Computer Networking Homework 1

Documentation

<table>
<tr><td>Author</td><td>Yun-Chih Chen</td><td>Contact</td><td colspan="3">b03902074@ntu.edu.tw</td></tr>
<tr><td rowspan="2">Tool</td><td colspan="2">Qt5
client-side GUI</td><td>Language</td><td colspan="2">C++</td></tr>
<tr><td colspan="2">Google Protocol Buffer
facilitating common data structures between server and client</td><td rowspan="3">How to run?</td><td>server</td><td><code>make run_server</code></td></tr>
<tr><td colspan="2" rowspan="2">How to build?</td><td rowspan="2">just run make
(if you don't have Protocol Buffer, please first run make dep to install it. If you're compiling on CSIE workstation, this would not be an issue.)</td><td>client</td><td><code>make run_client</code>
(If you run this on CSIE workstation, please forward your X server using the ssh option -X)</td></tr>
<tr><td>2 clients + 1 server</td><td><code>make run_both</code>
(Tmux required)</td></tr>
</table>

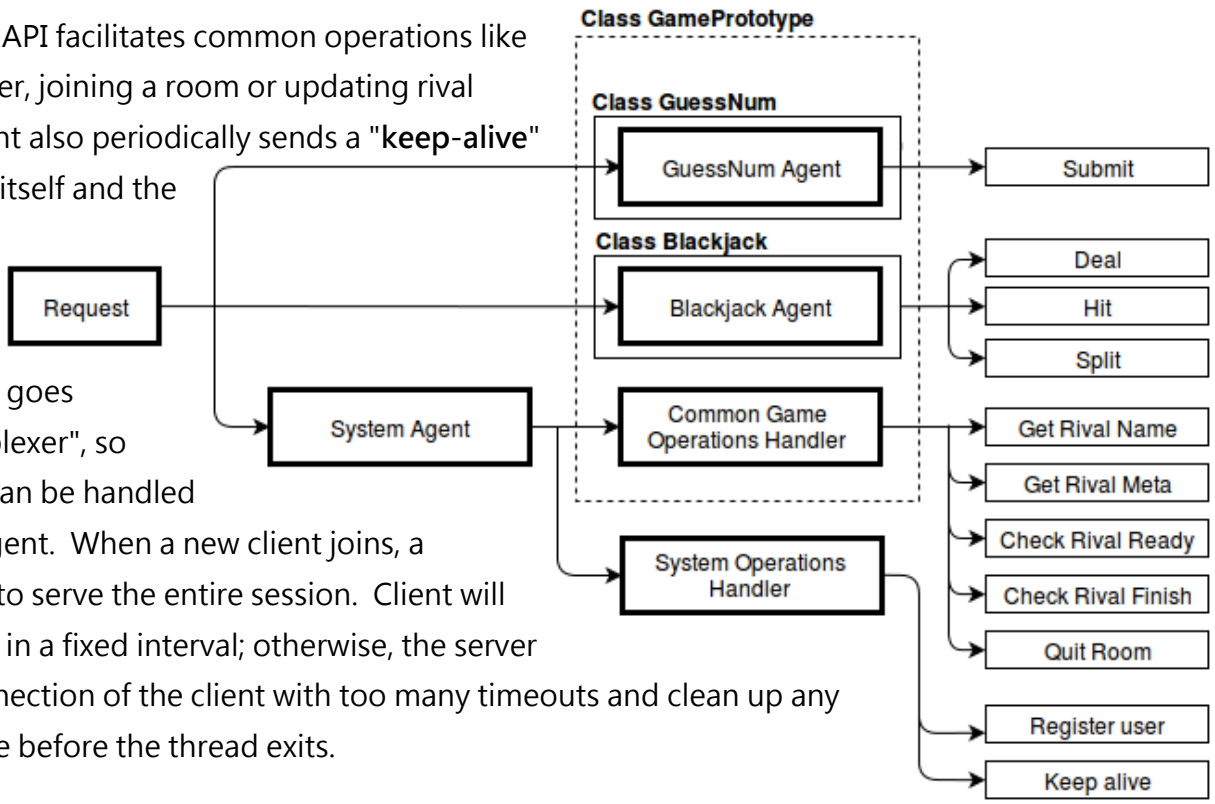
Overview

First of all, unlike traditional web app, the transmitted data via socket is not in plain-text; nor is it mirror of underlying memory structure. It is a special serialized memory-efficient binary that can be encoded and decoded by Protocol Buffer, regardless of the Endianness of host OS. To ensure data integrity, every outgoing data is prefixed by its length, so that the receiver can estimate when full data is received before it issues timeout.

Server Overview

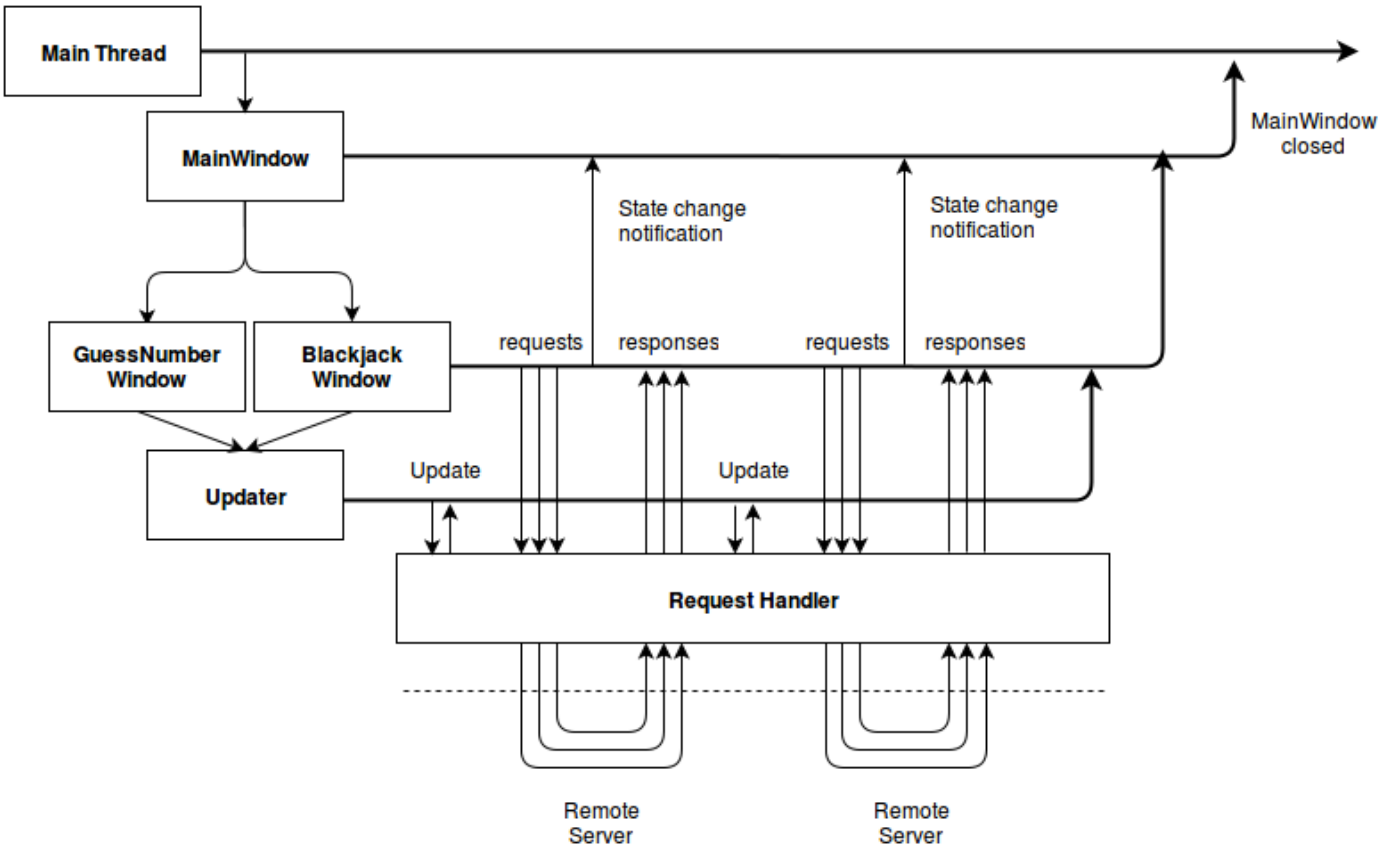
The server provides client with three types of APIs: **System-API**, **Blackjack-API** and **Guess-number-API**.

The System-API facilitates common operations like creating a new user, joining a room or updating rival information. Client also periodically sends a "**keep-alive**" request, assuring itself and the server that the connection is still valid. Every incoming request goes through a "multiplexer", so that the request can be handled by appropriate agent. When a new client joins, a thread is created to serve the entire session. Client will send "keep-alive" in a fixed interval; otherwise, the server will close the connection of the client with too many timeouts and clean up any allocated resource before the thread exits.



Client Overview

Client GUI programming is probably the most problematic aspect of this project. I have to take many situations into account, such as the case where user abruptly closes the window, or lose Internet connection, or other predictable state transitions in the games. In addition, the client program is highly multi-threaded, which severely complicates debugging. They can be roughly divided into four categories: main thread, GUI thread, updater thread and request handler threads. (request handler threads are actually workers that reside in a thread pool.) Inter-thread communications are critical,



too. It is carried out by Qt' s internal signals & slots mechanism, which sometimes lead to memory corruptions. In the mean time, inter-thread propagations between main thread and request handler thread are carried out by lambda callback function.

Data race or memory leak are very common during development. I try to do the best I can, but it always seems that the program is on the verge of crashing. Nevertheless, the program does fulfill the requirements of this homework. To mimic a multi-player atmosphere, the updater thread periodically requests for rival information from server. Such mechanism also notifies the player when his rival has left the room or there is any connection problem.

Further enhancement

UI design is all about responsiveness. When something goes wrong, instead of hanging helplessly, the program shall at least crash gracefully. When user triggers some state change, instead of dumbly waiting for backend response, the program shall at least give some feedback. My current work, although it rather suffices for the official requirements, is far from these criteria. These will be left for future enhancement.