**ASSIGNMENT 1**

**Last date: 12 April (By 5:00pm)**

**Assignment**

The goal of this assignment is to implement a client server application. You can use either C or C++. The server will start in *passive* mode listening on a specified port for a transmission from a client. Separately, the client will be started and will contact the server on a given IP address and port number that must be entered via the command line.

* The server application should run and respond to the client application. Basically you have to show inter-process communication between client and server processes.
* A client will give commands to the server process and the server must execute that commands. Server will run on a separate terminal and client will run a separate terminal on the same machine.
* Server has maintained one database in which tables like Telephone, Address, Email and Date of birth are stored. For example

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **telephone** | | | joe | 4711 | | bill | 11147 | | |  |  | | --- | --- | | **email** | | | joe | joe@foo.bar | | bill | bill@cookie.duh | |

* For communication with server client has to register itself with server, firstly client have to log in with id and password. If id and password of client is correct then client can communicate with server otherwise server sends a “Sorry your id and password is not correct” response to the client. Multiple clients can access the database simultaneously.
* Client can send only three commands to server: <Get, Table Name, Name>, <Put, Table Name, Attribute Name> and <Delete, Table Name, Attribute Name>.
* For each operation you should print the result, EXISTS/NONEXISTENT/OK/FAIL/ERROR as appropriate, on a line by itself. Any following lines are assumed to be the data returned or messages. A newline should follow whatever data is printed. I.e. for PUT, DEL etc only an empty line should be printed, unless an error occurred.

|  |  |
| --- | --- |
| **Request** | **Response** |
| GET(table, key) | Return EXISTS and set Data field in response if <key,data> pair exists in table.  Return NONEXISTENT if <key,data> pair does not exists in table.  Return ERROR and set Data to an appropriate error message in all other cases. |
| PUT(table, key, data) | Return EXISTS if some <key,data'> pair exists in table.  Return NONEXISTENT and insert <key,data> into the table if no pair <key,data'> exists in table.  Return ERROR and set Data to an appropriate error message in all other cases. |
| DEL(table, key) | Return EXISTS and remove <key,data'> pair from database, if a pair <key,data'> exists in table.  Return NONEXISTENT if no <key,data'> pair exists in table.  Return ERROR and set Data to an appropriate error message in all other cases. |

* If errors are detected internally in your client you should print CLIENT\_ERROR on a line by itself, followed by the actual error message. If the error occurred during your conversation with the name server, you should print NAMESERVER\_ERROR on a line by itself, followed by an error message.

#### Examples:

Machine1> client1 128.111.49.44 32000   
From server: Do you want to Register or Log in

From client1: Register

From server: Enter your id and password

From client1: abcd and \*\*\*\*\*

From server: Registered successfully

From server: Enter your id and Password for log in

From client1: abcd and \*\*\*\*\*

From server: Successfully logged in.

From client1: GET "telephone" "joe"

From server: EXISTS

4711

From client1: DEL "telephone" "joe"

From server: EXISTS

From client1: GET "telephone" "bjorn"

From server: NONEXISTENT

From client1: GET "feleton" "joe"

From server: NAMESERVER\_ERROR

Non-existent table 'feleton'.

From client1: GET "joe"

CLIENT\_ERROR

Too few arguments.

Correct syntax is: GET <table> <key>

From client1: LOG OUT

From server: Successfully logged out

**Errors**

Between the times you send a request until you receive the answer, a number of things may happen:

* The server might go down (either because the machine crashed, or because the server program terminated)
* The request or the response may be discarded in an overloaded router
* The request or the response may be duplicated, delayed or garbled while in transit.

Unfortunately, the network will not inform us when any of these errors occur, we have to discover for ourselves what happened.

We define TIMEOUT to be an upper limit on the time we will wait for a response to a single request before we consider it lost. We define RETRIES to be the maximum number of times we will try to send a request to a given server before we consider it or the link to it to be down. You should never send more than RETRIES requests to a server without getting a correct reply.

The only case in which you may report a complete failure is if you have sent RETRIES request to the same sender and waited up TIMEOUT seconds per request.

If the server is down, it will take RETRIES\*TIMEOUT seconds to determine this. If you receive irrelevant answers (i.e. duplicates, garbled packages etc) you may either reset or restart the timer after discarding it. I.e. you may in this case wait more than TIMEOUT seconds before timing out.

This will give you the necessary parameters to determine if the server is down, and when to consider a packet lost as well as a simple mechanism to recover from lost requests. The remaining problems you will have to figure out how to handle yourselves.