Monitorizarea traficului - documentatie

Ursaciuc Adrian

Universitatea "Alexandru Ioan Cuza" din Iasi

1 Introducere

Numele proiectului este Monitorizarea traficului, din categoria A. Proiectul mi s-a parut unul foarte bine explicat si mult mai practic, de aceea l-am ales.

Aplicatia presupune monitorizarea evenimentelor din trafic si notificarea participantilor despre anumite evenimente. La server se vor conecta mai multi clienti ce vor trimite viteza cu care astia se deplaseaza cu o frecventa de un minut. In cazul in care un numar de masini circula cu o viteza sub limita pe un anumit drum, atunci este posibil sa fie un blocaj pe acel drum, iar celalalte masini vor fi avertizate despre acest lucru. Clientii vor putea sa raporteze accidente catre server, iar acesta va notifica restul participantilor(posibil sa notifice doar participantii din zona?). De asemenea, clientii se pot abona la anumite servicii (informatii despre vreme, evenimente sportive, preturi despre combustibil) si vor primi in mod regulat informatii legate de serviciile la care s-au abonat.

2 Tehnologiile utilizate

Dezvoltarea aplicatiei este realiazta in C++ pentru a putea crea o arhitectura decenta folosind clase.

Pentru comunicarea dintre client si server am decis sa folosesc protocolul TCP deoarece se ocupa de unele lucruri in mod implicit, in comparate cu UDP. In cazul in care avem de trimis mai multe pachete nu trebuie sa avem noi grija sa le ordonam dupa ce le primim, iar daca dimensiunea pachetului este foarte mare protocolul TCP se ocupa de impartirea lui in pachete mai mici, trimiterea lui si reasamblarea lui in ordinea in care a fost trimis.

Citirea a fost realizata cu ajutorul functiei select, atat pe server, cat si pe client. Serverul incearca sa citeasca in mod consant de la fiecare client, iar clientul incearca sa citeasca de la tastatura si de la server.

read() este folosit de catre server pentru a citi mesajele de client, iar clientul foloseste functia pentru a citi un raspuns de la server sau pentru a citi o comanda data de catre utilizator

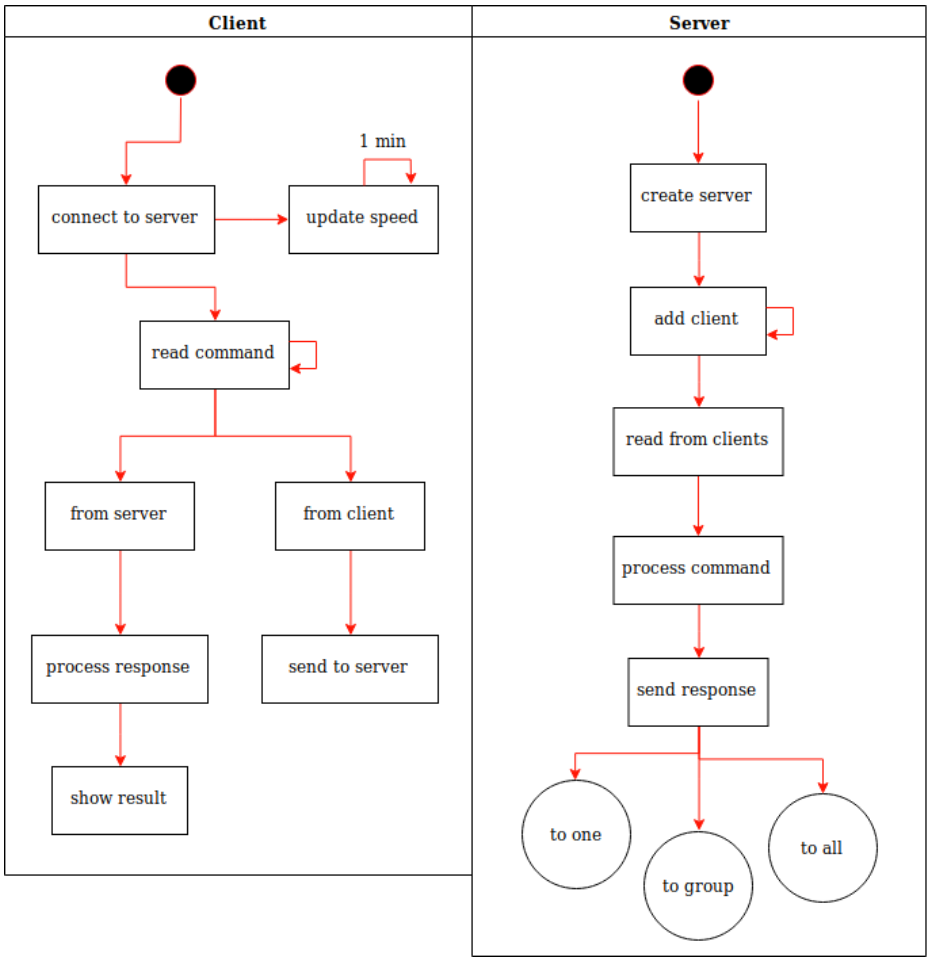
write() este o functie ce este folosita de catre ambele proiecte, este folostia pentru a trimite un mesaj la cealalta componenta (client/server).

O posibila varianta ar fi sa stochez datele intr-o baza de data(relationala sau nerelationala) pentru a nu pierde datele cand un utilizator se deconecteaza.

3 Arhitectura aplicatiei

Arhitectura aplicatiei poate fi vizualizata intrand pe link-ul urmator: Diagrama Arhitectura.

Arhitectura este compusa din doua mari componente, clientul si serverul. Clientul este respinsabil pentru citirea comenzilor de la server si afisarea lor, si trimiterea comenzilor la server. Scopul server-ului este de a citi comenzi de la server, sa le proceseze si sa produca un raspuns pentru un client, un grup de clienti sau pentru toti clientii.



4 Detalii de implementare

Datorita arhitecturii codul este unul foarte lizibl si usor de inteles. S-a incercat cat mai mult modularizarea componentelor deoarece modificarile si intretinerea codului va fi mult mai usoara.

A fost introdus si un protocol pentru scriere si citire. Prima data trimitem dimensiunea textului ce urmeaza sa fie citit, iar dupa aceasta trimitem textul in sine.

Write

```
static int Write(int socket, std::string &str)
{
    NormalizeString(str);

    unsigned long length = str.length();
    if(write(socket, &length, sizeof(length)) == -1)
    {
        return -1;
    }

    if(write(socket, str.c_str(), length) == -1)
    {
        return -1;
    }

    return 1;
}
```

Read

```
static int Read(int socket, std::string &str)
{
    unsigned long length;
    int bytes;

    if(socket == 0)
    {
        length = 1024;
    }
    else
    {
        if((bytes = read(socket, &length, sizeof(length)))
            == -1)
        {
            return -1;
        }

        if(bytes == 0)
        {
            return 0;
```

```
        }
    }

    char buf[length + 1];
    bzero(buf, length + 1);

    if((bytes = read(socket, &buf, length)) == -1)
    {
        return -1;
    }

    if(bytes == 0)
    {
        return 0;
    }

    str.assign(buf);
    NormalizeString(str);

    return 1;
}
```

La read pe socketul 0 (de la tastatura) trebuie sa avem grija deoarece nu putem citi prima data dimensiunea si dupa textul, trebuie sa citim direct textul.

Clientul si serverul folosesc aceleasi functii de Read si de Write pentru a asigura uniformitate.

In cazul in care un client se deconecteaza de le server din motive neasteptate, serverul o sa stie acest lucru si o sa elimine clientul din lista de clienti.

5 Concluzii

In concluzie, mai este mult munca de facut ca aplicatia sa ajunga la un final. Planuiesc sa refolosesc o mare parte din tema 1 pentru procesarea comenzilor pe server si sa implementez o modalitate de salvare a datelor in caz ca se intampla ceva cu clientul sau cu serverul.

6 Bibliografie

Computer Networks - Faculty of Computer Science
Computer Networks - Dr. Andrei Panu
Lecture Notes in Computer Science (LNCS)
Broken packets: IP fragmentation is flawed