John Parry

Dr. Nemo

Network Fundamentals

18 February 2019

Project 1 – Sockets Quiz Program

QSERVER SETUP:

The server program and the client programs were both written using Python2.7 and connect using TCP. When the server is started, it first checks for the debug flag [-d] to see if it should print any debug information while on. Next, it connects to the database "qbank", creating the file if it did not exist previously. A cursor for the sqlite3 data base is then instantiated and the program runs a check to see if the database has been setup yet. It does this by querying the database to see the total number of tables which should equal three. If the database has no tables, then it creates the following tables:

1. Questions (Number INTEGER PRIMARY KEY ASC, Question TEXT, Correct TEXT)
2. Tags (Number INTEGER PRIMARY KEY ASC, tag1 BLOB, tag2 BLOB, tag3 BLOB, tag4 BLOB, tag5 BLOB)
3. Answers (Number INTEGER PRIMARY KEY ASC, A TEXT, B TEXT, C TEXT, D TEXT)

After the database setup has been verified, the program checks for the host name [-h] and port [-p] flags. If either is present, then the values are adjusted according to the arguments following each flag; otherwise, host name defaults to "storm.cise.ufl.edu" and the port number to 12000. It then attempts to bind to the given port which on failure will attempt connection to hostname:(port+1) iterativly until an open port is discovered. Once the program binds to the port, it prints the host name and port that it is connected to, enters an infinite loop, and waits for a connection request.

QCLIENT:

Upon running qclient, it will take exactly two arguments. Any less will trigger an error and the program will exit. Next, the program will take the parameters and send a connection request to the specified host and port. Connection refusal will trigger an error and the program will exit. If the

request is accepted, then the program enters an infinite while loop in which it does a few things. First, it waits for user input given a prompt '> '. Upon receiving input, the elements of the input are split by the white space between them and stored in a request array. If the first element of the array is 'p', then the program waits for the user's input for tags, a question, choices a-c, and the answer before reconfiguring the request array as [req,tags,question,a,b,c,d,correct] where req is the 'p' that was initially requested. The request array is then passed to function buildRequest(arr) which builds a string containing all of the elements separated by '\\\\'. The clientSocket and new request string are then passed to the function sendRequest(socket, message) which encodes the message and sends it over the socket. Once the request has been sent to the server, the program calls function getResponse(socket) which takes the clientSocket as a parameter, receives 2048 bits of data, decodes it, and returns the response string. If the response string reads "EXIT", then the program will close it's connection to the socket and break from the while loop which ends the program. If the response string does not read "EXIT", then it is printed to the user and program jumps back to the top of the loop.

QSERVER:

Once qserver receives a connection on it's socket, it enters a second infinite loop specific to this session and waits for a request. Functionality in the beginning is similar to qclient as function getRequest(socket) is called which receives 2048 bits of information, decodes it, and then splits it into a request array delimited by '\\\\'. Once the program has the request, it passes the first element of the request to what is effectively a switch statement since Python doesn't support switch statements. If the request is "k" or "kill", the "EXIT" is sent in response, the connections to the port and the database are closed, and the program exits with status 0. If "q" or "quit", then the program reacts similarly except that it breaks from the inner loop instead of exiting which sets it up to be listening for a new connection.

If the request is type "p", then the request array is passed to function addQuestion(newQuestion). The function will first call another function which queries the database for the lowest available id

number. It the splits the tags element of the array by delimiting it by "," or ", ". An error check is then run on the tags to make sure there are exactly 5 entries to be passed to the database. If there are less than 5, the remaining spots are filled with empty strings and then tags is redefined as the first five elements of itself in case more than 5 were supplied. Next, using some array manipulation, three arrays are build containing the necessary information to insert into each table for the new question. The inserts are committed to the database and the id found in the beginning is returned as a string.

The functionality of the next few choices are not quite as extensive. If the user requests "d" or "delete", then the question number is passed to a function called deleteQuestion which checks it's existence before deleting the appropriate entry from each of the three tables in qbank. The existence check is also done in getQuestion and checkAnswer which on failure returns "Error: Question n not found" to the user. If the question is successfully deleted, then "Question n deleted" is instead returned to the user. If the user makes a get request, getQuestion performs the existence check and then queries the three tables for the appropriate entries. It then run a process that is a reversal of it's client counterpart which takes the elements derived from the sql database and rebuilds the string to display the question as it was input with the id at the top instead of the bottom. Once the string is rebuild, it is sent back to the user. A response to checkAnswer is similar to getQuestion though it only queries the Questions table and returns either "Correct" or "Incorrect" to the user if their answer matches that in the table. The function also case-corrects before checking so both "a-d" and "A-D" are valid responses. A request for a random question works similarly to getQuestion, but with a little extra setup. The first check is to see if the Questions table is empty. If so, "Error: No questions exist" is returned to the user. If there are questions in the table, then the "Number" column of the table is queried and the id's are stored in an array. A random integer is generated as the index to choose from and the id at that index is passed to getQuestion which will query and build the question to be returned to the user. If the user sends a help request with "h" or "help", then a hardcoded help page is returned as a string to the user. If the user requests anything other than the previous valid requests, they are returned "Error:

Invalid Request”. To handle any other unforeseen errors, each case aside from kill and quit are surrounded by a try-catch statement. These statements will catch any error that takes place as ‘e’ and on error, returns “Error: <String e>” to the user before continuing to the top of the while loop.