Offline Messenger

Radu Marian-Sebastian

December 2024

1 Introducere

Proiectul offline messenger este o aplicație CLI (Command Line) care permite utilizatorilor conectați la rețea să comunice cu alți utilizatori din rețea prin intermediul unui set de comenzi. Obiectivul proiectului este de a face posibilă comunicarea în timp real a utilizatorilor conectați la rețea.

2 Tehnologii Aplicate

2.1 Protocolul de comunicare

S-a folosit TCP (Transmission Control Protocol), deoarece față de UDP (User Datagram Protocol), garantează livrarea completă și în ordinea corectă a pachetelor, pe când UDP este mai rapid în transmiterea datagramelor, dar există posibilitatea ca o parte din acestea să nu ajungă la destinație.

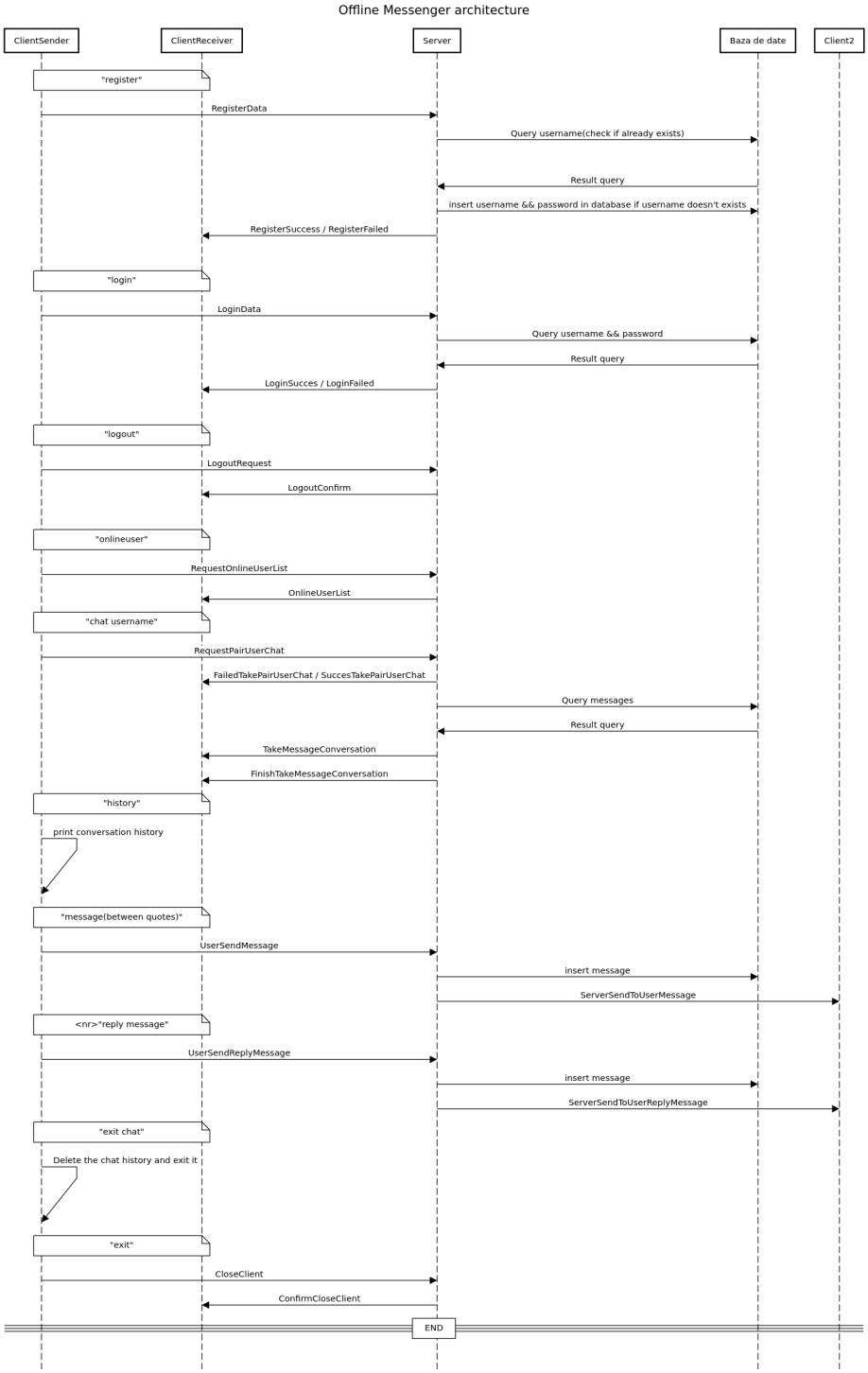
2.2 Baza de date

Utilizarea bazelor de date asigură persistența și organizarea eficientă a mesajelor și informațiilor utilizatorilor, permițând accesul facil la istoricul conversațiilor. S-a folosit ca bază de date SQLite, deoarece este simplu de integrat în codul C++, nu necesită un server de baze de date separat și previne condițiile de concurență și "race conditions" prin utilizarea mutex-urilor.

3 Structura aplicației

Serverul este concurent, asigurând comunicarea cu mai mulți clienți simultan.

Clientul este format din 2 thread-uri; thread-ul ClientSender trimite informațiile la server, iar thread-ul ClientReceiver primește informațiile de la server. Pentru a împiedica fenomenul de "race condition" în aplicația clientului am folosit mutex, care asigură accesul exclusiv la resursele partajate.
În diagrama de mai jos, este reprezentat modul în care lucrează aplicația.


Offline Messenger architecture
ClientSender
ClientReceiver
Server
Baza de date
Client2
"register"
RegisterData
Query username(check if already exists)
Result query
insert username && password in database if username doesn't exists
RegisterSuccess / RegisterFailed
"login"
LoginData
Query username && password
Result query
LoginSucces / LoginFailed
"logout"
LogoutRequest
LogoutConfirm
"onlineuser"
RequestOnlineUserList
OnlineUserList
"chat username"
RequestPairUserChat
FailedTakePairUserChat / SuccesTakePairUserChat
Query messages
Result query
TakeMessageConversation
FinishTakeMessageConversation
"history"
print conversation history
"message(between quotes)"
UserSendMessage
insert message
ServerSendToUserMessage
<nr>"reply message"
UserSendReplyMessage
insert message
ServerSendToUserReplyMessage
"exit chat"
Delete the chat history and exit it
"exit"
CloseClient
ConfirmCloseClient
END

4 Aspecte de implementare

4.1 Race Condition

Condițiile de race apar atunci când mai multe thread-uri accesează simultan aceleași date, cel puțin unul dintre accesări fiind o operație de scriere, ceea ce poate duce la rezultate imprevizibile și la coruperea datelor.

Pentru a preveni aceste probleme, am implementat mecanisme de sincronizare folosind std::mutex. Acestea asigură accesul exclusiv la resursele partajate, permițând doar unui singur thread să acceseze secțiunile critice de cod la un moment dat.

```
129  class Server {
130      vector<ClientInfo> clients;
131      vector<User> onlineUsers;
132      std::mutex clientsMutex;
133      std::mutex onlineUserMutex;
134  public:
135
136      void addClient(std::thread&& th, int socket) {
137          std::lock_guard<std::mutex> lock([&]clientsMutex);
138          clients.emplace_back(std::move(th), socket);
139      }
```

4.2 Clientul

Cum am precizat mai sus, clientul este format din 2 thread-uri (fire de execuție), ClientSender și ClientReceiver. Această abordare ne ajută ca clientul să primească instantaneu în chat (dacă acesta este deschis) mesajul de la celălalt utilizator. ClientSender-ul citește de la tastatură o comandă și testează dacă aceasta este validă și trimite task-ul respectiv serverului (în cazul comenzii help nu este necesar).

```
auto args:vector<string> = split_whitespace(buf);
if(args.size() ==0) {...}
if(args[0]=="exit") {...}
if(args[0]=="login") {...}
if(args[0]=="onlineuser") {...}
if(args[0]=="logout") {...}
if(args[0]=="chat") {...}
if(args[0]=="help") {...}
if(args[0]=="history") {...}
```

4.3 Server

Serverul primește un pachet de date de tipul Packet, care include atât tipul pachetului, cât și informația care trebuie procesată. Această structură permite serverului să identifice și să gestioneze corect conținutul transmis.

```
35  struct Packet{
36      TypeMsg typeMsg;
37      char buffer[MAXSIZEMSG]{};
38      explicit Packet(const string& str) {
39          (*this) = from_string(str);
40      }
41      explicit Packet(const TypeMsg typeMsg,const char * buf) {
42          this->typeMsg = typeMsg;
43          strcpy(this->buffer,buf);
44      }
45      Packet(): typeMsg(TypeMsg::Nothing),buffer{}{}
46      string to_string() const {
47          json j;
48          j["typeMsg"] = typeMsg;
49          j["buffer"] = buffer;
50          return j.dump();  // convertesc JSON-ul intr-un string
51      }
52      static Packet from_string(const string& str) {
53          json j = json::parse(str);
54          Packet msg;
55          msg.typeMsg = j["typeMsg"].get<TypeMsg>();
56          strcpy(msg.buffer, src:j["buffer"].get<string>().c_str());
57          return msg;
58      }
59  };
```

După preluarea pachetului și gestionarea acestuia, serverul la rândul său trimite un pachet răspuns către thread-ul ClientReceiver din client, care reprezintă soluția request-ului dat de ClientSender. Tipurile de pachete prin care se face comunicarea client/server se pot vedea mai jos:

```
enum TypeMsg:int{
    Nothing = -1,//by default
    //tipurile de pachete pe care le trasmite clientul serverului
    LoginData =0,
    RegisterData,
    LogoutRequest,
    RequestOnlineUserList,
    RequestPairUserChat,
    UserSendMessage,
    UserSendReplyMessage,
    CloseClient,
    //tipurile de pachete pe care le trasmite serverul clientului
    LoginSuccess,
    LoginFailed,
    RegisterSuccess,
    RegisterFailed,
    LogoutConfirm,
    OnlineUserList,
    SuccessTakePairUserChat,
    FailedTakePairUserChat,
    TakeMessageConversation,
    FinishTakeMessageConversation,
    ServerSendToUserMessage,
    ServerSendToUserReplyMessage,
    ConfirmCloseClient,
};
```

5 Concluzii

Aplicația oferă bazele necesare unui chat între utilizatorii conectați.

* Se poate implementa o interfață grafică pentru utilizator a aplicației pentru îmbunătățirea experienței utilizatorilor;

* Criptarea end-to-end pentru a asigura confidențialitatea mesajelor transmise între utilizatori, astfel doar expeditorul și destinatarul vor putea citi mesajele.

6 Bibliografie

- Computer Networks - Faculty of Computer Science
  https://edu.info.uaic.ro/computer-networks/index.php
- C++ Reference
  https://en.cppreference.com/w/
- SQLite Documentation
  https://www.sqlite.org/docs.html
- JSON for Modern C++
  https://github.com/nlohmann/json?tab=readme-ov-file