ICS 53, Winter 2020
Assignment 5: Client/Server programming

You will write a networked client/server application which allows a client program to query a simple database which is managed by a server program. For this assignment we selected a database of NFL games as an example. The client program sends a message containing the query about a match to the server and the server program responds back a message containing the requested information. Below, we will describe the details of query messages and the database.

1. Running the Client/Server

The application is actually two different programs, the client program and the server program. These two programs should both be running as two separate processes which are invoked by the user in a shell. These two programs may be running on the same machine but they may be running on two different machines. Since the client and server are communicating using TCP/IP, you will need to select a port to use for communication. ICS Computer Support has advised us that ports 30000 – 60000 are all available for your use, so you can use any one of those ports.

For the purposes of this explanation we will refer to the domain name of the server machine as “server.ics.uci.edu”. Although we use this name in this description, your program should operate correctly for any domain name. We will also assume that the port used is 30000, but your program should be able to communicate on any port.

Database as a .csv file

For this assignment we use a comma-separated values or csv file as an input to populate our database. This file will have the following fields separated by commas where each line has information regarding one match. Assume maximum number of lines is 200. The 1st row of csv is the name of each field and remains unchanged. It won't be read into the database. (The 1st line is still included in the MAX_LINE_NUM, 200.)

type	game_id	home_team	away_team	week	season	home_score	away_score

1. type: specifies if this match was a regular or post season match.
    - Possible values: [reg, post]
2. game_id: unique identifier for a specific match.
    - Example: 2018122400
3. home_team: short name of home team for each match.
    - Examples: GB, NYJ
4. away_team: similar to home_team but specifying the away team.
    - Examples: LA, KC

5. week: week number in the season that match was held.
    - Possible values: [1 to 20]
6. season: year that match belongs to.
    - Examples: 2005, 2006, 2019
7. home_score: number of points home team scored in that match
    - Possible values: [0-1000]
8. away_score: number of points away team scored in that match
    - Possible values: [0-1000]

Sample:

type	game_id	home_team	away_team	week	season	home_score	away_score
reg	2018090600	PHI	ATL	1	2018	18	12
reg	2018090900	BAL	BUF	1	2018	47	3
reg	2018090907	NYG	JAX	1	2018	15	20
reg	2018090906	NO	TB	1	2018	40	48
reg	2018090905	NE	HOU	1	2018	27	20

Starting the Server

If we assume that the name of the server's compiled executable is "server" then you would start the server by typing the following at a linux prompt,

"./server data_base.csv 30000".

When the server is started, it should print "server started\n" and then wait to receive messages from a client. One example of data_base.csv is provided in the assignment files.

Starting the Client

If we assume that the name of the client's compiled executable is "client" then you would start the client by typing the following at a linux prompt:

"./client server.ics.uci.edu 30000".

Remember server.ics.uci.edu is the domain name of the machine that server is running on. In most cases if you run both client and server on the same machine, you can use "localhost" instead to loop back the connection between client and server. When the client starts it should print a ">" prompt on the screen and wait for input from the user. (Note: "> " is one > and one space.)

2. User interface

User Interface of the Client

Requests for match information are made by the user entering *game_id* and *field* info at the client's prompt. When this information is entered at the client's prompt, the client will send a

request message containing the *game_id* and *field* to the server, and the client will await a response from the server. The server will send a response message containing the corresponding information of the *field* for the match with that *game_id*. The client will print the received information to the screen, print a prompt on a new line, and wait for more input from the user. An example of a user interaction with the client is shown below. It is derived from the first row of database examples.

> 2018090600 type
reg
> 2018090600 game_id
2018090600
> 2018090600 home_team
PHI
> 2018090600 away_team
ATL
> 2018090600 week
1
> 2018090600 season
2018
> 2018090600 home_score
18
> 2018090600 away_score
12
> whats up?
unknown
> 2018090600 away_score
12
> quit
$ ←*you are back to the linux prompt.*

The client will continue in this loop until the user enters “quit” which will cause the client’s process to exit. (Note: No need to print anything after quit was entered.)

User Interface of the Server

Once running, the server will only accept requests sent from one client over the network. When the server receives a request from a client, the server will print the *game_id* and *field* in the message on the screen on a new line. An example of the printed output of the server when communicating with the client is shown below.

server started
2018090600 game_id

2018090600 home_team
2018090600 away_team
2018090600 week
2018090600 season
2018090600 home_score
2018090600 away_score

The server will continue responding to requests and printing the associated information until its process is killed externally, for instance by a ctrl-c typed at the keyboard.

3. Message Format

Each message sent between the client and the server must contain two pieces of information, a string, and the length of the string. The string will be the *game_id* and *field* inside a request sent from the client to the server, and the string will be the value of that field for the corresponding match response sent from the server to the client. Each message must be shorter than 256 bytes long and must be formatted as follows:

- Byte 0: Length of the string
- Bytes 1 – n: Characters of the string

4. Implementation Details

1. If the *game_id* or the *field* typed into the client is not in the database which is known to the server, just return “unknown” and continue the program.
2. Don’t worry about invalid input typed into the client. Everything entered into
the client can be assumed to be a valid input.
For any corner cases, you can use “unknown” for commands that do not fit the description of the assignment and “unknown” for any information requested that do not exist.

5. Submission Instructions

Your source code must be two c files (client.c and server.c). Be sure that your program must compile on openlab.ics.uci.edu using gcc version 4.8.5. You can check your gcc version with the “gcc -v” command. Make sure both files can compile with “gcc filename.c” and do not require any flags or modifications. Submissions will be done through Gradescope. The first line of your submitted file should be a comment which includes the name and ID number of you and your partner (if you are working with a partner).