Computer Networks
Spring 2024 **Due Date: 02/25/2024**

Programming Assignment # 2: Hashing Server

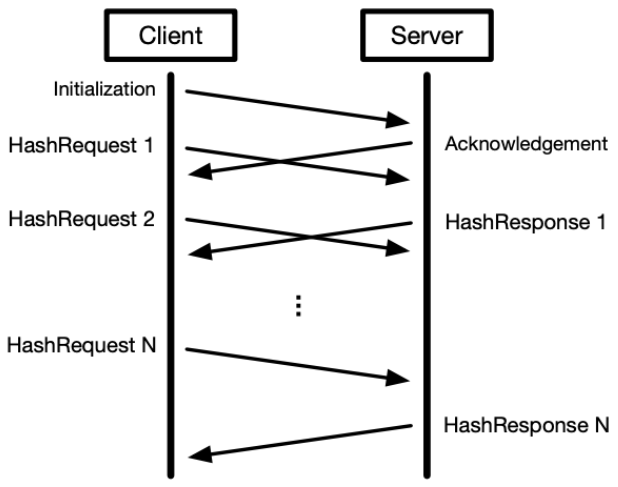
Description.

In this assignment, you will be writing a TCP server and client, where the TCP server's job is to take input requests from a TCP client to generate hash and send it back to the client. The client has a file and wants to obtain the hashes for various segments of the data inside the file. The client will send a message to the server for each segment, and the server will respond with the hash of the segment. The client converts each hash into a hexadecimal string representation and prints it to the console. The server must be able to handle more than one client concurrently.

The details of the protocol followed in this case is described below.

Protocol Overview.

The protocol between the client and server will use four types of messages: *Initialization*, *Acknowledgement*, *HashRequest*, and *HashResponse*. The client starts by sending an *Initialization* message to the server, which informs the server of the size of the input data that the client will send (denoted by the variable **S**). Afterwards, the client sends **N** *HashRequest* messages to the server, where each *HashRequest* contains the data segment to be hashed. The server responds to the *Initialization* message with an *Acknowledgement*, which informs the client about the total length of the response (40*S). After this, the server responds to each *HashRequest* with a *HashResponse* that contains the hash of the corresponding data included in *HashRequest*.



Format of messages.

For the sake of convenience, all the messages in this protocol follow same structure as shown below. However, some of the attributes in the messages will be different depending on the type of message.

Initialization (Client to server)

Type=0x1	S	Length	Data (length)

1. Type: A <u>4-bit value</u> in network byte order that is set to the value 1.

2. S: A 4-byte integer value in network byte order that corresponds to the size of each *HashRequest Segment* that the client will send to the server.

3. The other fields must be 0, and an empty 32-byte payload respectively.

Acknowledgement (Server to Client)

Type=0x2	i	Length	Data (length)

1. Type: A 4-bit value in network byte order that is set to the value 2.

2. Length: A 4-byte integer value in network byte order that denotes the total length of all *HashResponses*. This should be equal to 40*S.

3. Fields “i” and “Data” must be 0, and an empty 32-byte payload respectively.

HashRequest (Client to server)

Type=0x3	i	Length	Data (length)

1. Type: A 4-bit value in network byte order that is set to the value 3.

2. i: A 4-byte integer value in network byte order that denotes the zero-based index of the request. That is, the first *HashRequest* should have this set to 0, while the last *HashRequest* should have this set to N - 1.

3. Length: A 4-byte integer value in **network byte order** that denotes the length of the Data payload in number of bytes. (You can assume something on your own)

4. Data: A payload that holds the data segment to be hashed (of length defined in #2 above).

HashResponse (Server to Client)

Type=0x4	i	Length	Hash_of_i (32 bytes)

1. Type: A 4-bit value in network byte order that is set to the value 4.

2. i: A 4-byte integer value in network byte order that denotes the one-based index of the response. That is, the first *HashResponse* should have this set to 1, while the last *HashResponse* should have this set to N.

3. Field “*Length*” must be 32, as 32-bytes of data is included this packet.

4. *Hash_of_i*: A 32-byte (256-bit) value that corresponds to the hash of the data contained in the ith *HashRequest* sent by the client.

Output: The hash values received can be stored in an output file ("*outfile-clientIP.txt*") along with the original text. It can follow the format of

---

Text-segment1: hash-of-text-segment1

Text-segment2: hash-of-text-segment2

Text-segment3: hash-of-text-segment3

------- and so on ---

Server Implementation
The server program executes as follows and should ask for two command line arguments

a. -p <number> : port number that server will bind and listen on. Should be any value above 1024
b. -s <String> : a salt value that server will use in computing hash.

Example Usage:

```
$ python3 server.py -p 9000 -s abc23432salt
```

Important: The server must be able to handle more than one client concurrently. This means, you should have your server program leverage "select" or "multithreading" to meet this criterion. In addition, your test scenarios should show that your program has the ability to handle multiple clients' requests.

Client Implementation
Client program will take the below command line arguments:

a. -p <number> : port number of the server to connect
b. -a <string> : ip address of the server
c. -s <number> : The size of each *HashRequest* Data Payload that the client will send to the server
d. -f <File> : The file that the client reads data from for all *HashRequests*. Must be specified and have enough data to support N requests of random length between 512 to 2048 bytes.

An example usage is as follows:

```
$ python3 client.py -a 128.8.126.63 -p 41714 -s 100 -f /home/file.txt
```

Template Code:

A template server and client code are provided here. You must use them to build your programs. A sample file is provided to read and

What to be submitted:

1. A PDF formatted report with all evidence, codes, execution samples, multiple test scenarios, instructions for running the submitted programs, and references used.
2. Readability of the program is highly valued. Program comments has a weight of 10% of total points.
3. Submit your implemented server and client programs with well-defined documentation.
4. Submit everything (code and report) in **Blackboard.**