La Providence

TP TCP IP avec Qt

TCP Client / TCP Serveur

POLLET Quentin && LECRONIER Éloïse
17/10/2023

Sommaire

POLLET Quentin
LECRONIER Éloïse

Introduction

Présentation du projet

Dans ce TP, nous devons réaliser une simulation d'un échange entre un client et un serveur TCP en utilisant Qt et le C++.

Le but de cet échange consiste à ce que quand le client envoie une requête au serveur, où il peut demander :

- La température en Celsius (Tdxx)
- La température en Farhenheit (Thxx)
- L'hygrométrie (Hrxx)

Au lancement, l'application cliente doit être capable d'afficher une interface graphique, où l'utilisateur peut appuyer sur les données qu'il souhaite récupérer de se connecter à un serveur en spécifiant l'adresse IP et le port, d'envoyer une requête au serveur, de recevoir et de traiter la réponse, puis d'afficher les informations de manière claire pour l'utilisateur.

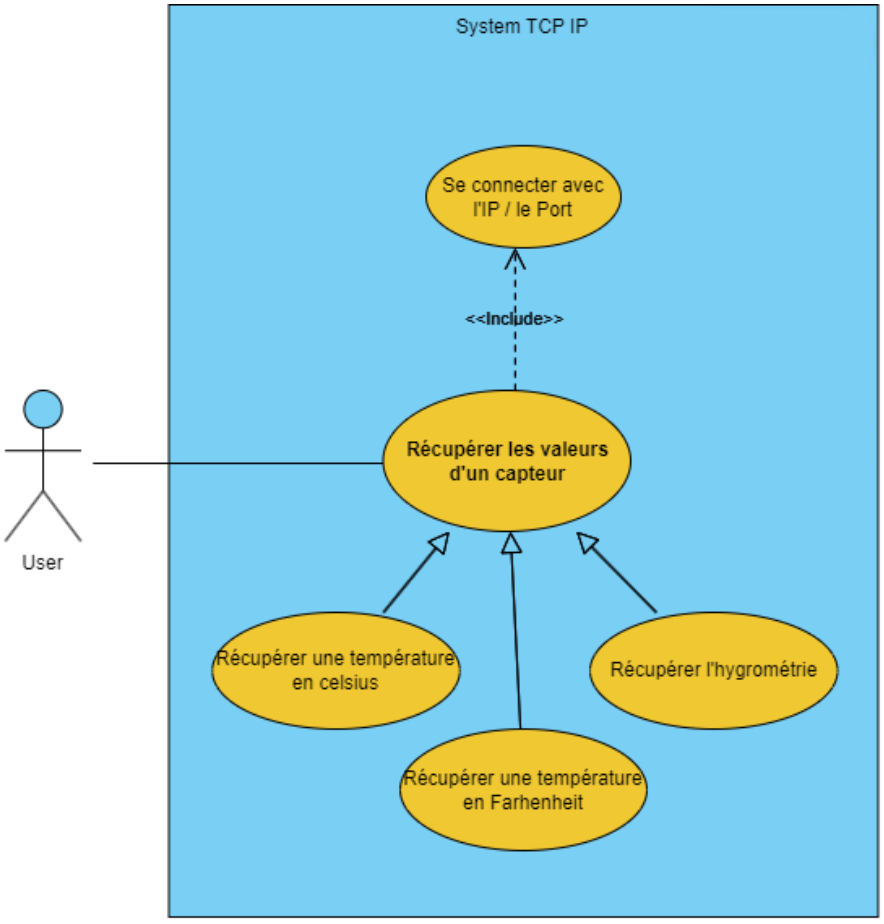
L'application serveur de son côté doit être capable de traiter les requêtes des clients, de simuler la température ou l'hygrométrie en générant aléatoirement des valeurs dans des plages spécifiées, et de retourner des réponses au client.

Les objectifs

Les objectifs du projet est la réalisation d'une application cliente en TCP/IP en C++ disposant d'une interface IHM où l'on peut sélectionner les informations désirées comme la température en degré, la température en Fahrenheit ou encore l'hygrométrie.

- On doit pouvoir se connecter au serveur en ayant la possibilité d'inscrire l'adresse IP et le port auquel on souhaite se connecter. L'état de connexion devra être affiché du côté Client comme du côté Serveur.
- Lorsque l'utilisateur est connecté, il peut indiquer le capteur sur lequel il souhaite récupérer les données. Ce facteur sera pris en compte lors de la requête TCP d'interrogation, quand il appuiera sur un bouton de température.
- Le Serveur doit être capable de réceptionner et d'interpréter correctement l'interrogation reçue afin de générer une valeur aléatoire sur la plage indiquée, en fonction de la température.
- Le Serveur renvoie ensuite une trame contenant toutes les informations demandées par le client.
- Le Client reçoit les informations via le Socket et affiche le résultat.

Use Case



Lien Github

Notre repository Github est disponible sur ce lien.

POLLET Quentin
LECRONIER Éloïse

Questions préliminaires

1. Quel est le principe de la notion client / serveur en informatique ?

Le principe de la notion client / server en informatique est un mode de communication en réseau entre plusieurs services ou programmes. Nous avons le client qui va généralement faire une demande ou des envoies d'information(s) au serveur. Le serveur va lui se charger d'attendre les demandes ou les envois d'information(s) de différents clients afin d'y répondre et d'établir une communication.

2. Qu'est-ce qu'un protocole ? A quoi sert-il ?

Un protocole en informatique est le moyen de spécifier par quels moyens, quelles règles on souhaite envoyer des données. Il en existe plusieurs : le TCP, l'UDP, l'IP, le FTP…

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

Un port sous TCP/IP est un numéro qui permet de distinguer différentes applications ou services sur un même appareil ou réseau. Ils sont utilisés par un logiciel en particulier et servent à acheminer les données vers une application.

Un socket se compose de 3 arguments : une adresse IP, un protocole de transport et un port dédié qui permettent de communiquer via ce même protocole. Ils sont utilisés pour identifier une machine et/ou un service sur une machine.

4. En utilisant l'aide de Qt, quels sont les classes Qt permettant la création d'une application client et d'une application serveur ?

Les classes de Qt qui permettent la création d'une application cliente et d'une application serveur sont :

Application cliente -> QLocalSocket, QSctpSocket, QTcpSocket

Application serveur -> QLocalServer, QSctpServer, QTcpServer

POLLET Quentin
LECRONIER Éloïse

Partie Client

C. Attendu

La partie cliente de ce projet vise à créer une application cliente TCP/IP en C++ qui permet de réaliser plusieurs tâches essentielles. Tout d'abord, l'objectif est de développer une interface utilisateur conviviale où l'utilisateur peut choisir le type d'information à récupérer, que ce soit la température en Celsius, en Fahrenheit, ou l'Hygrométrie. Elle doit disposer d'une interface graphique qui permette d'interagir avec elle. Ensuite, l'application cliente doit être capable de se connecter à un serveur distant en permettant à l'utilisateur de spécifier l'adresse IP et le port du serveur. L'état de la connexion doit toujours être affiché. Une fois la connexion établie, l'application doit être en mesure d'envoyer des requêtes conformes au protocole de communication défini (comme Tdxx?, Tfxx?, Hrxx?) au serveur. Lorsque des réponses sont reçues du serveur, l'application doit les traiter et afficher les informations à l'utilisateur.

C. Composition de la solution

Voici la composition de la solution « ClientTCPIP » :

ClientTCIP

```
└── External Dependencies
    └── ...

└── Form Files
    └── ClientTCPIP.ui

└── Header Files
    └── ClientTCIP.h

└── Resource Files
    └── ClientTCPIP.qrc

└── Source Files
    ├── ClientTCIP.cpp
    └── main.cpp
```

Les fichiers sur lesquels nous allons notamment nous concentrer sont « `ClientTCPIP.cpp` » et « `ClientTCPIP.h` ».

POLLET Quentin
LECRONIER Éloïse

1) Le fichier « `ClientTCPIP.cpp` » comporte 10 méthodes :

- ClientTCPIP::ClientTCPIP(QWidget *parent) : QMainWindow(parent)
  Le constructeur de l'IHM sert à l'initialisation et la configuration de la communication client-serveur.

```
3    // Constructeur IHM
4    ClientTCPIP::ClientTCPIP(QWidget *parent) : QMainWindow(parent)
5    {
6        ui.setupUi(this);
7
8        // Rendre invisible et désactiver les boutons tant que la connexion n'est pas établie
9        ui.pushBtnCelsius->setVisible(false);
10       ui.pushBtnCelsius->setEnabled(false);
11
12       ui.pushBtnFahrenheit->setVisible(false);
13       ui.pushBtnFahrenheit->setEnabled(false);
14
15       ui.pushBtnHygrometie->setVisible(false);
16       ui.pushBtnHygrometie->setEnabled(false);
17
18       // Rendre invisible le champ et le label du numéro du capteur
19       ui.numcapteur->setVisible(false);
20       ui.numcapteur->setEnabled(false);
21
22       ui.labelCapteur->setVisible(false);
23       ui.labelCapteur->setEnabled(false);
24
25       socketClient = new QTcpSocket(this); // Inialisation du socket client pour notre interface
26       QObject::connect(socketClient, SIGNAL(connected()), this, SLOT(onSocketConnected())); // On
27       QObject::connect(socketClient, SIGNAL(disconnected()), this, SLOT(onSocketDisconnected()));
28       QObject::connect(socketClient, SIGNAL(readyRead()), this, SLOT(onSocketReadyRead())); // On
29   }
```

- ClientTCPIP::~ClientTCPIP()
  Le destructeur de la classe est chargé de fermer la connexion et supprimer l'objet Qtcpsocket.

```
31   // Destructeur IHM
32   ClientTCPIP::~ClientTCPIP()
33   {
34       socketClient->close(); /
35       delete socketClient; //
36   }
```


POLLET Quentin
LECRONIER Éloïse

- void ClientTCPIP::onConnectButtonClicked()
  Méthode utilisée lorsque l'utilisateur clique sur le bouton de connexion. Elle gère la tentative de connexion au serveur en fonction de l'adresse IP et du port spécifiés.

```
38    // Méthode du slot lors du clic sur le bouton de connexion
39    void ClientTCPIP::onConnectButtonClicked()
40    {
41        // On récupère ce qui a été saisi
42        QString ip = ui.lineEditIP->text();
43        QString port = ui.lineEditPort->text();
44
45        // Vérifier si les champs ne sont pas pas vide
46        if (!ip.isEmpty() && !port.isEmpty())
47        {
48            // On converti le port de String en int
49            bool ok;
50            int portAsInt = port.toInt(&ok);
51
52            if (ok) // Si la conversion est réussie
53            {
54                // On va essayer de se connecter au serveur
55                socketClient->connectToHost(ip, portAsInt);
56            }
57        }
58        else
59        {
60            // le(s) champ(s) est vide -> Message d'erreur
61            ui.label_error->setText("<font color='red'>Veuillez remplir les champs</font>");
62        }
63    }
```

- void ClientTCPIP::onSocketConnected()
  Méthode appelée lorsque le signal 'connected' est émis par le socket client. Elle permet d'afficher les boutons des températures et du numéro de capteur quand le client est bien connecté au serveur.

```
65    // Méthode du slot lorsque le signal connected est émis par socketClient
66    void ClientTCPIP::onSocketConnected()
67    {
68        ui.labelStatus->setText("Status connexion : Connecté");
69
70        // Rendre visible et activer les boutons et effacer le texte d'erreu
71        ui.pushBtnCelsius->setVisible(true);
72        ui.pushBtnCelsius->setEnabled(true);
73
74        ui.pushBtnFahrenheit->setVisible(true);
75        ui.pushBtnFahrenheit->setEnabled(true);
76
77        ui.pushBtnHygrometie->setVisible(true);
78        ui.pushBtnHygrometie->setEnabled(true);
79
80        ui.label_error->setText("");
81
82        // désactiver et cacher le bouton connexion
83        ui.pushBtnConnect->setEnabled(false);
84        ui.pushBtnConnect->setVisible(false);
85
86        // désactiver les labels ip et port
87        ui.labelIP->setVisible(false);
88        ui.labelPort->setVisible(false);
89
```

etc ...

POLLET Quentin
LECRONIER Éloïse

- void ClientTCPIP::onSocketDisconnected()
  Méthode appelée lorsque le signal ‘disconnected’ est émis par le socket client. Elle déconnecte l’utilisateur quand le serveur n’est plus accessible.

```
105     // Méthode du slot lorsque le signal disconnected est émis par socketClient
106     void ClientTCPIP::onSocketDisconnected()
107     {
108         ui.labelStatus->setText("Status connexion : Déconnecté");
109
110         // Rendre invisible et désactiver les boutons tant que la connexion n'e
111         ui.pushBtnCelsius->setVisible(false);
112         ui.pushBtnCelsius->setEnabled(false);
113
114         ui.pushBtnFahrenheit->setVisible(false);
115         ui.pushBtnFahrenheit->setEnabled(false);
116
117         ui.pushBtnHygrometie->setVisible(false);
118         ui.pushBtnHygrometie->setEnabled(false);
119
120         // activer et rendre visible le bouton connexion
121         ui.pushBtnConnect->setEnabled(true);
122         ui.pushBtnConnect->setVisible(true);
123
124         // rendre visible les labels ip et port
125         ui.labelIP->setVisible(true);
126         ui.labelPort->setVisible(true);
127
128         // activer et rendre visible les champs de saisi ip et port
129         ui.lineEditIP->setEnabled(true);
130         ui.lineEditIP->setVisible(true);
            etc...
```

- void ClientTCPIP::onSocketReadyRead()
  Méthode appelée quand le signal ‘readyRead’ est émis par le socket client. Elle lit les données reçues du serveur et les affiche dans l'interface.

```
142     // Méthode du slot lorsque le signal readyRead est émis par socketClient
143     void ClientTCPIP::onSocketReadyRead()
144     {
145         QByteArray data = socketClient->read(socketClient->bytesAvailable()); // On va lire les données reçu
146         QString str(data); // On converti nos données en une chaine de caractère
147
148         if (str.left(2) == "Td")
149         {
150             ui.labelStatus->setText("Message du serveur : " + str.right(6) + "C (Celsius) " + " Capteur " + str.mid(2, 1) + str.mid(3, 1)); //
151         }
152         else if (str.left(2) == "Tf")
153         {
154             ui.labelStatus->setText("Message du serveur : " + str.right(6) + "F (Farenheit) " + " Capteur " + str.mid(2, 1) + str.mid(3, 1));
155         }
156         else if (str.left(2) == "Hr")
157         {
158             ui.labelStatus->setText("Message du serveur : " + str.right(5) + "% (Hygrometrie) Capteur " + str.mid(2, 1) + str.mid(3, 1)); // C
159         }
160         else
161         {
162             ui.labelStatus->setText("Message du serveur : " + str + " Capteur "); // On affiche les données reçu
163         }
164     }
```


POLLET Quentin
LECRONIER Éloïse

- void ClientTCPIP::onSendCelsiusButtonClicked()
  Méthode appelée lorsque l'utilisateur clique sur le bouton ‘pushBtnCelsius’ pour demander la température en Celsius. Elle génère la requête correspondante et l'envoie au serveur.

```
160     // Méthode du slot quand on demande la température en Celsius
161     void ClientTCPIP::onSendCelsiusButtonClicked()
162     {
163         // On récupère la valeur de la spinbox
164         int numcapteur = ui.numcapteur->value();
165
166         QString requestServer;
167
168         if (numcapteur < 10) // Si le numéro ne dépasse pas 10
169         {
170             requestServer = "Td0" + QString::number(numcapteur) + "?"; /
171         }
172         else
173         {
174             requestServer = "Td" + QString::number(numcapteur) + "?";
175         }
176
177         if (socketClient->state() == QTcpSocket::ConnectedState) // Si l
178         {
179             socketClient->write(requestServer.toUtf8()); // On envoie le
180         }
181     }
```

- void ClientTCPIP::onSendFarhenheitButtonClicked()
  Méthode utilisée lorsque l'utilisateur clique sur le bouton ‘pushBtnFahrenheit’ pour demander la température en Fahrenheit. Elle génère la requête correspondante et l'envoie au serveur.

```
183     // Méthode du slot lorsque l'on demande une température en Farenheit
184     void ClientTCPIP::onSendFarhenheitButtonClicked()
185     {
186         // On récupère la valeur de la spinbox
187         int numcapteur = ui.numcapteur->value();
188
189         QString requestServer;
190
191         if (numcapteur < 10) // Si le numéro ne dépasse pas 10
192         {
193             requestServer = "Tf0" + QString::number(numcapteur) + "?"; /
194         }
195         else
196         {
197             requestServer = "Tf" + QString::number(numcapteur) + "?";
198         }
199
200         if (socketClient->state() == QTcpSocket::ConnectedState) // Si l
201         {
202             socketClient->write(requestServer.toUtf8()); // On envoie le
203         }
204     }
```


POLLET Quentin
LECRONIER Éloïse

- void ClientTCPIP::onSendHygrometrieButtonClicked()
  Méthode utilisée lorsque l'utilisateur clique sur le bouton ‘pushBtnHygrometrie’ pour demander l'hygrométrie. Elle génère la requête correspondante et l'envoie au serveur.

```
206     // Méthode du slot quand on demande une valeur Hygrometrie
207     void ClientTCPIP::onSendHygrometrieButtonClicked()
208     {
209         // On récupère la valeur de la spinbox
210         int numcapteur = ui.numcapteur->value();
211
212         QString requestServer;
213
214         if (numcapteur < 10) // Si le numéro ne dépasse pas 10
215         {
216             requestServer = "Hr0" + QString::number(numcapteur) + "?";
217         }
218         else
219         {
220             requestServer = "Hr" + QString::number(numcapteur) + "?";
221         }
222
223         if (socketClient->state() == QTcpSocket::ConnectedState) // Si
224         {
225             socketClient->write(requestServer.toUtf8()); // On envoie
226         }
227     }
```

2) Le fichier « ClientTCPIP.h » comporte les déclarations des méthodes :

```
20  public slots:
21      void onConnectButtonClicked(); // Clic Bouton de connexion
22      void onSocketConnected(); // socketClient signal connecter
23      void onSocketDisconnected(); // socketClient signal déconnecter
24      void onSendMessageButtonClicked(); // Clic Bouton d'envoie de message
25      void onSocketReadyRead(); // socketClient signal readyRead
26      void onSendCelsiusButtonClicked(); // Demande Celsius Button
27      void onSendFarhenheitButtonClicked(); // Demande Farheneit Button
28      void onSendHygrometrieButtonClicked(); // Demande Hygrometrie Button
```


POLLET Quentin
LECRONIER Éloïse

Partie Serveur

S. Attendu

Le serveur doit écouter les demandes entrantes du client et générer les réponses appropriées, qu'il envoie via des trames. Il doit traiter les demandes via le protocole TCP en comprenant le type de demande (tel que la température en °C, °F ou l'hygrométrie) ainsi que le capteur visé. Le serveur doit ensuite générer des données aléatoires comprises entre les plages indiquées et renvoyer tout ça au client via un socket.

S. Composition de la solution

Voici la composition de la solution « ServeurTCP » :

ServeurTCP

```
└── External Dependencies
    └── ...

└── Header Files
    └── ServeurTCPIP.h

└── Source Files
    ├── ServeurTCPIP.cpp
    └── main.cpp
```

Les fichiers sur lesquels nous allons notamment nous concentrer sont « `ClientTCPIP.cpp` » et « `ClientTCPIP.h` ».

1) Le fichier « `ServeurTCPIP.cpp` » comporte 5 méthodes :

- ServeurTCPIP::ServeurTCPIP(QObject *parent) : QObject(parent)

  C'est le constructeur. Il crée un objet QTcpServer et connecte son signal '`NewConnection`' au slot '`onServerNewConnection`'. Il est configuré pour écouter sur le port 1234.

POLLET Quentin
LECRONIER Éloïse

```
3    // Constructeur
4    ServeurTCPIP::ServeurTCPIP(QObject *parent) : QObject(parent)
5    {
6        socketServer = new QTcpServer(this); // Inialisation du socket server pour notre int
7        connect(socketServer, SIGNAL(newConnection()), this, SLOT(onServerNewConnection()));
8
9        // On ouvre le serveur sur n'importe quelle adresse ipv4 (127.0.0.1 ou ip sur le rés
10       if (socketServer->listen(QHostAddress::AnyIPv4, 1234))
11       {
12           qDebug() << "Ouverture du serveur sur le port 1234";
13       }
14       else
15       {
16           qDebug() << "ERREUR lors de l'ouverture du port";
17       }
18   }
```

- ServeurTCPIP::~ServeurTCPIP()

  C'est le destructeur qui permet de fermer la connexion du SocketServer, ainsi que de le supprimer de la mémoire.

```
20   // Destructeur
21   ServeurTCPIP::~ServeurTCPIP()
22   {
23       socketServer->close(); // On ferme la connexion
24       delete socketServer; // On Supprime pour la mémoire
25   }
```

- void ServeurTCPIP::onServerNewConnection()

  Cette méthode slot est appelée lorsque le signal '`newConnection`' du serveur est lancé. Il permet une connexion du client grâce à '`nextPendingConnection`' et connecte les signaux '`readyRead`' et '`disconnected`'.

```
27   // Méthode du slot lorsque le signal newConnection est émis par socketServer
28   void ServeurTCPIP::onServerNewConnection()
29   {
30       QTcpSocket* client = socketServer->nextPendingConnection(); // On récupère le
31
32       qDebug() << "Un client s'est connecter : " << client->peerAddress();
33
34       connect(client, SIGNAL(readyRead()), this, SLOT(onClientReadyRead())); // On
35       connect(client, SIGNAL(disconnected()), this, SLOT(onClientDisconnected()));
36   }
```


POLLET Quentin
LECRONIER Éloïse

- void ServeurTCPIP::onClientDisconnected()

  Cette méthode vient déconnecter le client lorsque le signal ‘disconnected’ est émis. Elle déconnecte les signaux ‘readyRead’ et ‘disconnected’, puis supprime l’objet client pour libérer la mémoire.

```
38    // Méthode du slot lorsque le signal disconnected est émis par client
39    void ServeurTCPIP::onClientDisconnected()
40    {
41        QTcpSocket* client = qobject_cast<QTcpSocket*>(sender()); // On va récupèrer l'
42
43        qDebug() << "Un client s'est deconnecter : " << client->peerAddress();
44
45        disconnect(client, SIGNAL(readyRead()), this, SLOT(onClientReadyRead())); // On
46        disconnect(client, SIGNAL(disconnected()), this, SLOT(onClientDisconnected()));
47
48        client->deleteLater(); // Suppresion de l'object pour libérer la mémoire
49    }
```

- void ServeurTCPIP::onClientReadyRead()

  Cette méthode est utilisée pour générer la réponse appropriée à la demande du client, lorsque le signal ‘readyRead’ est émis. Elle lit la requête envoyée par le client et, en fonction du message, lui renvoie la donnée correspondante.

```
53    // Méthode du slot lorsque le signal readyRead est émis par client
54    void ServeurTCPIP::onClientReadyRead()
55    {
56        QTcpSocket* obj = qobject_cast<QTcpSocket*>(sender()); // On va récupèrer l'objet qui nous un envoyé le signal (c
57
58        QByteArray data = obj->read(obj->bytesAvailable()); // On va lire les données reçu
59        QString str(data); // On converti nos données en une chaine de caractère
60
61        // On va convertir les deux caractères après "Td" ou "Tf" ou "Hr" pour savoir si ce sont des entiers
62        bool isInt, isInt2;
63        int int1 = str.mid(2, 1).toInt(&isInt);
64        int int2 = str.mid(3, 1).toInt(&isInt2);
65
66        QString strNumber;
67        const char* tempFormat;
68
69        float randomTemp = 00.00; // Création de la température
70
71        if (str.left(2) == "Td" && isInt && isInt2) // Si c'est une demande de température en °C
72        {
73            qDebug() << "Demande du client : " << str << " [" << obj->peerAddress() << "]";
74
75            randomTemp = (-20.00) + static_cast <float> (rand()) / (static_cast <float> (RAND_MAX / (37.00 - (-20.00))));
76
77            if (randomTemp < 10 || randomTemp < -10)
78            {
79                if (randomTemp >= 0) {
80                    tempFormat = "+%05.2f";
81                }
82                else if (randomTemp >= -10) {
83                    tempFormat = "%06.2f"; // Correction ici
84                }
85                else {
86                    tempFormat = "%06.2f";
87                }
88
89                strNumber = QString::asprintf(tempFormat, randomTemp);
90                qDebug() << strNumber;
91            }
```

etc...

POLLET Quentin
LECRONIER Éloïse

2) Le fichier « `ServeurTCPIP.h` » comporte les déclarations des méthodes :

```
18      public slots:
19          void onServerNewConnection(); // Nouvelle connexion au serveur
20          void onClientDisconnected(); // client signal déconnecter
21          void onClientReadyRead(); // client signal readyRead
22      };
```


POLLET Quentin
LECRONIER Éloïse

Conclusion

Pour conclure, ce projet nous auras permis de mettre en œuvre une communication réseau TCP IP au moyen de Qt et du C++. Le challenge a été de réussir à faire communiquer un client et un serveur sous certaines conditions, et de leur permettre de s'envoyer des messages qu'ils puissent se comprendre tous les deux. Il aura aussi été un bon exercice pour revoir les notions de Port/Socket ainsi que les Classes, qui nous ont été indispensables pour mener à terme ce projet.

Vive les MCG. (HPI DE MERDE)