Raport Tehnic: Offline Messenger

Balan Gabriela

Facultatea de Informatica Iasi UAIC
gabry.balan05@gmail.com

Abstract. Dezvoltarea unei aplicații client/server care să permită schimbul de mesaje între utilizatori conectați și să ofere funcționalitatea trimiterii mesajelor și către utilizatorii offline, acestora din urmă apărându-le mesajele atunci când se vor conecta la server. De asemenea, utilizatorii vor avea posibilitatea de a trimite un răspuns (reply) în mod specific la anumite mesaje primite. Aplicația va oferi și istoricul conversațiilor pentru și cu fiecare utilizator în parte.

Keywords: TCP · Socket · Thread · Select · Mutex · SQLite.

1 Introducere

Viziunea generală:
Proiectul Offline Messenger se bazează pe dezvoltarea unei aplicații client/server care să permită schimbul de mesaje între utilizatorii care au făcut login cu username și parolă. Schimbul de mesaje se poate realiza cu un anumit utilizator, în urma inserării username-ului său. Pentru utilizatorii care nu sunt conectați, dar care sunt înregistrați, pot primi mesaje, iar acestea vor apărea în urma conectării utilizatorului la server. De asemenea, utilizatorii vor putea răspunde (reply) la un anumit mesaj primit necitit ,fiecare mesaj avand un identificator unic. Aplicația va afișa și istoricul conversațiilor, în urma cererii din partea utilizatorului, cu un user anume ales.
Obiectivele proiectului:
- asigurarea comunicării concurente între clienți în timp real;
- login cu username și parolă unică pentru fiecare utilizator, care va fi reținută într-un fișier;
- primirea mesajelor necitite;
- posibilitatea de a raspunde (reply) la mesaje necitite, acesta fiind identificat printr-un id unic;
- afisarea istoricului cu un anumit utilizator ;
- inserarea username-ului utilizatorului cu care se dorește a face schimbul de mesaje;
- stocarea mesajelor și a istoricului conversațiilor cu SQLite;

2 Tehnologii Aplicate

Protocol de Comunicare: TCP datorită faptului că acesta oferă:
- fiabilitate și ordinea datelor, asigurând că mesajele trimise sunt livrate corect

și în ordine la alt utilizator;
- gestionarea fluxului, pentru a evita supraîncărcarea rețelei și pierderea sau coruperea datelor;
- conexiune persistentă, fiind benefică pentru schimbul continuu de mesaje, iar conexiunea poate fi menținută deschisă pentru a gestiona mesajele trimise utilizatorilor offline.
Bază de Date: SQLite - folosit pentru stocarea mesajelor și istoricul conversațiilor.

3 Structura Aplicației

Conceptele de bază folosite în modelarea aplicației:
- Socket-uri: pentru a stabili conexiuni între server și clienți, acestea facilitează comunicația prin intermediul rețelei.;
- Thread-uri: fiecare client are un thread dedicat pe server, gestionat prin intermediul bibliotecii pthread ; aceste thread-uri asigură o abordare concurentă pentru gestionarea comunicării cu clienții;
- Select: se folosește pentru a monitoriza activitatea pe socket-uri; select permite serverului să aștepte și să reacționeze la evenimente de citire/scriere pe mai multe socket-uri, evitând astfel blocarea;
- Mutex: mecanism de sincronizare folosit pentru a evita condițiile de cursă în manipularea listei de clienți în server;
- Bază de date : voi integra o bază de date SQLite; acestă funcționalitate nu este implementată în codul actual.
Diagramă detaliată a aplicației : aici.

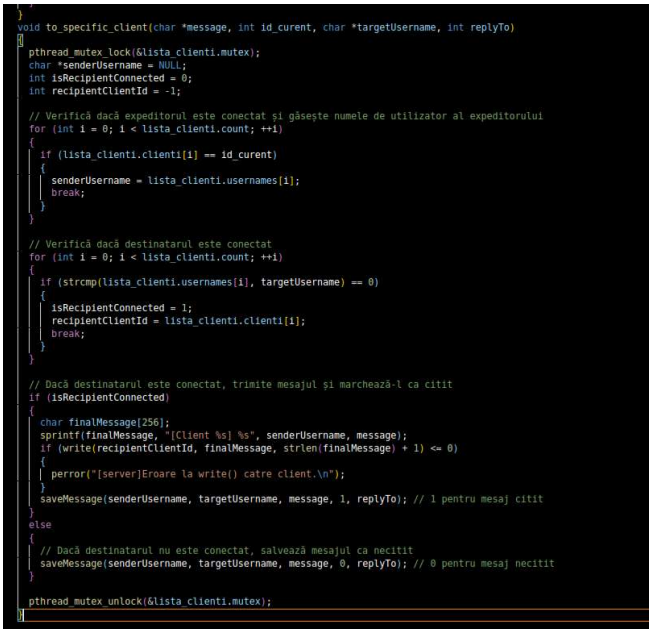
4 Aspecte de Implementare

Aplicatia se bazează pe trimiterea și primirea de mesaje între client și server. Mesajele sunt reprezentate sub formă de șiruri de caractere. Serverul ascultă pe un port specific și acceptă conexiuni de la clienți. După ce un client s-a conectat, se creeaza thread-uri dedicate pentru gestionarea comunicației client-server si server-client.
Scenariu real de utilizare: intr-un mediu profesional, echipa de proiect poate utiliza această aplicație pentru a comunica rapid și eficient între membrii echipei.
Secțiuni de cod specifice și inovative ale proiectului:

```
void *receiveMessage(void *arg)
{
  int clientSocket = *((int *)arg);
  char buffer[1024];

  while (1)
  { // Initializarea setului de descriptori pentru citire
    fd_set read_fds;
    FD_ZERO(&read_fds);
    FD_SET(clientSocket, &read_fds);//citire de la server

// Verificarea descriptorilor folosind select
    if (select(clientSocket + 1, &read_fds, NULL, NULL, NULL) == -1)
    {
      perror("Eroare la select().");
      break;
    }
// Verificăm dacă există date pe socket-ul clientului
    if (FD_ISSET(clientSocket, &read_fds))
    {
      ssize_t bytesRead;

      if ((bytesRead = read(clientSocket, buffer, sizeof(buffer))) <= 0)
      {
        perror("Eroare la read de la server.");
        break;
      }

      buffer[bytesRead] = '\0';
      printf("%s", buffer);
    }
  }

  pthread_exit(NULL);
}
```

Fig. 1: În client

Select este folosit aici pentru a face operațiile I/O non-blocante, astfel încât să pot trata mai mulți clienți simultan și să pot reacționa când datele sunt disponibile pentru citire fără a bloca întregul program.



```
}
void to_specific_client(char *message, int id_curent, char *targetUsername, int replyTo)
{
  pthread_mutex_lock(&lista_clienti.mutex);
  char *senderUsername = NULL;
  int isRecipientConnected = 0;
  int recipientClientId = -1;

  // Verifică dacă expeditorul este conectat și găsește numele de utilizator al expeditorului
  for (int i = 0; i < lista_clienti.count; ++i)
  {
    if (lista_clienti.clienti[i] == id_curent)
    {
      senderUsername = lista_clienti.usernames[i];
      break;
    }
  }

  // Verifică dacă destinatarul este conectat
  for (int i = 0; i < lista_clienti.count; ++i)
  {
    if (strcmp(lista_clienti.usernames[i], targetUsername) == 0)
    {
      isRecipientConnected = 1;
      recipientClientId = lista_clienti.clienti[i];
      break;
    }
  }

  // Dacă destinatarul este conectat, trimite mesajul și marchează-l ca citit
  if (isRecipientConnected)
  {
    char finalMessage[256];
    sprintf(finalMessage, "[Client %s] %s", senderUsername, message);
    if (write(recipientClientId, finalMessage, strlen(finalMessage) + 1) <= 0)
    {
      perror("[server]Eroare la write() catre client.\n");
    }
    saveMessage(senderUsername, targetUsername, message, 1, replyTo); // 1 pentru mesaj citit
  }
  else
  {
    // Dacă destinatarul nu este conectat, salvează mesajul ca necitit
    saveMessage(senderUsername, targetUsername, message, 0, replyTo); // 0 pentru mesaj necitit
  }

  pthread_mutex_unlock(&lista_clienti.mutex);
}
```

Fig. 2: În server

În server tot folosesc select astfel încât atunci când apare ceva de citit în socket, serverul citește un mesaj de la client și îl trimite catre utilizatorul ales`to_specific_client`.

5 Concluzii

Potențialele îmbunătățiri ale soluției propuse:
-să pot comunica și cu un grup de utilizatori aleși, nu doar cu un singur utilizator;

6 Referințe Bibliografice

Referințe la sursele de informații folosite în cadrul proiectului:
`https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php`
`https://profs.info.uaic.ro/~eonica/rc/`
`https://man7.org/linux/man-pages/`