Compte Rendu - TP Client/Serveur TCP/IP avec Qt

Sommaire

Introduction

Ce rapport décrit la création d'une application client/serveur utilisant le protocole TCP/IP et le framework Qt. L'objectif principal est de créer des applications client et des applications serveur capables de communiquer selon des protocoles définis.

Questions Préliminaires :

1. Principe de la notion de client/serveur en informatique : Le modèle client-serveur est un paradigme de communication dans lequel un serveur fournit des services ou des ressources à un ou plusieurs clients. Le client envoie une requête au serveur, et le serveur traite la requête et renvoie une réponse. Cela permet l'allocation des tâches et la gestion des ressources.

2. Qu'est-ce qu'un protocole ? A quoi sert-il ? Un protocole est un ensemble de règles et de conventions qui définissent la manière dont les données sont échangées entre les parties communicantes. Dans ce cas, le protocole décrit le format des messages entre client et serveur et les actions à réaliser en réponse à ces messages.

3. Expliquer la notion de port et de socket sous TCP/IP :

- Un port est un numéro associé à un processus sur un ordinateur. Il permet de diriger le trafic réseau vers le bon processus.
- Un socket est un point de communication bidirectionnel entre deux ordinateurs sur un réseau. Il est défini par une adresse IP et un port.

4. En utilisant l'aide de Qt, quels sont les classes de Qt permettant la création d'une application cliente et d'une application serveur ? Qt propose des classes pour créer des applications client/serveur TCP/IP. Pour le client, on peut utiliser `QTcpSocket`, et pour le serveur, on peut utiliser `QTcpServer`.

Applications demandées

Réalisation de l'application cliente TCP/IP en C++

L'application client est développée en C++ avec Qt. Elle comprend une interface utilisateur qui permet de sélectionner les informations à récupérer (température en °C, température en °F ou mesure d'humidité). L'application se connecte au serveur en spécifiant l'adresse IP et le port, ces derniers sont pré remplis permettant de gagner du temps dans le traitement de la demande. Il envoie des trames de requête au serveur, reçoit des réponses, traite les trames et affiche les informations de manière stylé.

```cpp
//Affichage du resultat
void InitiationQt::onSocketReadyRead(){
    QByteArray data = socket->read(socket->bytesAvailable());
    QString str(data);

    int indexVirgule = str.indexOf(',');  // trouver l'index de la virgule

    QString avantLaVirgule;
    QString aprèsLaVirgule;

    if (indexVirgule != -1 && indexVirgule < str.length() - 1) {
        avantLaVirgule = str.left(indexVirgule).trimmed();
        aprèsLaVirgule = str.mid(indexVirgule + 1).trimmed();
    }

    ui.connectionStatusLabel1->setText("la valeur demandee pour le capteur " + avantLaVirgule + " est : " + aprèsLaVirgule);
}
//
void InitiationQt::onServerNewConnection(){
    ui.connectionStatusLabel->setText("Un client s'est connecte");
    QTcpSocket * client = server->nextPendingConnection();
    QObject::connect(client, SIGNAL(readyRead()), this, SLOT(onClientReadyRead()));
    QObject::connect(client, SIGNAL(disconnected()), this, SLOT(onClientDisconnected()));
}
void InitiationQt::onClientDisconnected(){
    QTcpSocket *obj = qobject_cast<QTcpSocket*>(sender());
    QObject::disconnect(obj, SIGNAL(readyRead()), this, SLOT(onClientReadyRead()));
    QObject::disconnect(obj, SIGNAL(disconnected()), this, SLOT(onClientDisconnected()));
    obj->deleteLater();
}
void InitiationQt::onClientReadyRead(){
    QTcpSocket *obj = qobject_cast<QTcpSocket*>(sender());
    QByteArray coucou = obj->read(obj->bytesAvailable());
    QString str(coucou);
    ui.connectionStatusLabel1->setText("Message recu = " + str);
    obj->write(coucou);
}
```

```
}

//Code bouton Celcius
void InitiationQt::EnvoiCel() {
    QString cel = "Td";
    QString num = ui.plainTextEdit->toPlainText();
    QByteArray data = cel.toUtf8();
    QByteArray data2 = num.toUtf8();
    socket->write(data + data2);

}
// code bouton Far
void InitiationQt::EnvoiFar() {

    QString far = "Tf";
    QString num = ui.plainTextEdit->toPlainText();
    QByteArray data = far.toUtf8();
    QByteArray data2 = num.toUtf8();
    socket->write(data + data2);



}
// Code bouton hygrométrie
void InitiationQt::EnvoiHyg() {

    QString hyg = "Hr";
    QString num = ui.plainTextEdit->toPlainText();
    QByteArray data = hyg.toUtf8();
    QByteArray data2 = num.toUtf8();
    socket->write(data + data2);



}
```

Réalisation de l'application serveur TCP/IP en C++

L'application serveur a également été développée en C++ avec Qt. Elle utilise la classe QTcpServer pour créer un serveur TCP. L'application traite les requêtes provenant du client en suivant un protocole de communication spécifique. Elle génère aléatoirement des valeurs de température et d'hygrométrie dans les plages spécifiées et renvoie des réponses au client selon le protocole qui a été établie dans la consigne du TP.

```cpp
#include <QCoreApplication>
#include <QDebug>
#include "server.h"

int main(int argc, char *argv[])
{
    QCoreApplication app(argc, argv); // On initialise QT
    Server server;

    if (!server.listen(QHostAddress::Any, 12345)) {
        qCritical() << "Erreur de lancement du serveur."; // Si le serveur ne démarre pas
        return 1;
    }
    else {
        qDebug() << "Le serveur est config sur le port 12345."; // Indication de connexion au port
    }

    // Exemple d'affichage de l'état de connexion
    if (server.isClientConnected()) {
        qDebug() << "Un client est connecte au serveur."; // Si un user est connecté
    }
    else {
        qDebug() << "Aucun client n est connecte au serveur."; // Si aucun user n'est connecté
    }

    return app.exec();
}
```

```cpp
#ifndef SERVER_H
#define SERVER_H

#include <QTcpServer>

class Server : public QTcpServer // On déclare la classe
{
    Q_OBJECT
public:
    Server(QObject *parent = nullptr); // déclaration du constructeur
    bool isClientConnected() const; // Déclaration de la fonction d'ino de connexion

protected:
    void incomingConnection(qintptr socketDescriptor) override; // Pour une connection entrante

private:
    void processRequest(const QByteArray &request, QTcpSocket *clientSocket); // requete
    bool clientConnected = false; // Client connecté ? défaut = false
};

#endif // SERVER_H
```

```cpp
#include "server.h"
#include <QTcpSocket>
#include <QRandomGenerator>
#include <QDebug>

Server::Server(QObject *parent) : QTcpServer(parent) {
    // Initialisation du serveur
}

void Server::incomingConnection(qintptr socketDescriptor) {
    // Nouvelle connexion entrante
    QTcpSocket *clientSocket = new QTcpSocket(this);
    clientSocket->setSocketDescriptor(socketDescriptor);

    // Récupérer l'adresse IP du client
    QHostAddress clientAddress = clientSocket->peerAddress();
    QString clientIPv4 = clientAddress.toString();

    // Supprimer "::ffff:" si présent
    if (clientIPv4.startsWith("::ffff:")) {
        clientIPv4 = clientIPv4.mid(7); // Pour supprimer les 7 premiers caractères "::ffff:"
    }


    // On met à jour l'état de connexion
    qDebug() << "Nouvelle connexion entrante depuis : " << clientIPv4;
    clientConnected = true;
    qDebug() << "Etat de connexion mis a jour : le client est connecte au serveur.";


    connect(clientSocket, &QTcpSocket::readyRead, this, [clientSocket, this]() {
        // Lorsque des données sont prêtes à être lues
        QByteArray data = clientSocket->readAll();
        processRequest(data, clientSocket);
        });
}
bool Server::isClientConnected() const {
    return clientConnected; // Une personne est connectée
}
```

Conclusion

Ce TP a permis de comprendre les concepts de base liés aux applications client/serveur, aux protocoles, aux ports et aux sockets. Les applications créées utilisent Qt pour la communication TCP/IP et respectent un protocole défini. Les applications cliente et serveur fonctionnent correctement et sont capables d'interagir de manière efficace.

Annexe : Code Source

Voici le lien du repository GitHub : https://github.com/HugoTby/TP-TCP-IP