CS39006: Networks Lab

Lab Test 2

April 12, 2024

Time: 90 minutes **Full marks: 50**

IMPORTANT: Please write your name and roll no. in a comment as the beginning of every file you write. Name your files strictly as per the instructions given in the submission instruction at the end.

You have to design a TCP server and client to implement an online voting system. The details are given below. Assume that the server will run at IP address 127.0.0.1 and port 30000.

Votes are cast for an Entity. Each entity has a name, a null-terminated string of at most 200 characters containing alphabets only (can be lower and uppercase). The names of all current entities against which a vote can be given is stored by the server in a table, along with the current number of votes against them (initially 0). Name this table *Vote_Info_Table*. You can assume that the maximum number of entities against which votes can be given at any one time can be at most 10. Assume for simplicity that the table is not stored in a file, so that if the server closes, all information is lost, and the voting system just starts from scratch.

The administrator of the voting system should be able to (i) print list of entities and the current number of votes received by each entity, (ii) add new entities to the table, and (iii) delete an existing entity from the table, through the keyboard. A menu should be shown to the administrator user to choose from the above three. If the user chooses any one, the corresponding action should be taken (with associated input like entity name to add/delete, or display like the list etc. as needed), and then the menu should be shown again for the user to choose from.

A client should be able to see the list of entities for which votes can be given, and be able to cast a vote for exactly one of them.

The server will be a TCP server with a single process (no additional process or threads should be created). The server will run indefinitely but you can assume that not more than 5 clients will be connected to the server at any one time (though the total number of clients over time can be much more).

The functionality of the server will be as follows. Use the names given exactly wherever shown.

1. It will wait on a socket named *vote_req* to receive a connection from a client. On receiving a connection, it will send the list of all entity names to the client in two send() calls. The first send() call will send one integer (in network byte order) stating the total number of entity names present in the table (You can assume that the 4 bytes of an integer will be received together at the client end). The second send() call will send all the entity names together, with each name sent as a null-terminated string. Nothing else should be sent. It will then wait indefinitely for the client to send an entity name to vote for, or to close the connection. A connection close before receiving anything from the client indicates that the client does not want to vote for any of the entities. When the server receives a name (a null-terminated string), it checks if the name matches (case-sensitive) that of an existing entity in the table. If yes, it increments the vote for that entity by 1, and sends the string (null-terminated) "Vote registered successfully" to the client. If no, it sends a string (null-terminated) "Problem in voting. Try again later" (Note that an entity may not exist in the table even if its name was sent to the client as it may be deleted by the administrator after it was sent to the client but before the client voted for it). In both cases, it then waits for the client to

close the connection. When the client closes the connection (see below), it removes any information about the client that it may have saved (if any). A client can vote for only one entity in one connection.

Note that the above describes the behaviour of the server with respect to one client. Other clients may want to vote at any point of time while the above interactions are going on, they should not have to wait for any other client to complete.

2. At any time during any of the above, the server must be able to take inputs from the keyboard from the administrator user through a menu to (i) print list of entities and the current number of votes received by each entity, (ii) add new entities to the table, and (iii) delete an existing entity from the table through the keyboard The inputs should be given through a menu shown to the administrator user.

The functionality of the client is as follows. It first connects to the server. After this, it waits to receive the number of entities and the list of entity names from the server. Once received, it shows the list of names to the user and asks the user if it wants to vote. If the user does not want to vote for any of the entities, it simply closes the connection. Otherwise, the user selects one of them to vote for. The client sends the name of that entity (exact one as sent by server, case-sensitive) to the server as a null-terminated string, and waits for the response. It shows the appropriate response to the user (see server details above) and then closes the connection and exits.

Submission Instruction:

You have to submit the following two files: <your roll no>_vserver.c containing the server code and <your roll no>_vclient.c containing the client code. For example, if your roll number is 21CS30002, then the files should be named 21CS30002_vserver.c and 21CS30002_vclient.c. Upload both the files through the assignment submission link in the Moodle submission page (The assignment will accept up to 2 files, do NOT zip).

Header files to use

You are free to use other/alternate header files if you feel so. In any case, for every call you use, you can easily look up the header file to include from the man page of that call.

```
#include <stdio.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/select.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <unistd.h>
```