Mersul trenurilor

Ojoc Georgiana Diana

12 Decembrie 2019

Abstract

Acest raport tehnic îşi propune să prezinte o implementare a unui sistem de gestionare a mersului trenurilor, folosind modelul client/server, în limbajul C++ standard, sub Linux.
Cuvinte-cheie: socket, thread, command queue, command design pattern

1 Introducere

Ideea proiectului porneşte de la programul IRIS, care a fost implementat de CFR şi care este disponibil încă din anul 2016. Această aplicaţie oferă călătorilor înregistraţi şi autentificaţi informaţii în timp real despre trenurile aflate în circulaţie în ziua respectivă sau după o anumită oră. Acest lucru este posibil prin actualizarea bazei de date care gestionează cursele zilnice cu informaţii furnizate de către utilizatorii autentificaţi în sistem cu drepturi de administrator. Aşadar, consumatorii vor afla la ce oră pleacă şi la ce oră ajunge fiecare tren identificat de un număr unic, astfel încât aceste informaţii respectă orarul general sau sunt estimări îmbunătăţite de numărul de minute de întârziere furnizate de pe traseu.

2 Tehnologii utilizate

Pentru implementarea paradigmei de proiectare în reţea client/server se utilizează modelul TCP concurent, care asigură procesarea mai multor cereri în acelaşi timp. Acest protocol orientat conexiune garantează trimiterea fluxurilor de octeţi în ordine şi fără erori, de la o maşină la alta. De asemenea, TCP tratează şi controlul fluxului de date pentru a se asigura că un emițător rapid nu aglomerează un receptor lent cu mai multe mesaje decât poate acesta să prelucreze. Serverul creează câte un fir de execuţie pentru fiecare client acceptat, deoarece această abordare care îi permite să servească mai mulţi clienţi simultan este mai rapidă decât multiprocesarea.

3 Arhitectura aplicaţiei

Conceptele implicate

Iniţial, serverul preia informaţiile generale despre trenuri şi orarul lor de funcţionare din documente JSON, cu ajutorul aplicației MySQL Workbench, şi le introduce în baza de date MySQL creată pentru uşurinţa în accesarea şi actualizarea datelor.

Baza de date se numește *train_schedule* și conţine tabelul *users*, cu informațiile clienților, adică numele de utilizator și parola, tabelul *departures*, cu informații despre plecările trenurilor, adică numărul trenului, staţia de plecare, dacă este sau nu prima stație, numărul liniei de la care pleacă, ora plecării, câte minute întârzie și ora estimată de plecare, și tabelul *arrivals*, cu informații despre sosirile trenurilor, adică numărul trenului, stația de sosire, dacă este sau nu ultima stație, numărul liniei la care ajunge, ora sosirii, câte minute întărzie și ora estimată de sosire. În primul tabel, cheia primară este numele de utilizator, iar în următoarele două tabele, cheia primară este reprezentată de indexul liniei, care se auto-incrementează.

În funcţie de privilegii şi cerinte, există două tipuri de clienţi, şi anume utilizatori obişnuiţi, care pot cere informaţii despre toate trenurile care circulă în ziua respectivă sau doar despre plecările şi sosirile care se recepţionează începând cu o anumită oră, sau administratori, care pot solicita şi actualizarea programului de circulaţie al trenurilor. Utilizatorii obişnuiţi se înregistrează şi autentifică folosind numele şi o parolă dorită, iar administratorii îşi confirmă identitatea folosind contul *administrator* şi parola *?Kv8H*.

Fiecare comandă este manipulată de câte o clasă derivată din clasa abstractă Command, care conține ca date indexul firului de execuție, descriptorul clientului şi argumentele, iar ca metodă funcţia *execute*, în care este implementată pentru fiecare tip de comandă metoda de răspuns corespunzătoare. În funcţie de tipul comenzii, serverul va interoga corespunzător tabelele *departures* și *arrivals*, adică va selecta rândurile necesare sau va actualiza câmpurile cu numărul minutelor de întârziere, ora estimată de plecare și ora estimată de sosire pentru trenul specificat, adunând numărul de minute introdus de utilizator. Deoarece mai mulți utilizatori pot interoga baza de date în același timp, se va utiliza o coadă de comenzi cu operații protejate de un lacăt, în care cerinţele vor fi adăugate în funcţie de timpul lor de sosire la server.

Diagrama aplicaţiei

4 Detalii de implementare

Codul serverului

```
class Command
{
public:
    virtual errors execute() = 0;
    virtual int getThreadID() = 0;
};

class Delay : public Command
{
    _thread_data    threadData;
    string          train;
    int             minutes;
public:
    Delay(_thread_data newThreadData, string newTrain, int newMinutes)
    {
        threadData.thread_ID = newThreadData.thread_ID;
        threadData.client_descriptor = newThreadData.client_descriptor;
        train = newTrain;
        minutes = newMinutes;
    }
    errors execute() override
    {
        if (!clients[threadData.thread_ID].isAdministrator)
        {
            unsigned int bytes;
            SEND_MESSAGE(threadData.client_descriptor, "Not logged in as administrator", threadData.thread_ID,
            printf("[server thread %d] Not logged in as administrator\n", threadData.thread_ID);
            FLUSH;
            return SUCCESS;
        }
        try {
            Driver* driver;
            Connection* connection;
            PreparedStatement* preparedStatement;
            ResultSet* resultSet;
            driver = get_driver_instance();
            connection = driver->connect("tcp://127.0.0.1:3306", "root", ""); // create connection
            connection->setSchema(DATABASE); // connect to database
            preparedStatement = connection->prepareStatement("UPDATE departures SET delay=delay+?, ETD=ADDTIME(
            preparedStatement->setInt(1, minutes);
            string minutes_format;
            minutes_format += to_string(minutes * 100);
            preparedStatement->setString(2, minutes_format.c_str());
            preparedStatement->setString(3, train.c_str());
            preparedStatement->executeUpdate();
```

```
    preparedStatement = connection->prepareStatement("UPDATE arrivals SET delay=delay+?, ETA=ADDTIME(ET
    preparedStatement->setInt(1, minutes);
    preparedStatement->setString(2, minutes_format.c_str());
    preparedStatement->setString(3, train.c_str());
    preparedStatement->executeUpdate();
    preparedStatement = connection->prepareStatement("SELECT * FROM departures NATURAL JOIN arrivals WH
    preparedStatement->setString(1, train.c_str());
    resultSet = preparedStatement->executeQuery();
    string answer;
    char line[MAXIMUM_LINE_LENGTH];
    CHECK_RETURN_THREAD(sprintf(line, "%-9s%-29s%-29s%-5s%-19s%-19s%-10s%-10s%-10s\n", "Number", "Depar
                                "Line", "Departure time", "Arrival time", "Delay", "ETD", "ETA"), threa
    answer += line;
    bool found = false;
    while (resultSet->next())
    {
        CHECK_RETURN_THREAD(sprintf(line, "%-9s%-29s%-29s%-5s%-19s%-19s%-10d%-10s%-10s\n", resultSet->g
                                    resultSet->getString("arrival_station")->c_str(), resultSet->getStr
                                    resultSet->getInt("delay"), resultSet->getString("ETD").c_str(), re
        answer += line;
        found = true;
    }
    delete resultSet;
    delete preparedStatement;
    delete connection;
    if (!found)
    {
        unsigned int bytes;
        SEND_MESSAGE(threadData.client_descriptor, "No entries modified", threadData.thread_ID, "write
        printf("[server thread %d] No entries modified\n", threadData.thread_ID);
        FLUSH;
    }
    else
    {
        unsigned int bytes;
        SEND_MESSAGE(threadData.client_descriptor, answer.c_str(), threadData.thread_ID, "write answer
        printf("[server thread %d] Modified delay for train %s with %d minutes\n", threadData.thread_ID
        FLUSH;
    }
    return SUCCESS;
}
catch (SQLException& exception)
{
    string message;
    message += "Delay for train ";
    message += train;
    message += " with ";
    message += to_string(minutes);
    message += " minutes:\n";
    message += get_exception(exception);
```

```
            unsigned int bytes;
            SEND_MESSAGE(threadData.client_descriptor, message.c_str(), threadData.thread_ID, "write answer byt
            printf("[server thread %d] %s\n", threadData.thread_ID, message.c_str());
            FLUSH;
            return SQL;
        }
    }

    int getThreadID() override
    {
        return threadData.thread_ID;
    }
};

enum errors treat(_thread_data* thread_data)
{
    while (true)
    {
        unsigned int bytes;
        char request[MAXIMUM_REQUEST_LENGTH];
        MEMSET(request);
        RECEIVE_MESSAGE(thread_data->client_descriptor, &request, thread_data->thread_ID, "read request bytes",
        _type type;
        _arguments arguments;
        if (set(request, type, arguments) != SUCCESS)
        {
            SEND_MESSAGE(thread_data->client_descriptor, UNKNOWN, thread_data->thread_ID, "write answer bytes",
            continue;
        }
        switch (type)
        {
            case _REGISTER:
            {
                unique_ptr<Command> _register = make_unique<Register>(*thread_data, arguments.username, argumer
                queue_mutex.lock();
                commands.push(_register.get());
                Command* command = commands.front();
                commands.pop();
                queue_mutex.unlock();
                command->execute();
                break;
            }
            case _LOGIN:
            {
                unique_ptr<Command> login = make_unique<Login>(*thread_data, arguments.username, arguments.pass
                queue_mutex.lock();
                commands.push(login.get());
                Command* command = commands.front();
                commands.pop();
                queue_mutex.unlock();
```

```
    command->execute();
    break;
}
case _EXIT:
{
    SEND_MESSAGE(thread_data->client_descriptor, EXITED, thread_data->thread_ID, "write answer byte
    CLOSE_THREAD(thread_data->client_descriptor, thread_data->thread_ID, "client close");
    return SUCCESS;
}
case _ALL:
{
    unique_ptr<Command> all = make_unique<All>(*thread_data);
    queue_mutex.lock();
    commands.push(all.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _ALL_BETWEEN:
{
    unique_ptr<Command> all_between = make_unique<AllBetween>(*thread_data, arguments.station1, arg
    queue_mutex.lock();
    commands.push(all_between.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _DEPARTURES_FROM:
{
    unique_ptr<Command> departures_from = make_unique<DeparturesFrom>(*thread_data, arguments.stati
    queue_mutex.lock();
    commands.push(departures_from.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _DEPARTURES_AFTER:
{
    unique_ptr<Command> departures_after = make_unique<DeparturesAfter>(*thread_data, arguments.hou
    queue_mutex.lock();
    commands.push(departures_after.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
```

```
    command->execute();
    break;
}
case _DEPARTURES_BETWEEN:
{
    unique_ptr<Command> departures_between = make_unique<DeparturesBetween>(*thread_data, arguments
    queue_mutex.lock();
    commands.push(departures_between.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _ARRIVALS_TO:
{
    unique_ptr<Command> arrivals_to = make_unique<ArrivalsTo>(*thread_data, arguments.station1);
    queue_mutex.lock();
    commands.push(arrivals_to.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _ARRIVALS_AFTER:
{
    unique_ptr<Command> arrivals_after = make_unique<ArrivalsAfter>(*thread_data, arguments.hour1);
    queue_mutex.lock();
    commands.push(arrivals_after.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _ARRIVALS_BETWEEN:
{
    unique_ptr<Command> arrivals_between = make_unique<ArrivalsBetween>(*thread_data, arguments.hou
    queue_mutex.lock();
    commands.push(arrivals_between.get());
    Command* command = commands.front();
    commands.pop();
    queue_mutex.unlock();
    command->execute();
    break;
}
case _ROUTE:
{
    unique_ptr<Command> route = make_unique<Route>(*thread_data, arguments.train);
```

```
                queue_mutex.lock();
                commands.push(route.get());
                Command* command = commands.front();
                commands.pop();
                queue_mutex.unlock();
                command->execute();
                break;
            }
            case _DELAY:
            {
                unique_ptr<Command> delay = make_unique<Delay>(*thread_data, arguments.train, arguments.minutes
                queue_mutex.lock();
                commands.push(delay.get());
                Command* command = commands.front();
                commands.pop();
                queue_mutex.unlock();
                command->execute();
                break;
            }
            default:
                SEND_MESSAGE(thread_data->client_descriptor, UNKNOWN, thread_data->thread_ID, "write answer byt
        }
    }
}

int main(int argc, char** argv)
{
    if (argc < 2)
    {
        printf("syntax: %s <port>\n", argv[0]);
        FLUSH;
        return -1;
    }
    initialize_users();
    initialize_trains();
    int port = atoi(argv[1]);
    int socket_descriptor = socket(AF_INET, SOCK_STREAM, 0); // IPv4, bidirectional bytes streaming
    CHECK_RETURN(socket_descriptor, "[server] socket create", SOCKET);
    int option = 1;
    CHECK_RETURN(setsockopt(socket_descriptor, SOL_SOCKET, SO_REUSEADDR, &option, sizeof(option)), "[server] so
    sockaddr_in server{};
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = htonl(INADDR_ANY); // localhost
    server.sin_port = htons(port);
    CHECK_RETURN(bind(socket_descriptor, (sockaddr*)&server, sizeof(sockaddr)), "[server] socket bind", BIND);
    CHECK_RETURN(listen(socket_descriptor, BACKLOG), "[server] socket listen", LISTEN); // pending connections
    printf("[server] Waiting at port %d\n", port);
    FLUSH;
    sockaddr_in client{};
```

```
    memset(&client, 0, sizeof(client));
    int client_descriptor;
    socklen_t client_length = sizeof(client);
    pthread_t threads[THREADS];
    _thread_data threads_data[THREADS];
    int index = -1;
    while (true)
    {
        client_descriptor = accept(socket_descriptor, (sockaddr*)&client, &client_length); // connection
        CHECK_CONTINUE(client_descriptor, "[server] socket accept");
        ++index;
        threads_data[index].thread_ID = index + 1;
        threads_data[index].client_descriptor = client_descriptor;
        CHECK_THREAD(pthread_create(&threads[index], nullptr, &routine, (void*)&threads_data[index]), threads_d
    }
}
```

Codul clientului

```
void show_commands()
{
    printf("Commands:\n");
    printf("\t%-20s%-12s%-12s\n", "register", "<username>", "<password>");
    printf("\t%-20s%-12s%-12s\n", "login", "<username>", "<password>");
    printf("\tall\n");
    printf("\t%-20s%-12s%-12s\n", "all-between", "<station>", "<station>");
    printf("\t%-20s%-12s\n", "departures-from", "<station>");
    printf("\t%-20s%-12s\n", "departures-after", "<hour>");
    printf("\t%-20s%-12s%-12s\n", "departures-between", "<hour>", "<hour>");
    printf("\t%-20s%-12s\n", "arrivals-to", "<station>");
    printf("\t%-20s%-12s\n", "arrivals-after", "<hour>");
    printf("\t%-20s%-12s%-12s\n", "arrivals-between", "<hour>", "<hour>");
    printf("\t%-20s%-12s\n", "route", "<train>");
    printf("\t%-20s%-12s%-12s\n", "delay", "<train>", "<minutes>");
    printf("\texit\n");
    FLUSH;
}

int main(int argc, char** argv)
{
    if (argc < 4)
    {
        printf("syntax: %s <address> <port> <type>\n", argv[0]);
        FLUSH;
        return -1;
    }
    int port = atoi(argv[2]);
    int socket_descriptor = socket(AF_INET, SOCK_STREAM, 0);
    CHECK(socket_descriptor, "[client] create socket", SOCKET);
    sockaddr_in server{};
    memset(&server, 0, sizeof(server));
    server.sin_family = AF_INET;
    server.sin_addr.s_addr = inet_addr(argv[1]);
    server.sin_port = htons(port);
    CHECK(connect(socket_descriptor, (sockaddr*)&server, sizeof(sockaddr)), "[client] connect", CONNECT);
    printf("[client] connected to port %d\n", port);
    FLUSH;
    unsigned int bytes;
    char answer[MAXIMUM_ANSWER_LENGTH];
    int type = atoi(argv[3]);
    if (type == 1)
    {
        show_commands();
        char command[MAXIMUM_COMMAND_LENGTH];
        while (true)
        {
            printf("command> ");
```

```
            FLUSH;
            MEMSET(command);
            CHECK(read(STDIN, command, MAXIMUM_COMMAND_LENGTH), "[client] read command", READ);
            command[strlen(command) - 1] = '\0';
            SEND_MESSAGE(socket_descriptor, command, "[client] write command bytes", "[client] write command me
            MEMSET(answer);
            RECEIVE_MESSAGE(socket_descriptor, &answer, "[client] read answer bytes", "[client] read answer mes
            printf("%s\n", answer);
            FLUSH;
            if (strcmp(answer, EXITED) == 0)
            {
                break;
            }
        }
    }
    else if (type == 2)
    {
        vector<string> commands;
        commands.emplace_back("register georgiana_ojoc 3007");
        commands.emplace_back("login administrator ?Kv8H");
        commands.emplace_back("all");
        commands.emplace_back("all-between Iasi Tecuci");
        commands.emplace_back("departures-from Crasna");
        commands.emplace_back("departures-after 15");
        commands.emplace_back("departures-between 12 21");
        commands.emplace_back("arrivals-to Buzau");
        commands.emplace_back("arrivals-after 17");
        commands.emplace_back("arrivals-between 6 11");
        commands.emplace_back("route 1662");
        commands.emplace_back("delay 1838 5");
        commands.emplace_back("exit");
        for (auto command : commands)
        {
            SEND_MESSAGE(socket_descriptor, command.c_str(), "[client] write command bytes", "[client] write co
            MEMSET(answer);
            RECEIVE_MESSAGE(socket_descriptor, &answer, "[client] read answer bytes", "[client] read answer mes
            printf("%s\n", answer);
            FLUSH;
            if (strcmp(answer, EXITED) == 0)
            {
                break;
            }
        }
    }
    CLOSE(socket_descriptor, "[client] close socket");
    return 0;
}
```

Scenariu de utilizare

5 Concluzii

Așadar, aplicaţia ar putea fi mai atractivă dacă se implementează o interfaţă grafică utilizând mediul de dezvoltare *Qt Creator*.

6 Bibliografie

- https://appiris.infofer.ro
- https://infofer.ro/index.php/ro/
- https://ro.wikipedia.org/wiki/TCP/IP
- https://en.wikipedia.org/wiki/Command_pattern
- https://profs.info.uaic.ro/~computernetworks/cursullaboratorul.php
- https://dev.mysql.com/doc/connector-cpp/1.1/en/