonical” (the real) name.
- We enter the hostname and we should receive a response including:
 - The IP of the server. **[0.5 mark]**
 - Real name of the server. **[1.5 marks]**
- Ex: www.ibm.com is really serveeast.backup2.ibm.com
 - Value: canonical name = serveeast.backup2.ibm.com

Example 1:

local_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

root_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

www.fci.com 192.168.10.10

TLD_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

www.fci.com 192.168.10.10

www.ibm.com 192.168.2.149 serveeast.backup2.ibm.com

authoritative_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

gaia.cs.umass.edu 128.119.245.12

=====

// Client

C:\dns_project>java Client

Enter Name / Enter 'quit' to exit

Client :

www.google.com

Reply from Server is : URL=www.google.com IP Address= 127.9.9.1 query type = A

Server name: local DNS

Enter Name / Enter 'quit' to exit
Client :
quit

// Server
C:\dns_project>java Server
Client Requested : www.google.com
URL :: www.google.com
Query type = A
IP Address :: 127.9.9.1

Client is Disconnected.....

Example 2:

local_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1

root_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1
www.fci.com 192.168.10.10

TLD_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1
www.fci.com 192.168.10.10
www.ibm.com 192.168.2.149 serveeast.backup2.ibm.com

authoritative_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1
gaia.cs.umass.edu 128.119.245.12

=====


```
// Client
C:\dns_project>java Client
Enter Name / Enter 'quit' to exit
Client :
gaia.cs.umass.edu
Reply from Server is : URL= gaia.cs.umass.edu IP Address= 128.119.245.12 query type =
A ,NS
Server name: authoritative DNS
Authoritative answer:
Name: authoritative_dns_table.txt
IP= 10.16.142.66

Enter Name / Enter 'quit' to exit
Client :
quit
```

```
// Server
C:\dns_project>java Server
Client Requested : gaia.cs.umass.edu
URL      :: gaia.cs.umass.edu
Query type = A
IP Address :: 128.119.245.12
Query type = NS
Found record on authoritative DNS servers:
Name: authoritative_dns_table.txt
IP= 10.16.142.66
```

Client is Disconnected.....

Example 3:

local_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

root_dns_table.txt

www.yahoo.com 127.0.0.1 13

www.google.com 127.9.9.1

www.fci.com 192.168.10.10

TLD_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1
www.fci.com 192.168.10.10
www.ibm.com 192.168.2.149 serveeast.backup2.ibm.com
authoritative_dns_table.txt

www.yahoo.com 127.0.0.1 13
www.google.com 127.9.9.1
gaia.cs.umass.edu 128.119.245.12
=====

// Client

C:\dns_project>java Client

Enter Name / Enter 'quit' to exit

Client :

www.ibm.com

Reply from Server is : URL=www.ibm.com IP Address= 192.168.2.149 query type = A,
CNAME

Server name: TLD DNS

Canonical name: serveeast.backup2.ibm.com

Aliases: www.ibm.com

Enter Name / Enter 'quit' to exit

Client :

quit

// Server

C:\dns_project>java Server

Client Requested : www.ibm.com

URL :: www.ibm.com

query type = A

IP Address :: 192.168.2.149

Query type = CNAME

Canonical name: serveeast.backup2.ibm.com

Aliases: www.ibm.com

Client is Disconnected.....

6. FTP: (TA Eman)

- Implement multiple clients that run on different devices to connect to FTP server.
- FTP uses two TCP connections for communication (2 ports), One to pass control information (ex: username & password) and the other for data connection to send the data files to client.
- FTP Server will have 2 files, the first to save the authorized Clients with their username & password, and the second will save the dirs. that allowed to each client (minimum 4 clients and for each client 3 dirs. and into each dir. 3 files).
- **The process:**
 - a. Client sends its username.
 - i. Server will search in the first file if this username exist
 - If it exists, it will request the password.
 - b. Client sends its password.
 - i. Server will compare this password with the password that belong to the username
 - If they both match, it will send “Login Successfully”. **[1 mark]**
 - Note: if the username or password is wrong, the server will reply with “Login Failed and the connection will terminate”.
 - c. Client will enter the command “show my directories” to list all the available dirs. **[1 mark]**
 - i. Server will retrieve list all the available dirs..
 - d. Client will send the wanted dir. to the server. **[1 mark]**
 - i. Server will reply with the available files.
 - e. Client sends a request which includes the name of the targeted file (download file). **[2 marks]**
 - i. Server responses with the file content.
 - f. Client will receive the previous data & write it into file (bonus). **[+1 mark]**
 - g. Client enters the command “Close” to close the connection that is created on the second port#.
 - i. It means that the client can download another file without login again. **[1 mark]**

Example of downloading new file “Joker” that exists in “Movies dir.”:

Client: login by entering its Username.

Server: Username OK, password required.

Client: entering its Password.

Server: Login Successfully.

Client: enters the command “show my directories”

Server: replies with the available dirs. (Movies, music, Docs, ...).

Client: enters the command “show Movies”

Server: replies with the available files into this dir. (Joker, WAR, Free Fire,...).

Client: send the file name (Joker).

Server: reply with the file content.

Client: enters the command “close”

Server: “Connection is terminated”