Travaux Pratique programmation réseau avec QT5

1. OBJECTIF

Utilisation des sockets avec la bibliothèque QT5

Mettre en pratique les sockets en mode événementiel. Utiliser cette propriété pour réaliser un serveur multi-Clients.

Points techniques abordés :

- Codage de l'information
- Programmation sockets avec QT5
- Communication réseau

2. CONDITIONS DE RÉALISATION

Travail individuel sur micro-ordinateurs avec QT Creator

Un serveur et un client sont disponibles pour la mise au point.

Compte rendu et code à rendre.

3. RESSOURCES

Les classes « socket » dans la technologie QT, consulter le site http://doc.qt.io/qt-5/qtnetwork-index.html et plus particulièrement les classes, ***QTcpSocket, QTcpServeur.***

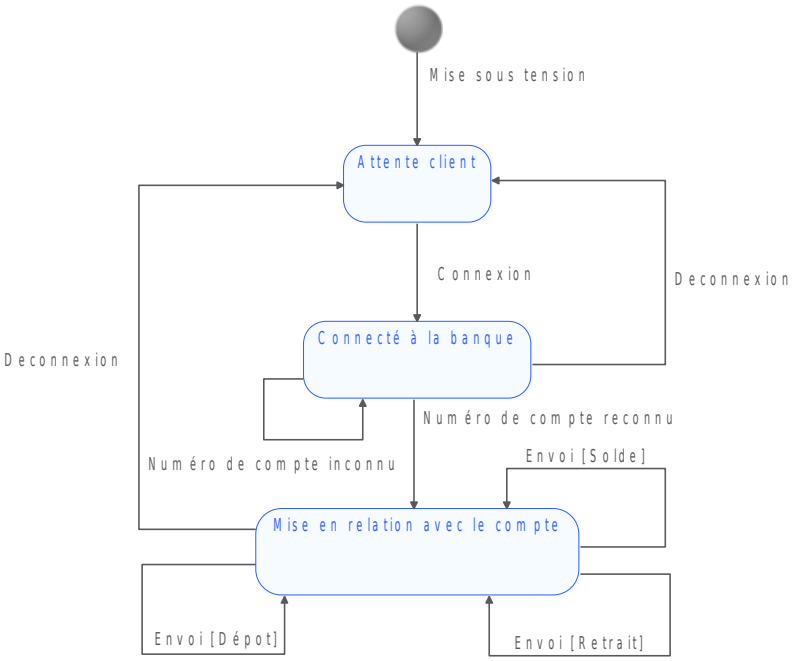
4. LE BESOIN

Les distributeurs automatiques de billets (DAB) permettent de se connecter à la banque. Ils offrent la possibilité d'interroger un compte pour en connaître le solde, de déposer de l'argent et d'en retirer. C'est ces fonctionnalités que nous allons étudier dans un premier temps côté client (sur le DAB) puis côté serveur (à la banque).

5. BANQUE ~ CÔTÉ SERVEUR

5.1. Rappels

Fonctionnalités attendues du distributeur automatique de billets :



Comme l'indique le diagramme à états ci-dessus, c'est seulement lorsque le client est connecté à la banque et que son numéro de compte est reconnu que les opérations vers la banque peuvent être réalisées .

5.1.1. Format des trames client -> serveur

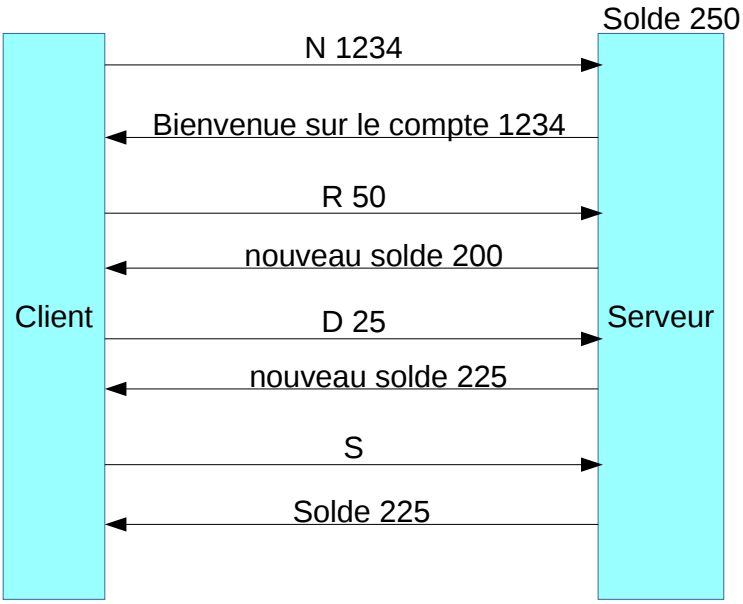
Taille des données	Commande	Valeur
De type quint16	Le caractère 'N' sous la forme d'un Qchar	Un entier correspondant au numéro de compte (type int)

Taille des données	Opération	Montant
De type quint16	Le caractère 'R' sous la forme d'un Qchar	Un réel (type float)
De type quint16	Le caractère 'D' sous la forme d'un Qchar	Un réel (type float)
De type quint16	Le caractère 'S' sous la forme d'un Qchar	Sans objet

5.1.2. Format des trames Serveur -> client

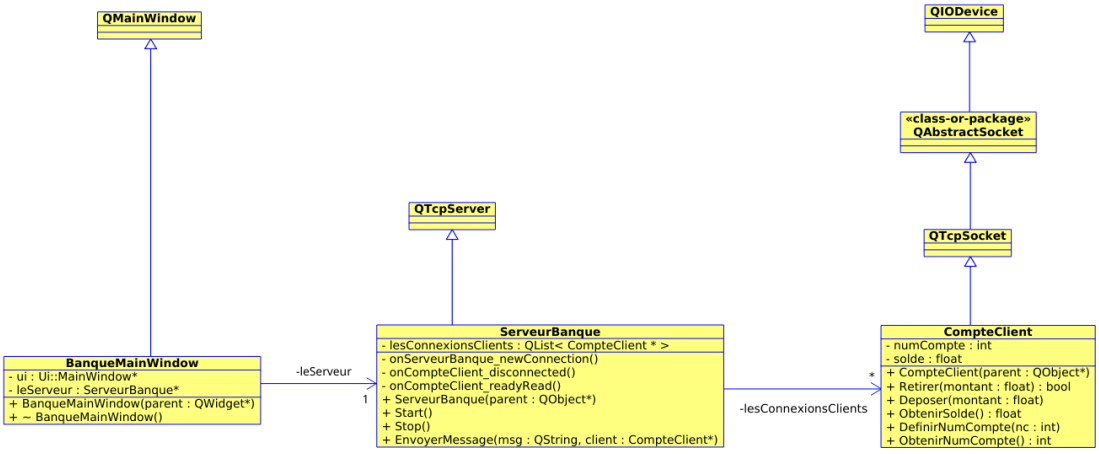
Taille des données	Commande
De type quint16	Une chaîne de caractères sous la forme d'une QString.

Exemple de communication client-serveur:



5.2. Création du projet

Créez le projet ***ServeurBanque*** de type Application graphique en C++ sous QT5 avec QT Creator. La classe principale se nomme ***BanqueMainWindow*** elle hérite de ***QMainWindow*** comme le montre le diagramme de classes partiel ci-dessous.



L'IHM de la banque comporte juste un bouton Quitter qui ferme l'application.

1. Réalisez ce traitement.

5.2.1. Réalisation de la classe ServeurBanque (1ère Partie)

Pour la gestion du serveur de la banque, on propose la classe ***ServeurBanque*** qui hérite de la classe ***QTcpServer*** comme le montre le diagramme de classes.

Le constructeur de la classe ***ServeurBanque*** prend comme paramètre d'entrée un pointeur sur un ***QObject*** par défaut initialisé à 0 qui sera transmis au constructeur de la classe de base.

Il réalise également la liaison entre le signal indiquant la présence de la connexion d'un nouveau client au slot correspondant.

En cas d'erreur une boîte de dialogue de type **QMessageBox** apparaît signalant que la liaison n'a pu avoir lieu.

2. *Codez le constructeur de la classe* ***ServeurBanque****.*

La méthode ***ServeurBanque::Start()*** lance, si cela est possible, l'écoute du socket sur toutes les interfaces réseau de l'ordinateur sur le port 8888.

Si l'écoute n'est pas possible, une boîte de message signale l'erreur et ferme le socket Serveur.

3. *Codez la méthode* ***Start*** *de la classe* ***ServeurBanque****.*

La méthode ***ServeurBanque::Stop()*** est chargé de fermé le socket serveur.

Cette méthode devra être appelée dans le destructeur de la classe ***BanqueMainWindow.***

4. *Codez la méthode* ***Stop*** *de la classe* ***ServeurBanque****.*

Après avoir instancié l'attribut ***leServeur*** dans la classe ***BanqueMainWindow*** comme le préconise le diagramme de classes, la méthode ***Start()*** de la classe ***ServeurBanque*** est appelée.

5. *Complétez le code de la classe* ***BanqueMainWindow*** *comme décrit ci-dessus.*

Pour la testé:

- Lancez une première fois l'application.
- Ne la fermez pas
- Relancez une deuxième fois l'application, une boîte indiquant l'erreur de l'écoute du socket doit apparaître.

5.2.2. Réalisation de la classe CompteClient

6. *Créez la classe* ***CompteClient*** *comme le montre le diagramme de classes.*

Le constructeur de la classe ***CompteClient*** prend comme paramètre d'entrée un pointeur sur un ***QObject*** par défaut initialisé à 0 qui sera transmis au constructeur de la classe de base.

7. Codez le constructeur.

La méthode ***CompteClient ::DefinirNumCompte*** prend un entier en paramètre d'entrée qui initialise l'attribut numCompte et fixe le solde à 200 euros.

8. *Codez* ***DefinirNumCompte***

La méthode ***CompteClient ::Deposer*** prend un réel en entrée et ajoute sa valeur au solde du client.

9. *Codez* ***Deposer***

La méthode ***CompteClient ::Retirer*** prend un réel en entrée si cela est possible la valeur est retranchée et la méthode retourne vrai. Sinon le paramètre de retour prend la valeur faux, la banque n'accepte pas de découvert.

10.*Codez* ***Retirer***

La méthode ***CompteClient ::ObtenirSolde*** retourne la valeur du solde du compte du client.

11.*Codez* ***ObtenirSolde***

Enfin, la méthode ***CompteClient ::ObtenirNumCompte*** retourne le numéro du compte client.

12.*Codez* ***ObtenirNumCompte***

5.2.3. Réalisation de la classe ServeurBanque (2nd partie)

Le slot ***ServeurBanque::onServeurBanque_newConnection*** a pour objectif d'accepter les différents sockets client qui tentent de se connecter au serveur.

L'algorithme de haut niveau ci-dessous présente le traitement à effectuer hors-mis les traitements d'erreurs qui devront être ajoutés.

```
Tant qu'il y a des connexions en attente
        1. client ← la connexion du client suivant

        2. Association des signaux indiquant des données à lire
           et la déconnexion en provenance du client aux deux
           autres slots de la classe ServeurBanque.

        3. Ajout du client dans la QList représentant les
           connexions des différents clients.

        4. Envoi d'un message au client lui demandant son numéro
           de compte.

Fin tantQue.
```

13.*Codez* ***onServeurBanque_newConnection***

Comme son nom l'indique, la méthode ***ServeurBanque::EnvoyezMessage*** est chargée d'envoyer un message sous la forme d'un ***QString*** passé en paramètre au client dont le pointeur est également passé en paramètre.

14.*Réalisez le code de la méthode* ***ServeurBanque::EnvoyezMessage.***

Le code du slot ***ServeurBanque::onCompteClient_disconnected*** est donné ci-après:

```
void ServeurBanque::onCompteClient_disconnected()
{
    CompteClient *client=(CompteClient*)sender();
    if (!client)
    {
      QMessageBox msg;
      msg.setText("erreur deconnexion : "+client->errorString());
      msg.exec();
    }
    else
    {
      lesConnexionsClients.removeOne(client);
      client->deleteLater();
    }
}
```

La méthode ***QObject::sender*** retourne le pointeur sur l'objet qui a envoyé le signal. La méthode ***QList::removeOne*** supprime l'objet passé en paramètre de la liste.

15.En vous inspirant du code précédent, réalisez le slot ***ServeurBanque::slotReadyRead.***

Ce dernier est appelé lorsque des données sont à lire en provenance d'un client.

Il est dans un premier temps nécessaire de rechercher de quel client il s'agit.

Vous vous attacherez ici à répondre aux besoins du client décrit dans la première partie.

16.Comme Bonus, réalisez le code nécessaire pour qu'un client retrouve le solde qu'il a laissé lors de sa dernière connexion (voir QFile, QDir...)**.**