Tema 2 - Retele de calculatoare

Chichirau Razvan

Facultatea de informatica,
Universitatea Alexandru Ioan Cuza Iasi
razvan.chichirau@info.uaic.ro

Abstract. Aceasta aplicatie a fost bazata pe cerintele proiectului "Public Mobility Advisor" din cadrul materiei de studiu "Retele de calculatoare" din anul 2 de la Facultatea de Informatica Iasi.

Keywords: *C++ · TCP/IP · fork() processes · sockets · pipes*

1 Introducere

1.1 Descriere

Proiectul pe care il voi prezenta in cadrul materiei *Retele de calculatoare* este numit *Public Mobility Advisor*, de categoria *B*. Acest proiect necesita implementarea unui server ce va asculta simultan la 2 port-uri diferite, unul fiind rezervat strict pentru client, iar celalalt fiind ocupat de transferul datelor cu privire la tramvaiele si autobuzele din oras. Informatiile despre mijloacele de transport vor fi simulate cu exactitate astfel incat folosirea aplicatiei sa corespunda realitatii.

1.2 Motivatie

Am ales acest proiect deoarece l-am considerat a fi unul practic, reusind cu usurinta sa fac o paralela intre realitate si aplicatie. Mi-a placut modul in care a fost gandita arhitectura acestui proiect in sensul ca imita cu exactitate procesul prin care un client adevarat ar primi informatii referitoare la mijloacele de transport din orasul sau prin intermediul unei aplicatii.

2 Tehnologii utilizate

2.1 Limbajul de programare folosit

Conform *www.techopedia.com*, C++ (numit C plus plus) este un limbaj orientat pe obiecte creat de informaticianul Bjorne Stroustrop ca parte a evoluției familiei de limbaje C. A fost dezvoltat ca o îmbunătățire cross-platform a limbajului C pentru a oferi dezvoltatorilor un grad mai ridicat de control asupra memoriei și a resurselor sistemului. Astăzi, C++ este încă foarte apreciat pentru portabilitatea sa notabilă care permite dezvoltatorilor să creeze programe

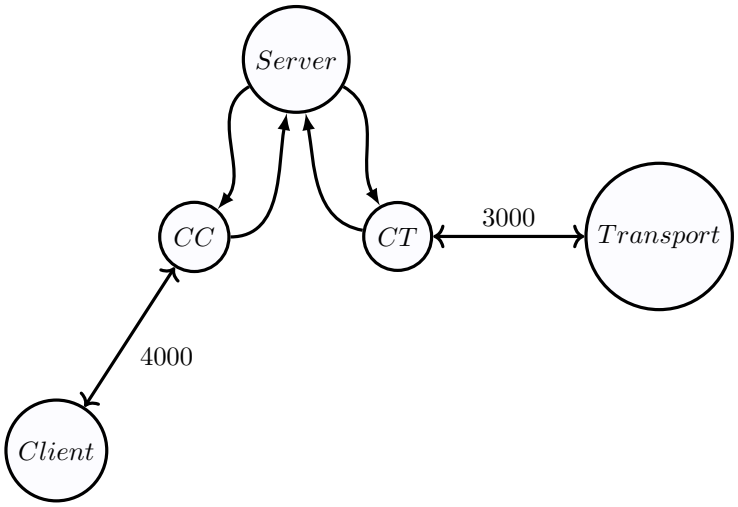
care pot rula pe diferite sisteme de operare sau platforme foarte ușor. În ciuda faptului că este un limbaj de nivel înalt, deoarece C++ este încă aproape de C, poate fi folosit pentru manipulare la nivel scăzut datorită relației sale strânse cu limbajul mașină.
Am ales sa proiectez aplicatia in acest limbaj de programare deoarece am considerat faptul ca ar fi mai usor organizarea tuturor obiectelor precum autobuzele, tramvaiele, harta orasului sub forma unor clase. Cu ajutorul acestora, am putut retine starile mijloacelor de transport la anumite momente de timp, precum si listele de adiacente corespunzatoare nodurilor hartii pe care se vor deplasa acestea. De asemenea, limbajul $C++$ este apropiat de programarea low-level astfel incat se pot face cu usurinta optimizari. In plus, a fost nevoie de implementarea unei clase pe care am denumit-o CTP care contine cate un pointer catre toate mijloacele de transport disponibile, astfel realizandu-se o abstractizare a datelor prin intermediul acestei interfete, dar si o clasa pe nume $Transport$, aceasta fiind la randul ei o interfata pentru mijloacele de transport propriu-zise.

2.2 Protocoale

In cadrul acestui proiect am decis sa utilizez $IPV4$ (prin precizarea constantei AF_INET la crearea socket-ului) ca protocol de internet, acesta fiind o alternativa relativ simpla de implementat. Mai exact, Internet Protocol versiunea IV (IPv4) este a patra versiune a Internet Protocol (IP). Este unul dintre protocoalele de bază ale metodelor de interconectare bazate pe standarde în internet și în alte rețele cu comutare de pachete. De asemenea, am optat la folosirea unui $Transmission\ Control\ Protocol$ sau TCP deoarece am prioritizat transmiterea in siguranta a datelor precum si garantia faptului ca datele primite ajung in aceeasi ordine in care au fost trimise. Acest lucru s-a demonstrat benefic mai ales in cadrul transmiterii datelor de la mijloacele de transport la server deoarece orice lipsa de informatii cu legatura la acestea se poate dovedi a fi generatoare de erori. In acest fel, fiabilitatea datelor este un aspect mult mai important in comparatie cu timpul petrecut in trimiterea si retrimiterea acestora. De asemenea, prin folosirea tipului $SOCK\_STREAM$ asiguram fluxuri de octeți secvențiate bidirectionale cu un mecanism de transmisie pentru fluxuri de date.

Asa cum relateaza site-ul *www.cloudflare.com*, relația TCP/IP este similară cu trimiterea unui mesaj scris pe un puzzle prin poștă. Mesajul este scris și puzzle-ul este rupt în bucăți. Fiecare piesă poate călători apoi prin rute postale diferite, dintre care unele durează mai mult decât altele. Când piesele puzzle-ului sosesc după ce și-au parcurs diferitele căi, piesele pot fi dezordonate. Protocolul Internet asigură că piesele ajung la adresa lor de destinație. Protocolul TCP poate fi considerat ca asamblatorul de puzzle de pe cealaltă parte, care pune piesele împreună în ordinea corectă, solicită retrimiterea pieselor lipsă și informează expeditorul că puzzle-ul a fost primit. TCP menține conexiunea cu expeditorul dinainte de trimiterea primei piese de puzzle până la trimiterea piesei finale.

3 Arhitectura aplicatiei



Serverul principal isi va creea exact 2 copii care vor avea atributii diferite.
1. $CC = Child\ Client$. Acesta se va ocupa de inregistrarea cerintelor clientului, transmiterea acestora catre server, precum si furnizarea raspunsului final catre client.
2. $CT = Child\ Transport$, care se va ocupa de preluarea si stocarea informatiilor transmise de $Transport$, procesarea comenzii transmise de catre $Server$, precum si trimiterea raspunsului catre parinte.

De asemenea, $Transport$ va simula in $real - time$ detalii despre autobuze si tramvaie precum viteza, ruta, statia curenta, numarul de identificare, posibilitatea oferirii asistentei pentru persoanele cu dizabilitati, capacitatea, numarul actual de persoane, orientarea, etc, si le va trimite o data la 3 secunde catre CT. La fiecare iteratie, acestea se vor actualiza in sensul ca algoritmul va decide in mod aleatoriu daca respectivul mijloc de transport a ajuns in urmatoarea statie precum si cate persoane au coborat si urcat.

Totodata, *serverul* va juca rol de intermediar intre cei 2 copii, intrucat acesta realizeaza singura conexiune intre procesele-fiu.

Dintr-un punct de vedere tehnic al implementarii, toata simularea se bazeaza pe comunicarea intre procese cu ajutorul unui socket conectat la port-ul 4000 intre *client* si CC, unui socket conectat la port-ul 3000 intre *transport* si CT + 4 pipe-uri ce asigura fluxul de octeti intre *server* si cei 2 copii.

Pentru evitarea caracterului blocant al apelului functiei *read*() se va folosi functia *select*() ce va monitoriza diferiti *file descriptori*, in functie de proces. Astfel:
- *transport* va monitoriza doar file descriptorul de citire aferent lui CT
- CT va monitoriza 2 file descriptori de citire: *transport* si *server*
- *server* va monitoriza 2 file descriptori: CC si CT
- CC va monitoriza 2 file descriptori: *client* si *server*

- *client* nu va monitoriza nici un file descriptor, intrucat acesta doar trimite o comanda si asteapta in mod blocant raspunsul prin apelarea functiei *read*().

4 Detalii de implementare

4.1 Crearea socket-ului

```
if ((sdClient = socket(AF_INET, SOCK_STREAM, 6)) == -1)
{
    printf("Eroare la creare socket.\n");
    exit(2);
}
int aux = 1;
if (setsockopt(sdClient, SOL_SOCKET, SO_REUSEADDR, &aux, sizeof(aux)) == -1)
    printf("Eroare la setsockopt()\n");

childClient.sin_family = AF_INET;
childClient.sin_addr.s_addr = htonl(INADDR_ANY);
childClient.sin_port = htons(4000); // PORT 4000

if (bind(sdClient, (struct sockaddr *)&childClient, sizeof(struct sockaddr)) == -1)
{
    perror("Eroare la bind: \n");
    exit(3);
}

if (listen(sdClient, 5) == -1)
{
    printf("Eroare la listen\n");
    exit(4);
}
```

Acest prim exemplu de implementare este preluat din codul scris in CC si reprezinta crearea liniei de comunicatie cu clientul folosind un socket. In prima parte se poate observa folosirea protocolului $IPV4$ prin constanta AF_INET, dar si folosirea unui TCP prin constanta numerica 6. Adaugarea functiei *setsockopt*() este optionala, aici fiind folosita pentru a-i comunica kernel-ului refolosirea portului daca acesta este ocupat. Dupa populatarea structurii *childClient* de tip *sockaddr_in*, aceasta se leaga de file-descriptorul *sdClient* ca apoi sa anuntam socket-ul ca poate accepta clienti prin *listen*().

4.2 Serializarea

```
void serialization(CTP vehicles, char *stringSerial)
{
    string text;
    for (int i = 0; i < vehicles.getVehicles().size(); i++)
    {
        for (int j = 0; j < vehicles.getVehicles().at(i)->getRoute()->size(); j++)
        {
            text += std::to_string(vehicles.getVehicles()[i]->getRoute()->at(j)); // ruta
            text += ',';
        }
        text += '|';
        text += vehicles.getVehicles()[i]->getType(); //Type
        text += '|';
        text += vehicles.getVehicles()[i]->getOrientation(); // Orientarea
        text += '|';
        text += std::to_string(vehicles.getVehicles()[i]->getDisabilities()); // disabilitati
        text += '|';
        text += std::to_string(vehicles.getVehicles()[i]->getBack()); // merge inapoi? 0/1
        text += '|';
        text += std::to_string(*vehicles.getVehicles()[i]->getZoneCount()); // zoneCount
        text += '|';
        text += std::to_string(vehicles.getVehicles()[i]->getID()); //ID
        text += '|';
        text += std::to_string(*vehicles.getVehicles()[i]->getSpeed()); // speed
        text += '|';
        text += std::to_string(*vehicles.getVehicles()[i]->getcurrentHold()); // currentHold
        text += '|';
        text += std::to_string(vehicles.getVehicles()[i]->getCapacity()); //Capacity
        text += ']';
    }
    strcpy(stringSerial, text.c_str());
}
```

Un alt exemplu relevant pentru acest proiect ar fi serializarea informatiilor din procesul *transport* pentru a putea fi transmise prin socket catre CT. Aceasta metoda este una favorabila din pricina timpului alocat executiei si a fiabilitatii. Avand in vedere ca prin socket nu se pot transmite pointeri la structuri intrucat acestia sunt locali masinii iar trimiterea unei intregi structuri este deseori evitata din pricina dificultatii de implementare, serializarea reprezinta o modalitate prin care putem codifica si salva cu usurinta starea curenta a mijloacelor de transport. Astfel, cu ajutorul unui $char*$, informatiile pot fi transmise printr-o simpla apelare a functiei $write()$. In cazul meu, dupa fiecare camp din structura va aparea caracterul | pentru a putea deosebi trecerea de la un camp la altul, precum si] pentru sfarsitul datelor despre un mijloc de transport.

Deserializarea este realizata pe "cealalta parte" a socket-ului. Se va citi acest sir de octeti intr-un $char[]$ iar apoi se va decodifica mesajul, populand o structura ce va avea aceleasi atribute.

4.3 Functia *select*()

```
while (1)
{
    tvClient.tv_sec = 2;
    tvClient.tv_usec = 0;
    FD_ZERO(&readfds);
    if (clientDescriptor == 0)
        add_set(sdClient, &readfds, &nfds);
    else
    {
        add_set(clientDescriptor, &readfds, &nfds);
        add_set(ServerToCC[0], &readfds, &nfds);
    }
    if (select(nfds + 1, &readfds, NULL, NULL, &tvClient) < 0)
        perror("select");

    if (FD_ISSET(sdClient, &readfds))
    {
        unsigned int len = sizeof(client);
        clientDescriptor = accept(sdClient, (struct sockaddr *)&client, &len);
        if (clientDescriptor < 0)
        {
            perror("[server] Eroare la accept().\n");
        }
        add_set(clientDescriptor, &readfds, &nfds);
    }
    if (FD_ISSET(clientDescriptor, &readfds))
    {
        char aux[100];
        read(clientDescriptor, &aux, sizeof(aux));
        printf("CC : Am citit mesajul de la client %s\n", aux);
        write(CCtoServer[1], &aux, sizeof(aux));
    }
    if (FD_ISSET(ServerToCC[0], &readfds))
    {
        char aux[100];
        read(ServerToCC[0], &aux, sizeof(aux));
        if (aux[0] != 'N')
            aux[2] = '\0';
        write(clientDescriptor, &aux, sizeof(aux));
        clientDescriptor = 0;
    }
}
```

Inca o componenta principala a acestui proiect o reprezinta folosirea functiei *select*() ce imi permite sa ascult mai multi file-descriptori pana cand unul devine *ready*. Am ales folosirea acestei functii deoarece am avut nevoie ca fiecare apel de *read*() sa nu fie blocant $\Rightarrow$ citim din socket/pipe doar atunci cand sunt octeti disponibili.

Exemplul de mai sus este luat din codul scris in interiorul lui *CC*. Acesta verifica daca sunt octeti disponibili pentru citire din socket-ul catre client si pipe-

ul catre server. Atunci cand unul dintre ei este disponibil, executa comenzile corespunzatoare de citire/scriere.

De asemenea, am incercat pe tot parcursul acestui proiect sa pastrez un caracter mai general al obiectelor. Astfel, urmatoarele aspecte sunt generate automat de catre algoritm: harta, traseele tuturor autobuzelor/tramvaielor, calibrarea hartii, a punctelor si a liniilor din grafica 2D. Programul trebuie doar sa stie numarul de noduri din graful ce va reprezenta harta (acesta trebuie sa fie patrat perfect).

Pentru implementarea unei interfete grafice 2D am folosit SFML prin care am reusit sa dinamizez ruta mijloacelor de transport. Atunci cand un client va solicita o informatie de la server, statia in care se afla clientul se va colora in verde pentru a observa mai usor functionalitatea algoritmului.

5 Concluzii

Solutia de rezolvare a problemei ar putea fi imbunatatita prin folosirea protocolului UDP in loc de TCP care favorizeaza timpul de transmitere a datelor in defavoarea fiabilitatii acestora. Pe langa posibilitatea de imbunatatire, exista si o probabilitate mare de aparitie a erorilor pe parcursul executarii programului deoarece unele datagrame se pot pierde $\Rightarrow$ informatiile pot fi eronate sau pot chiar lipsi.

De asemenea, pentru eficientizarea transmiterii datelor prin pipe-uri, putem configura serverul astfel incat acesta nici sa nu faca parte din procesarea datelor de la client. In aceasta situatie, ar fi nevoie de 2 pipe-uri in loc de 4 ce vor fi legate intre cei 2 fii.

6 Bibliografie

1. https://en.wikipedia.org/wiki/IPv4
2. https://www.cloudflare.com/en-gb/learning/ddos/glossary/tcp-ip/
3. https://www.techopedia.com/definition/26184/c-plus-plus-programming-language