Initiation au développement Qt sur les sockets

Philippe Latu
philippe.latu(at)inetdoc.net

https://www.inetdoc.net

Résumé

Ce support est la suite de l'initiation au développement en Langage C sur les sockets. L'objectif est à nouveau le même ; utiliser un code minimaliste accessible aux débutants. Les bibliothèques Qt présentent un grand intérêt lorsque l'on souhaite produire du code indépendant du système d'exploitation sous-jacent. On aborde aussi la programmation orientée objet avec ces bibliothèques.

Table des matières

1. Copyright et Licence

1.1. Meta-information

Cet article est écrit avec DocBook XML sur un système Debian GNU/Linux. Il est disponible en version imprimable au format PDF : `socket-qt.pdf`.

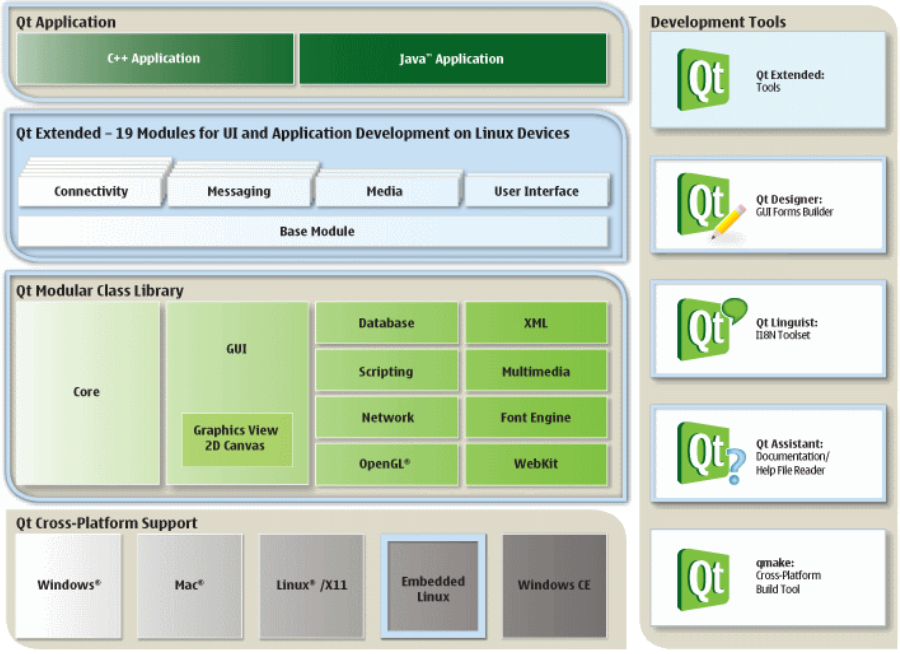
2. Contexte de développement

Comme ce support est la suite de celui sur le Langage C, on reprend le même découpage en deux programmes distincts : un serveur et un client qui échangent des chaînes de caractères. À la différence du support précédent, on ne présente pas le code des deux programmes. Dans le but de limiter le volume du document, on introduit uniquement le code de la partie serveur qui reçoit le message, le traite et le réexpédie au client. Pour les tests, on peut très bien réutiliser le programme client déjà développé en Langage C : Code du programme udp-client.c

Le principe d'illustration des communications réseau reste le même : le client ou talker émet un message que le serveur ou listener traite et retransmet vers le client. Le traitement est toujours aussi minimaliste ; le serveur convertit la chaîne de caractères en majuscules.

2.1. Bibliothèques Qt

Le diagramme ci-dessous présente l'architecture des bibliothèques Qt. Il met en évidence l'indépendance entre les développements et les systèmes d'exploitation cibles.



Dans notre contexte, on utilise le Langage C++ et l'environnement de développement intégré QtCreator. Cet environnement comprend les bibliothèques Qt réseau dont nous avons besoin.

On se propose de développer le programme serveur en deux temps.

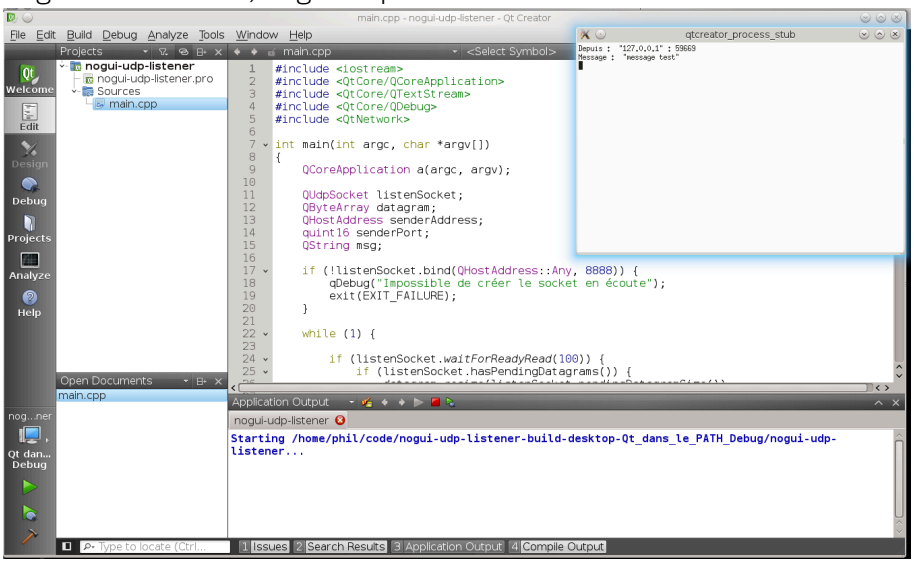
- Une version sans graphisme mettant en évidence les appels de bibliothèques propres à l'utilisation des sockets.

- Une version avec une fenêtre graphique dans laquelle on affiche les messages reçus depuis un programme client ainsi que l'adresse IP et le numéro de port utilisés par ce programme client.

2.2. Instructions d'exécution

On exécute le programme client déjà développé en Langage C dans un Shell alors que le programme serveur est géré directement par l'environnement QtCreator. On peut exécuter ces deux programmes sur le même hôte en utilisant l'interface de boucle locale pour les communications réseau.

Le programme serveur, nogui-udp-listener



Note

Déplacer le pointeur de la souris sur l'image, clicker sur le bouton droit et lancer «Afficher l'image» pour lire les messages.

C'est dans la partie inférieure droite de la copie d'écran que l'on retrouve en rouge les messages échangés entre les programmes client et serveur. Le code utilisé ici correspond à la version sans graphisme du programme serveur.

Le programme client, udp-talker.o

```
$ ./udp-talker.o
Entrez le nom du serveur ou son adresse IP :
127.0.0.1
Entrez le numéro de port du serveur :
8888

Entrez quelques caractères au clavier.
Le serveur les modifiera et les renverra.
Pour sortir, entrez une ligne avec le caractère '.' uniquement.
Si une ligne dépasse 100 caractères,
seuls les 100 premiers caractères seront utilisés.

Saisie du message :
        texte de test avec tabulation et espaces
Message traité : TEXTE DE TEST AVEC TABULATION ET ESPACES
Saisie du message :
_exit_
Message traité : _EXIT_
Saisie du message :
serveur arrêté
Pas de réponse dans la seconde.
Saisie du message :
.
```

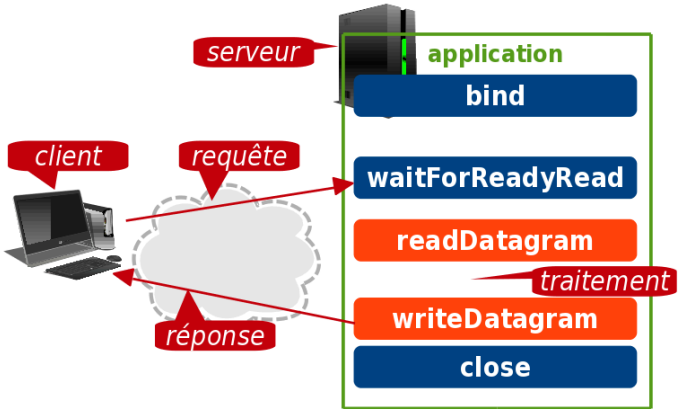
Lorsque le programme serveur est en cours d'exécution, il est possible de visualiser la correspondance entre le processus en cours d'exécution et le numéro de port en écoute à l'aide de la commande netstat.

```
$ netstat -aup | grep -e Proto -e 8888
(Tous les processus ne peuvent être identifiés, les infos sur les processus
non possédés ne seront pas affichées, vous devez être root pour les voir toutes.)
Proto Recv-Q Send-Q Adresse locale    Adresse distante  Etat  PID/Program name
udp        0      0 *:8888            *:*                     3321/QtCreator_UDP
```

Dans l'exemple ci-dessus, le numéro de port `8888` apparaît dans la colonne `Adresse locale` et processus numéro `3321` correspond bien au programme `QtCreator_UDP_receiver` dans la colonne `PID/Program name`.

2.3. Bibliothèques utilisées

Les bibliothèques standards du Langage C utilisées par le programme client sont présentées dans le document Initiation au développement C sur les sockets. On ne s'intéresse ici qu'à l'utilisation de la «brique» réseau des bibliothèques Qt.



`bind`

Cette méthode de la classe `QUdpSocket` assure la liaison d'un socket avec les adresses et le numéro de port spécifiés.

`waitForReadyRead`

C'est une fonction bloquante qui attend que de nouvelles données soient disponibles en lecture et que le signal `readyRead()` ait été émis. Le blocage expire après `msecs` millisecondes. La valeur par défaut est 30000 millisecondes.

La fonction retourne la valeur booléenne vrai si le signal `readyRead()` est émis.

`readDatagram`

Cette fonction assure la réception d'un datagramme de taille inférieure à `maxSize` octets et le stocke dans le tableau `data`. L'adresse et le numéro de port de l'émetteur sont stockées dans les pointeurs `*address` et `*port` (à moins que les pointeurs aient une valeur nulle).

Si `maxSize` est trop petit, le reste du datagramme est perdu. Pour éviter les pertes de données, on fait appel à `pendingDatagramSize()` pour déterminer la taille du datagramme en attente avant d'essayer de le lire. Si `maxSize` vaut 0, le datagramme sera éliminé.

`writeDatagram`

Cette fonction assure l'émission d'un datagramme de taille `size` à destination de l'adresse `address` et du port `port`. Elle renvoie le nombre d'octets émis avec succès ou -1 en cas d'erreur.

Les datagrammes sont toujours écrits en un bloc. La taille maximum d'un datagramme dépend beaucoup du système utilisé. Elle peut atteindre 8192 octets. Si le datagramme est trop grand, la fonction renvoie -1 et la fonction `error()` renvoie `DatagramTooLargeError`. Il est déconseillé d'émettre des datagrammes de plus de 512 octets est en général. Même s'ils sont émis avec succès, ils seront probablement fragmentés au niveau de la couche IP avant d'arriver à leur destination finale.

3. Programme serveur UDP en mode console

Avec l'environnement intégré de développement QtCreator, il est possible de créer une application en mode console.

À partir de l'écran d'accueil, il faut créer un nouveau projet et choisir Qt Console Application dans la catégorie Application.

L'intérêt de cette étape est de produire un fichier source unique de taille réduite dans lequel on trouve l'ensemble des appels de bibliothèques Qt.

3.1. Utilisation des sockets avec le serveur UDP console

Avec les bibliothèques Qt, on retrouve les éléments classiques de représentation des communications réseau.

```
QUdpSocket listenSocket;
QByteArray datagram;
QHostAddress senderAddress;
quint16 senderPort;
```

listenSocket

Objet de la classe `QUdpSocket` constituant le canal de communication entre le sous-système réseau du noyau du système d'exploitation et l'application.

datagram

Objet de la classe `QByteArray` contenant les données vues de la couche application de la modélisation réseau.

senderAddress

Objet de la classe `QHostAddress` contenant la représentation de l'adresse de l'hôte qui a émis le datagramme reçu par le serveur.

senderPort

Entier en format court contenant le numéro de port utilisé par l'hôte qui a émis le datagramme reçu par le serveur.

```
<snipped/>
if (!listenSocket.bind(QHostAddress::Any❶, 8888❷)) {
  qDebug("Impossible de créer le socket en écoute");
  exit(EXIT_FAILURE)❸;
}
```

❶ `QHostAddress::Any` spécifie que le programme est en écoute sur toutes les adresses IP des interfaces de l'hôte.
❷ L'application est en écoute sur le port numéro `8888` uniquement.
❸ Le le programme s'interrompt en cas d'erreur. On reprend ici le même principe que dans les autres programmes écrits en Langage C.

Une fois la liaison en place, le programme attend les datagrammes provenant du client.

```
<snipped/>
if (listenSocket.waitForReadyRead(100))❶ {
  if (listenSocket.hasPendingDatagrams())❷) {
    datagram.resize(listenSocket.pendingDatagramSize());❸
    if (listenSocket.readDatagram(datagram.data(), datagram.size(),
                                  &senderAddress, &senderPort) == -1) {
      listenSocket.close();
      exit(EXIT_FAILURE);
    }
    msg = datagram.data();❹
```

❶ La méthode `waitForReadyRead(100)` bloque l'exécution du programme pendant 100ms, le temps de détecter un évènement de réception de données sur le socket en écoute.
❷ Si un datagramme est en attente, on procède à sa lecture.
❸ On redimensionne le tableau des données en fonction de la taille du datagramme reçu.
❹ On stocke les données du datagramme dans la chaîne de caractères `msg`.

Enfin, l'émission de datagramme du serveur vers le client utilise la fonction `writeDatagram` avec un jeu de paramètres identique à l'opération de lecture.

3.2. Code source complet

```
#include <iostream>
#include <QtCore/QCoreApplication>
#include <QtCore/QTextStream>
#include <QtCore/QDebug>
#include <QtNetwork>

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);

    QUdpSocket listenSocket;
    QByteArray datagram;
```

```
    QHostAddress senderAddress;
    quint16 senderPort;
    QString msg;

    if (!listenSocket.bind(QHostAddress::Any, 8888)) {
        qDebug("Impossible de créer le socket en écoute");
        exit(EXIT_FAILURE);
    }

    while (1) {

        if (listenSocket.waitForReadyRead(100)) {
            if (listenSocket.hasPendingDatagrams()) {
                datagram.resize(listenSocket.pendingDatagramSize());
                if (listenSocket.readDatagram(datagram.data(), datagram.size(),
                                              &senderAddress, &senderPort) == -1) {
                    listenSocket.close();
                    exit(EXIT_FAILURE);
                }
                msg = datagram.data();

                qDebug() << "Depuis : " << senderAddress.toString() << ':' << senderPort;
                qDebug() << "Message : " << msg;

                msg = msg.toUpper();
                datagram.clear();
                datagram.append(msg);

                if (listenSocket.writeDatagram(datagram, senderAddress, senderPort) == -1) {
                    qDebug("Émission du message modifié impossible");
                    listenSocket.close();
                    exit(EXIT_FAILURE);
                }
            }

            if (msg  == "_EXIT_") {
                listenSocket.close();
                exit(EXIT_SUCCESS);
            }
        }
    }

    listenSocket.close();

    return a.exec();
}
```

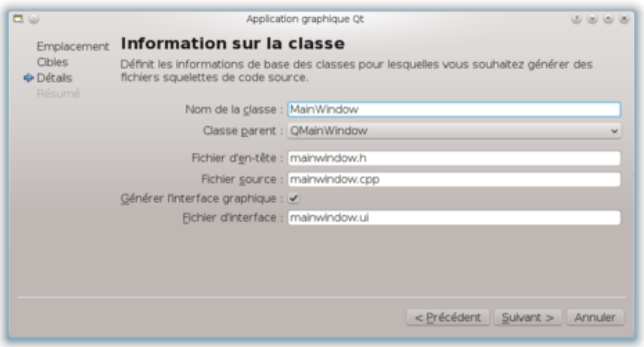

Note

Le code complet du projet est disponible via le fichier `nogui-udp-listener.tar.gz`.

4. Programme serveur UDP en mode graphique

Dans cette section, on crée une nouvelle application graphique avec l'environnement intégré de développement QtCreator. À partir de l'écran d'accueil, il faut créer un nouveau projet et choisir Application graphique Qt dans la catégorie Projet Qt Widget.

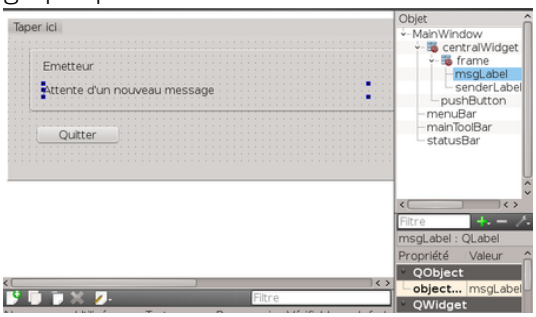
Relativement à la section précédente dans laquelle on utilisait un fichier source unique de taille limitée, on se trouve face à un pré découpage des fichiers défini par défaut. Avec cette organisation des sources, la gestion de la fenêtre graphique est séparée du code proprement dit. Tous les attributs de la fenêtre utilisée sont décrits dans le fichier `mainwindow.ui`.



4.1. Utilisation des sockets avec le serveur UDP graphique

Puisque cette section est consacrée à une application graphique, on commence par décrire le volet «interface utilisateur».

Une fois le projet ouvert, on sélectionne l'onglet Formulaires puis le fichier de description de la fenêtre graphique `mainwindow.ui`.



> **Note**
>
> Déplacer le pointeur de la souris sur l'image, cliquer sur le bouton droit et lancer «Afficher l'image» pour lire les messages.

Dans cette fenêtre, on implante trois objets.

pushButton

L'objet bouton, dont on a changé le texte affiché, sert à quitter le programme. Le code correspondant à l'utilisation du bouton se trouve dans le fichier `mainwindow.cpp`.

msgLabel

Cet objet champ de texte est utilisé pour l'affichage de la chaîne de caractères `msg`. C'est lors de la réception d'un nouveau message que les caractères de ce champ sont modifiés.

senderLabel

Cet objet champ de texte est utilisé pour l'affichage de l'adresse IP et le numéro de port de l'émetteur du message. Comme dans le cas précédent, il faut qu'un nouveau message ait été reçu et que l'émetteur soit identifié pour que ce champ soit modifié.

En revenant au mode Éditer, on peut parcourir les différents fichiers sources du projet.

`main.cpp`

Le programme principal est complété par défaut lors de la création d'un nouveau projet. À notre niveau, il n'est pas nécessaire de le modifier. Son seul travail est d'afficher la fenêtre `w` de la classe `MainWindow` avant d'entrer dans la boucle d'évènements (event loop).

`mainwindow.h`

Ce fichier en-tête contient la déclaration de la classe `MainWindow`. C'est à ce niveau que l'on débute l'édition de code avec l'ajout d'une méthode et de plusieurs membres privés. Le mot clé private spécifie que seuls les objets de la classe ont accès à ces ressources.

La méthode `processPendingDatagrams()` contient tous les traitements de réception, de transformation et de ré émission des messages. Cette méthode utilise les membres déclarés dans la classe.

On retrouve ici la liste des variables du programme de la section précédente en plus du pointeur de fenêtre `ui` (user interface) et des deux champs de texte présentés plus haut.

```
private:
  Ui::MainWindow *ui;
  QLabel *senderLabel, *msgLabel;
  QUdpSocket *listenSocket;
  QHostAddress senderAddress;
  quint16 senderPort;
  QString msg;
```

`mainwindow.cpp`

Après les déclarations des membres et des méthodes de la classe `MainWindow`, ce fichier contient le code des méthodes. Les points importants ici sont le traitement des évènements et le traitement des datagrammes réseau.

4.2. Traitement des évènements et des datagrammes

Dans ce programme, deux types d'évènements sont scrutés : l'arrivée d'un nouveau datagramme et le clic sur le bouton Quitter. Le code correspondant à la scrutation des évènements est décrit dans la méthode `MainWindow` du fichier `mainwindow.cpp`.

```
<snipped/>
  QObject::connect(listenSocket, SIGNAL(readyRead()),❶
                   this, SLOT(processPendingDatagrams()));❷

  QObject::connect(ui->pushButton, SIGNAL(clicked()),
                   this, SLOT(close()));❸
```

❶ L'appel de la méthode `connect()` de la classe `QObject` crée une connexion entre le socket `listenSocket` et le signal issu de la méthode `readyRead()`.

❷ Dès qu'un signal est reçu par la méthode `connect()`, un datagramme est en attente de lecture. Cette lecture et le traitement associé sont effectués à l'aide du sous-programme `processPendingDatagrams()` dont le code est donné dans le même fichier source.

❸ Ici, l'appel à la méthode `connect()` de la classe `QObject` crée une connexion entre l'objet `pushButton` et le signal issu de la méthode `clicked()`. Dès que ce signal est reçu, la méthode `close()` entraîne la fin d'exécution du programme.

Pour le traitement des datagrammes, c'est le code du sous-programme `processPendingDatagrams()` qui assure la lecture, le passage en majuscules des caractères de la chaîne et la réémission. L'opération de transformation des caractères en majuscules n'est qu'une illustration d'un traitement possible. Comme la plupart des protocoles de l'Internet sont basés sur l'échange de messages sous forme de chaînes de caractères, on utilise ici une forme minimaliste de traitement par requête - réponse.

```
<snipped/>
void MainWindow::processPendingDatagrams()
{

  while (listenSocket->hasPendingDatagrams())  {❶

    QByteArray datagram;

    datagram.resize(listenSocket->pendingDatagramSize());
    if (listenSocket->readDatagram(datagram.data(), datagram.size(),
                                   &senderAddress, &senderPort) == -1) {❷
       listenSocket->close();
       exit(EXIT_FAILURE);
    }

    msg = datagram.data();
    qDebug() << "Depuis : " << senderAddress.toString() << ':' << senderPort;
    qDebug() << Message : " << msg;

    ui->senderLabel->setText(tr("Depuis : %1:%2")
                             . arg(senderAddress.toString())
        . arg(senderPort));❸
    ui->msgLabel->setText(tr("Message : \"%1\"")
                             . arg(msg));❹

    msg = msg.toUpper();

    datagram.clear();
    datagram.append(msg);

    if (listenSocket->writeDatagram(datagram, senderAddress, senderPort) == -1) {❺
       qDebug("Émission du message modifié impossible");
       listenSocket->close();
       exit(EXIT_FAILURE);
       }
    }
}
```

On retrouve les éléments du précédent programme en mode console dans le code ci-dessus.

❶ La méthode `hasPendingDatagrams()` renvoie la valeur booléenne `vrai` tant qu'un datagramme est présent dans la file d'attente.
❷ La méthode `readDatagram()` collecte les données du datagramme ainsi que sa provenance en identifiant l'adresse de l'émetteur et le numéro de port source utilisé.
❸ La méthode `setText()` est appelée pour changer le texte référencé par l'étiquette `senderLabel` dans la fenêtre `MainWindow`. On fait apparaître l'adresse IP et le numéro de port de l'émetteur du message.
❹ La méthode `setText()` est à nouveau appelée pour changer le texte référencé par l'étiquette `msgLabel` dans la fenêtre `MainWindow`. On fait apparaître ici le contenu de la chaîne de caractères reçue.
❺ Une fois la chaîne de caractères transformée, elle est renvoyée à l'émetteur à l'aide de la méthode `writeDatagram()`.

Note

Le code complet du projet est disponible via le fichier `gui-udp-listener.tar.gz`.

5. Documents de référence

Qt Network Programming

Qt Network Programming : page d'entrée dans la documentation en ligne des bibliothèques Qt sur les communications réseau.

Modélisations réseau

Modélisations réseau : présentation et comparaison des modélisations OSI et Internet.

Adressage IPv4

Adressage IPv4 : support complet sur l'adressage du protocole de couche réseau de l'Internet (IP).

Configuration d'une interface réseau

Configuration d'une interface de réseau local : support sur la configuration des interfaces réseau. Il permet notamment de relever les adresses IP des hôtes en communication.