CS 39006: Lab Test 2
Part - A
Date: April 7, 2023, Time: 10 am to 11 am

IMPORTANT:
Please write your name and roll no. in a comment as the beginning of every file you write.
Follow the submission instructions exactly, otherwise a small penalty may be imposed.

You will write a client-server system that simulates a very simple online voting system.

The server S (a TCP concurrent server with threads (not processes)). It first reads a positive integer n ($n \leq 10$), and then reads a list of n names (first name and last name, separated by one or more spaces) one by one from the keyboard and stores them in a table. These names are the names of the candidates standing in the election. You can assume that each part of a name (first name or last name) has at most 20 characters, so it is easy to define an array of structures to store the names. Each candidate then is identified by (1 + the index in the array where the name is stored). So for example if n = 2, and the names entered and stored (in that order) are "Adam Shandler" and "Harry Potter", then the id of Adam Shandler is 1 and that of Harry Potter is 2. The server also creates another thread called *Checker* (its purpose is given later)

The client is the voter. The server also maintains a table T of clients who have already voted which stores the following entries for each client: Its IP address and port and the integer id of the candidate it has voted for. You can assume that there can be at most 500 voters.

The server then waits for a client connection. When a client connects, the server creates a separate thread to handle the client (it is a concurrent server). The thread T does the following:

- Checks if the IP address and port of the client already exists in T.
- If yes, simply closes the connection, and exits.
- If not, it sends the names of all the candidates to the client. The names are sent in this format: each name is sent as one integer (sent as a 4-byte integer in network byte order, not as string) which is the id of the candidate, followed by the name (both first and last name separated by exactly one space) as a null-terminated string. The end of all names is indicated by sending the integer 0 followed by an empty string (only '\0') at the end. You must send the names this way, nothing else will be accepted.
- It then waits to receive the client vote.
- If no vote is received from the client within 1 minute, the thread closes the connection and exits.
- On receiving the client vote (see format in client part), it checks if the integer id received is a valid id (of one of the candidates). If not, it closes the connection and exits.

- If it is a valid id, stores the client's IP (can store in any form), port (in host byte order), and the client's vote in table T.
- It then prints the client's vote on the screen in the format "Client <IP, port> voted for <first name, last name>" (replace the parts inside <…> appropriately) where IP address of the client must be in dotted decimal format.
- It then sends the string "Vote recorded successfully" as a single null terminated string and closes the connection and exits.

(This is not really a good voting system at all, just retrying will get someone to vote again and again, but that's ok, it is just a lab test ☺)

The *Checker* thread simply does the following: it sleeps for 5 minutes, wakes up, counts the number of votes received by each candidate so far, and prints it on the screen.

The client does the following:

- Connects to the server and waits to receive.
- If the connection closes before anything is received, should print "Could not vote: retry after some time" and exits.
- On receiving the candidates' names, displays them on the screen (integer id followed by full name), and asks the user to vote (chose an id)
- The user enters the integer id of the candidate it wants to vote for from the keyboard.
- The client sends the integer id to the server (as an integer, not as a string).
- The client then waits for the response from the server. If received, it prints the message on the screen and exits.
- At any intermediate point, if there is any connection close detected for any unknown reason, print an error message and exit.

The following should be noted:

- Ensure that the messages are sent exactly in the format specified.
- Client cannot assume it knows how many candidates are there, though you can assume it knows the upper bound (the value of *n* on the server side).
- You can only assume that a single 4-byte integer, when sent in a single send() call, will be received together in a single recv(() call. For everything else, remember how TCP works.

You should submit two files, named *client.c* and *server.c* in the moodle link provided. Make sure to have your name and roll no. written as comments in the first lines of each file.