Offline Messenger*

Sandu Smaranda

Facultatea de Informatică Iași, UAIC
smaranda.sandu2002@hotmail.com

Abstract. Necesitatea de transmitere rapidă de mesaje a oamenilor a dus la apariția diverselor platforme de socializare, cea mai populară formă de comunicare din zilele noastre fiind mesageria. Scopul acestui proiect este de a dezvolta o aplicație care să permită comunicarea între utilizatori conectați, păstrând, în același timp, istoricul conversațiilor acestora. Comparând concepte și tehnologii prezentate în timpul cursurilor și dezvoltate la laboratoare, le-am ales pe cele mai potrivite pentru acest proiect în particular.

Keywords: TCP · baze de date · threads · server · client.

1 Introducere

Acest raport va detalia planul de implementare a proiectului pentru nota finală la materia *Rețele de calculatoare*, proiectul ales fiind *Offline Messenger*.

2 Tehnologiile utilizate

2.1 TCP

În urma comparării avantajelor și dezavantajelor celor două, am ajuns la concluzia că în cadrul acestui proiect, folosirea TCP[1] este de preferat folosirii UDP[2].

Motivele sunt următoarele:

– Fiind de tipul *connection − oriented*, TCP va stabili o conexiune între transmițător și receptor, spre deosebire de UDP, care este *connectionless* și nu trebuie să creeze vreo conexiune între cele 2. Având în vedere că proiectul simulează "conversații" între utilizatori, această conexiune este necesară. TCP va verifica dacă cel care recepționează este capabil să primească un mesaj la momentul când acesta încearcă să fie transmis, spre deosebire de UDP, care va trimite fără a mai verifica, ceea ce poate cauza pierderi de informații.

* Proiect de tip B din lista regăsită pe siteul materiei
[1] Transmission control protocol
[2] User datagram protocol

- Conexiunea în cazul TCP este de încredere, fiind stabilită prin sistemul *three-way handshake* (clientul trimite la server un număr pe care vrea să îl folosească pentru verificarea conexiunii, serverul îl primește și îl trimite înapoi incrementat cu 1, alături de un număr propriu. Clientul primește cele 2 numere și îl trimite pe cel al serverului înapoi, incrementat cu 1. Astfel se stabilește conexiunea între cele două).
- TCP dispune de protocoale pentru evitarea aglomerării, luând în considerare capacitatea rețelei, stabilind viteza porivită de transmitere a datelor. UDP însă nu implementează astfel de protocoale, motiv pentru care în momentul congestionării rețelei, renunță la date care se află în așteptare, prioritizându-le pe celelalte. Există astfel posibilitatea de a pierde informații pe parcurs, ceea ce se dorește a fi evitat într-o aplicație de mesagerie, întrucât dorim ca toate mesajele să ajungă la destinatar.
- TCP are mai multe mecanisme de asigurare a integrității transmisiunii mesajelor, precum închiderea conexiunii după o anumită perioadă de inactivitate, folosirea unui câmp de checksum (care verifică integritatea după transmitere a informației) și trimiterea și primirea de mesaje ACK (de confirmare) la transmiterea de informație sau la stabilirea de conexiuni.
- În cazul TCP, datele sunt transmise cu ajutorul unui mecanism de secvenționare, fiecărui segment de date fiindu-i asociat un număr de ordine, astfel încât informația să fie transmisă în ordinea corespunzătoare, prevenindu-se coruperea datelor. UDP însă nu ia în calcul ordinea de transmitere a informației, aceasta fiind ulterior reconstruită la destinație în ordinea de sosire a pachetelor. Am putea primi astfel mesaje fără sens sau corupte.
- Prin specificațiile sale, prezentate mai sus, TCP asigură ajungerea mesajului corect și întreg la destinatar, spre deosebire de UDP care nu asigură acest lucru. Având în vedere specificul acestui proiect, putem sacrifica viteza mai mare de transmitere a datelor pe care o aduce UDP în favoarea integrității datelor garantată de TCP.

2.2 Threads

Cea de-a doua tehnologie utilizată în implementarea proiectului va fi *threads*.

Motivele sunt următoarele:

- Dacă un *thread* este în așteptare, celelalte pot rula în continuare, independente de acesta. Astfel, dacă avem un mesaj în așteptare, trimis către un client care este offline, alți clienți vor putea trimite în continuare mesaje, fără a aștepta conectarea acestuia.
- Spre deosebire de $fork$, *threadurile* folosesc mai puține resurse și sunt mai eficiente, inclusiv din punct de vedere al timpului.
- Din punct de vedere al memoriei, *threadurile* sunt mai eficente, împărțind același bloc de memorie, spre deosebire de $fork$, unde fiecare proces copil are propriul spațiu de memorie. Un avantaj al acestui spațiu de memorie comun este rapiditatea comunicării între procese.

– Un dezavantaj al *forkului* este apariția proceselor orfan în cazul în care părintele se termină înaintea copilului. *Threadurile* nu prezintă acest dezavantaj, întrucât dacă părintele se termină brusc, *threadul* se termină automat.

2.3 Baze de date

Se vor folosi *baze de date* în loc de *fișiere text*

Motivele sunt următoarele:

– Viteza de căutare în cazul *bazelor de date* este mult mai mare, întrucât există o metodă specifică pentru acest lucru.
– Introducerea de noi utilizatori este mai ușoară, folosim metoda specifică pentru adăugarea informațiilor și urmăm structura prestabilită a tabelelor.
– Modificarea datelor este mai ușoară, având o metoda specifică, pe când la fișierele text aceasta trebuie făcută manual. (ex de utilizare: când vrem să modificăm numele de utilizator).
– Datele sunt structurate mult mai bine, bazele de date fiind relaționale.
– Nu putem avea probleme sau confuzii cauzate de duplicate, întrucât bazele de date nu permit acest lucru (fiecare utilizator are un ID unic, care va reprezenta o cheie primară. Astfel, chiar dacă avem doi utilizatori cu același nume, aceștia vor putea fi diferențiați.).
– Logica structurală a bazelor de date permite evitarea problemelor cauzate de introducerea datelor:
  - *greșite* : într-un fișier text, de exemplu, putem introduce un număr în locul unui șir de caractere, pe când bazele de date nu ne permit acest lucru;
  - *incomplete* : într-un fișier text putem exclude, accidental, anumite informații necesare (ex: nume de utilizator). În cazul bazelor de date, putem marca anumite coloane ce conțin date esențiale ca fiind *required*, ceea ce ne va obliga să introducem acele detalii.

3 Arhitectura aplicației

Un utilizator T (client) se conectează și trimite un mesaj către alt utilizator R (client). Acest mesaj ajunge la server, care stochează mesajul în baza de date. Dacă destinataul este online și își cere mesajul, îl va primi de la server și se va actualiza în baza de date statusul mesajului (trimis). Apoi, R poate trimite către T mesaje in același fel și așa mai departe.

Datorită bazelor de date ce stochează toate mesajele trimise, vom avea acces la istoricul fiecărei conversații.

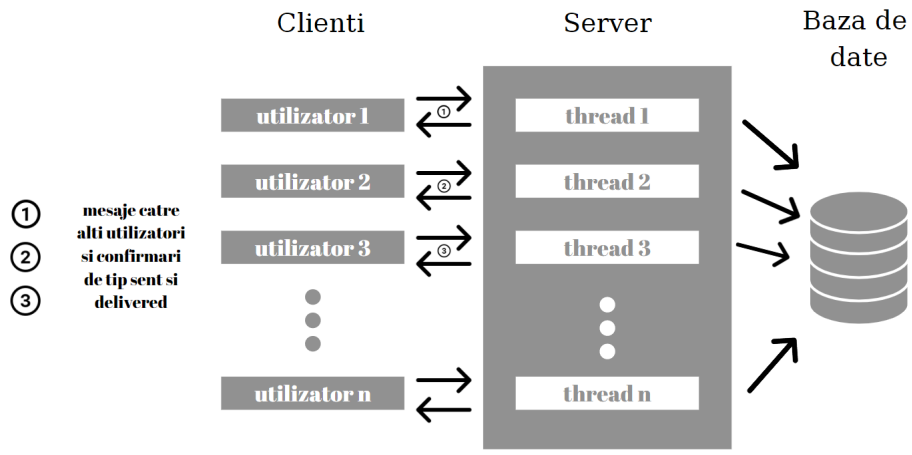

Fig. 1. Diagrama proiect

4 Detalii de implementare

Logare utilizator

```
int login(int sd, thData *ptd)
{

    char *nume = (char *)malloc(40 * sizeof(char));
    char *confirmare = (char *)malloc(100 * sizeof(char));
    int status = 1; // cand se logheaza clientul statusul lui devine 1
    sqlite3 *db = NULL;
    char *err_msg = 0;
    char exista[200];
    struct utiliz ex;
    ex.ID = -1;

    bzero(exista, sizeof(exista));
    bzero(verif, sizeof(verif));
    bzero(nume, sizeof(nume));
    bzero(confirmare, sizeof(confirmare));

    if (read(ptd->cl, nume, sizeof(nume)) <= 0)
    {
        printf("[Thread %d]\n", ptd->idThread);
        perror("Eroare la citirea de la client.\n");
    }
```

```
int rc = sqlite3_open("conturi.db", &db);

if (rc != SQLITE_OK)
{

    fprintf(stderr, "Cannot open database: %s\n", sqlite3_errmsg(db));
    sqlite3_close(db);

    return 1;
}
char verif[100];

sprintf(verif, "SELECT Id, Name FROM Users WHERE Name = '%s'", nume);

exista[0] = '\0';

rc = sqlite3_exec(db, verif, callback, &ex, &err_msg);

printf("%d", ex.ID);
printf("\n");
printf("%s", nume);
printf("\n");

if (ex.ID > -1)
{
    printf("Exista\n");
    ptd->idUtil = ex.ID;
}

else
    printf("Nu exista\n");

printf("\n");
if (rc != SQLITE_OK)
{

    fprintf(stderr, "SQL error: %s\n", err_msg);

    sqlite3_free(err_msg);
    sqlite3_close(db);

    return 1;
}
```

```
    char stmt[100];
    sprintf(stmt, "UPDATE Users SET Online = '%d'"
                  "WHERE Name = '%s'",
            status, nume);

    rc = sqlite3_exec(db, stmt, 0, 0, &err_msg);

    if (rc != SQLITE_OK)
    {

        fprintf(stderr, "SQL error: %s\n", err_msg);

        sqlite3_free(err_msg);
        sqlite3_close(db);

        return 1;
    }

    if (ex.ID > -1)
    {
        strcat(confirmare, "Logat");
    }

    else
        strcat(confirmare, "Nelogat");

    if (write(ptd->cl, confirmare, sizeof(confirmare)) <= 0)
    {
        printf("[Thread %d] ", ptd->idThread);
        perror("[Thread]Eroare la write() catre client.\n");
    }
    else
        printf("[Thread %d]Mesajul a fost trasmis cu succes.\n", ptd->idThread);

    if (db != NULL)
        sqlite3_close(db);

    return 0;
}
```

5 Concluzii

Astfel, cu ajutorul tehnologiilor ale caror beneficii au fost prezentate mai sus va putea fi implementat proiectul, conform arhitecturii descrise.

Bibliografie

1. Andrew S. Tanenbaum, Herbert Bos : Modern Operating Systems (2015 edition)
2. https://profs.info.uaic.ro/ computernetworks/ (2022-2023)
3. https://www.ibm.com/docs/en/zos/2.2.0?topic=chart-concurrent-iterative-servers
4. https://www.tutorialspoint.com/ - multi-threading
5. https://www.sqlite.org/c3ref/prepare.html
6. https://en.m.wikipedia.org/