Collaborative Notepad*

Niță L. Dennis-Alexandru

[1] Facultatea de Informatică, Gen. Henri Mathias Bethelot 16, 700483 - România
[2] dennis.nita@info.uaic.ro

Abstract. Documentație referitoare la proiectul ales pentru materia Computer Networks. Proiectul constă într-o aplicație client-server care permite utilizatorilor să prelucreze fișiere text concurent, în timp real.

Keywords: TCP · Server · MongoDB · Client · Mutex · Concurent · Thread

1 Introducere

Proiectul ales reprezintă o aplicație client-server care permite editarea simultană a fișierelor text. Serverul trebuie să implementeze creearea simultană a mai multor "camere" la care cel mult doi clienți se pot conecta și care pot prelucra fișiere text, simultan. Fiecare client are posibilitatea de a salva fișierul prelucreat anterior în server, în baza de date, și de a-l descărca ulterior.

2 Tehnologii Aplicate

Pentru acest proiect s-a folosit limbajul **C++**, standardul 2017, compilat cu **g++ 13.1.0** pe distro-ul **Ubuntu 23.0**. Motivația alegerii acestui limbaj a fost accesul la programarea orientată pe obiecte, drept și librăriile aferente lui. Ca și protocol de comunicație, este folosit Transmission Control Protocol (**TCP**) deoarece pentru acest gen de aplicație este nevoie de un stream constant și sigur de date, actualizarea **notepad-ului** fiind făcută în timp real.

Thread-ul main al procesului executat este serverul propriu-zis, acesta ascultând după conecțiuni noi de la clienți. În urma conectării unui client, se creează un thread noi specific acelui client care va asculta informațiile de la client, le va prelucra și va trimite înapoi la client un răspuns.

Comunicarea dintre thread-urile aferente server-ului, dar și cele asociate client-ului se face prin actualizarea variabelelor globale, fiecare variablă fiindu-i alocată câte un **mutex**. Pentru interfața grafică a clientului (serverul nu are interfață grafică) s-a folosit **SFML 2.6.1** deoarece este o librărie puternică, care satură nevoile grafice ale acestui proiect.

Pentru stocarea și manipularea eficientă a datelor se folosește baza de date non-relațională **MongoDB**, datele fiind salvate sub formă de documente **JSON**.

* Proiect pentru materia Computer Networks.

3 Structura aplicației

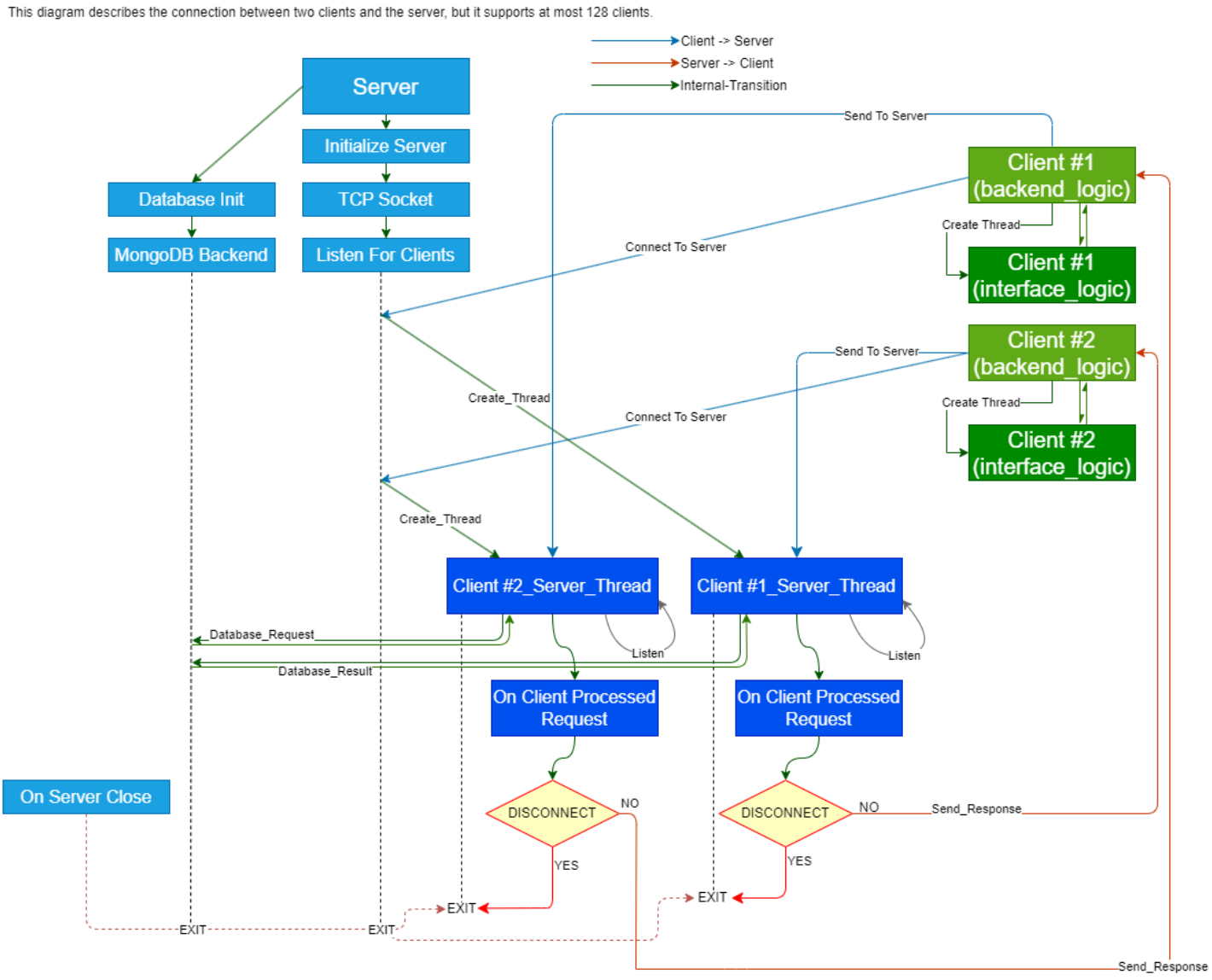


Fig. 1. Structura aplicației din punct al vedere al conecțiunii.

Diagrama UML

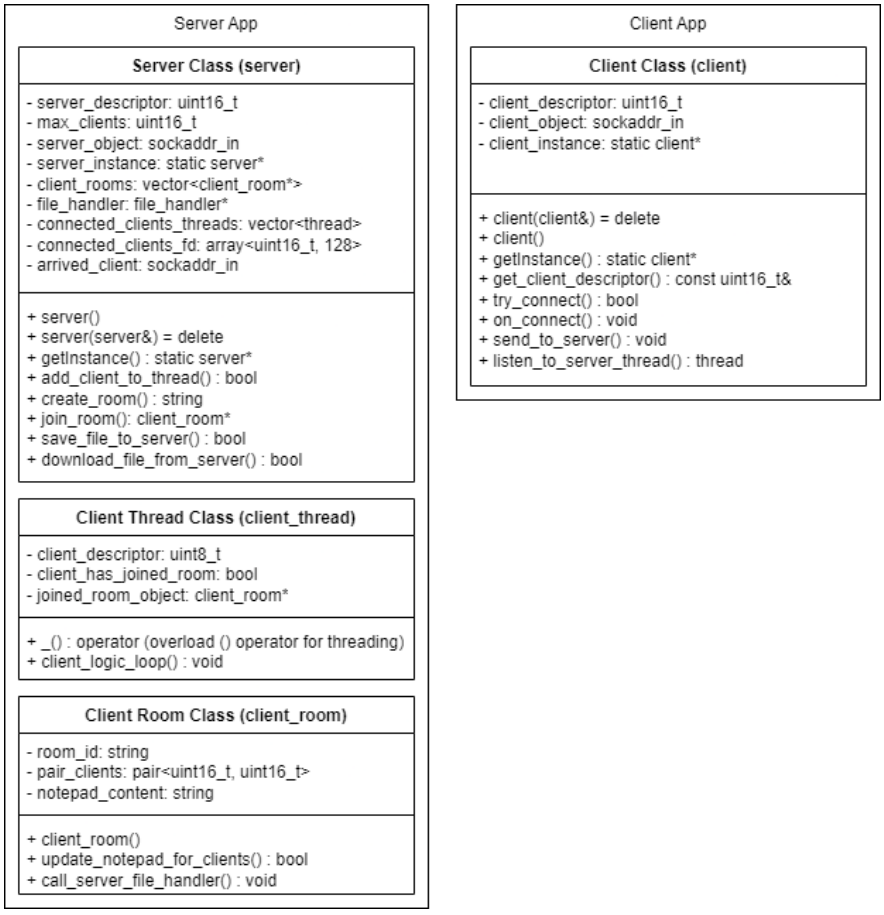


Fig. 2. Clasele obiectelor folosite în aplicație.

3.1 Elemente cheie

Serverul

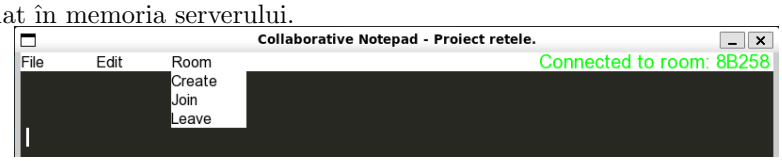
Serverul este componenta care face legătura între toți utilizatorii. Acesta folosește drept protocol, protocolul **TCP**, fiind necesară transmiterea datelor într-un flux continuu, sigur. Serverul are drept design pattern **singleton-ul**, asigurând existența unei singure instanțe per aplicație și totodată accesul global la aceasta. Instanța se află pe thread-ul main, aceasta ascultând după noi posibile conecțiuni (accept(2)). În urma unei noi conecțiuni, serverul instanțiază, într-un thread nou, detașat, un obiect de tipul (client_thread), instanță specifică clientului conectat.

Această instanță se ocupă în intregime de primirea mesajelor de clientul respectiv, prelucrarea informațiilor și trimiterea unui răspuns. Folosirea multithreading-ului saturează nevoia concurenței serverului.

Instanțele respective sunt șterge din memorie la deconectarea clienților.

Camera clienților

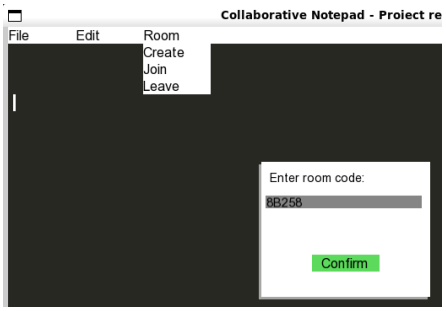
Camera clienților este componenta care permite celor mult doi clienți să prelucreze fișiere simultan. Un obiect de tipul (client_room) se ocupă cu transmitirea datelor doar între utilizatorii conectați la cameră, precum și conținul fișierului aflat în memoria serverului.



Această componentă are mai multe funcționalități care pot fi accesate prin poziționarea cursorului pe butonul **Room**.

Creare cameră. Un utilizator poate crea o cameră prin apăsarea butonului **Create**. Utilizatorul va fi conectat imediat la camera creată, cameră cu un ID alfanumeric de cinci caractere, generat aleatoriu. Interfața va specifica întotdeauna ID-ul camerei la care clientul este conectat.

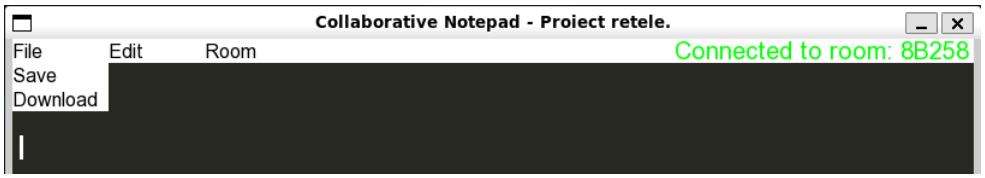
Alăturare cameră. Un utilizator poate să se alature unei camere prin apăsarea butonului **Join**.



Părăsește camera. Prin apăsarea butonului **Leave** utilizatorul poate părăsi camera.

Managerul de fișiere

Această componenta se ocupă cu manipularea fișierelor de către utilizatorii conectați la o cameră.



Funcționalitățile pot fi accesate prin poziționarea cursorului pe butonul **File**. De asemenea, utilizatorul poate manipula text-ul ca oricare alt notepad obișnuit prin apăsarea butonului **Edit**.

Salvează fișier. Un utilizator poate salva fișierul asociat camerei la care este conectat prin apăsarea butonului **Save**. Fișierul este salvat în baza de date, utilizatorilor primind astfel cod de identificare a fișierului, format din zece caractere alfanumerice.

Descarcă fișier. Un utilizator poate descărca fișierul asociat camerei la care este conectat sau un fișier salvat în baza de date. Utilizatorul poate descărca fișierul la care lucrează în momentul respectiv, sau oricare alt fișier prin introducerea unui cod de identificare format din zece caractere aflanumerice. Dacă în baza de date există un astfel de fișier, atunci utilizatorul îl poate descărca în orice loc din SO, oferindu-i-se această opțiune într-o căsuță pop-up.

Deschide fișier. Un utilizator poate deschide un fișier local prin apăsarea butonului **Open**, fișier pe care ulterior îl poate asocia unei camere.

Șterge fișier. Un utilizator poate șterge un fișier aflat în baza de date prin identificarea acestuia cu ID-ul unic.

Clientul

Clientul reprezintă aplicația prin care utilizatorul poate interacționa cu serverul, acesta putând să facă schimb de date neintrerupt cu acesta. Instanța clientului este formată din trei thread-uri cu funcționalități esențiale.

Thread-ul main este thread-ul care ascultă input-uri de la client, fie ele comenzi de la consolă sau cereri primite de la intefața grafică. În momentul unei astfel de cereri, thread-ul trimite către server comanda, serverul o prelucrează, ulterior clientul citește răspunsul primit.

Al doilea thread constă într-un **listener** care ascultă în permanență serverul, într-un mod neblocant, procesând orice mesaj primit de la server.

Al treilea thread reprezintă logica și funcționalitatea interfației grafice.

Clientul trimite către server o dată la câteva secunde un **heartbeat** pentru a se asigura că conecțiunea încă există.

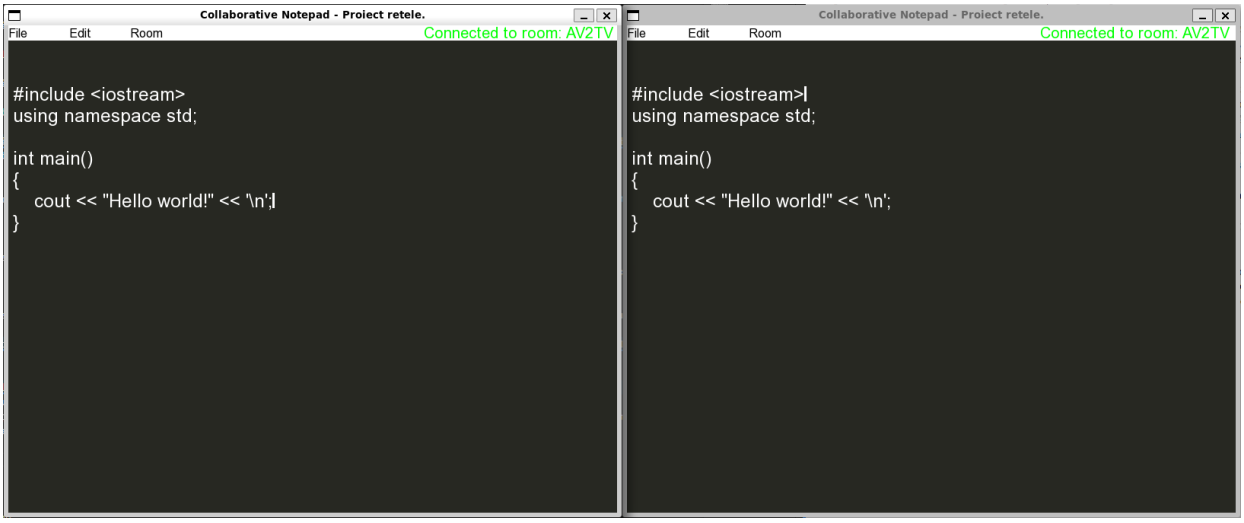


Fig. 3. Prototip al interfației grafice scris în SFML.

4 Aspecte de implementare

Pentru acest proiect partea de server-client a fost implementată cu librăriile **UNIX** (ex: sys/socket, netinet/ip.h) și drept **design-pattern**; serverul și clientul sunt **singleton**-uri pentru că orice instanță a aplicației client sau a aplicație server au, implicit, un singur client sau un singur server.

Mai jos este un snippet de cod care reprezintă inițializarea logicii thread-ului principal al serverului.

server.cpp

```
#define undefined -1

...

int main()
{
    signal_handler_logic();
    server* server_object = server::instance(AF_INET, SOCK_STREAM, 0,
        INADDR_ANY, 25565, 32);

    uint8_t connected_client_descriptor = undefined;

    std::vector<std::string> server_processing_thread_parameters;

    while(true)
    {
        socklen_t _l = sizeof(server_object->arrived_client);
        if((connected_client_descriptor =
            accept(server_object->get_server_descriptor(),
            reinterpret_cast<sockaddr*>(&server_object->arrived_client),
            &arrived_client_length)) == -1)
        {
            handle_error("Couldn't accept connection");
        }
        printf("Client with ID: %d has connected.\n",
            connected_client_descriptor);
        std::vector<std::string> thread_parameters;
        thread_parameters.push_back(std::to_string(connected_client_descriptor));
        std::thread connected_client_thread(client_thread(),
            thread_parameters);
        server_object->connected_clients_fds[++server_object->connected_clients_count]
            = connected_client_descriptor;
        connected_client_thread.detach();
    }
printf("OK\n");
}
```

5 Concluzii

În concluzie, Collaborative-Notepad este un proiect complex, interesant cu uzabilitate în viața reală. Spre exemplu, doi indivizi pasionați pot programa concomitent, acest lucru accelerând timpul de dezvoltare al unei aplicații. Un alt exemplu ar fi reprezentat de dorința a doi prieteni de a compune o carte împreună, putând astfel să o creeze concomitent.

Este mult loc de îmbunătățiri și de adăugare a unor caracteristi noi și inovative aplicației. Spre exemplu, aplicația va putea permite, în viitor, conectarea a mai multor clienți în aceeași cameră; sau distribuirea locației curente a cursorului fiecărui client.

Referințe bibliografice

1. Pagina cursului de rețele și calculatoare, UAIC - Facultatea de Informatică https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
2. Pagina profesorului îndrumător, profesorul de la seminar. https://profs.info.uaic.ro/~georgiana.calancea/laboratories.html
3. Listă cu porturile TCP https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers
4. Informații despre std::mutex https://en.cppreference.com/w/cpp/thread/mutex
5. Informații legate de std::thread, precum și alte mici noțiuni. https://cplusplus.com/reference/thread/thread/
6. Informații despre formatul LNCS http://www.springer.com/lncs.