Travaux Dirigés n°2 programmation réseau avec Qt

Côté Serveur

1. OBJECTIF

Utilisation des sockets avec la bibliothèque Qtnetwork

Comprendre le fonctionnement des sockets en mode événementiel. En effet, les logiciels actuels (visual studio, Qt…) utilisent le système d'exploitation pour répondre aux sollicitations externes (les événements). Lorsqu'un événement survient, le système appelle une fonction prévue et déclarée à cet effet.

Points techniques abordés :

- Protocole applicatif maison
- Programmation sockets avec le module **QtNetwork**
- Communication réseau protocole transport **TCP**

2. CONDITIONS DE RÉALISATION

Travail sur micro-ordinateurs avec Qt Creator

Un client est disponible

3. COMPTE RENDU RAPPEL

Toutes les questions et/ou points techniques abordés dans le TD devront apparaître dans le compte rendu. Celui-ci sera réalisé à l'aide de LibreOffice Writter.

4. RESSOURCES

Les classes du module QtNetwork, consulter le site http://doc.qt.io/qt-5/qtnetwork-index.html et plus particulièrement la classe, ***QTcpServeur***.

5. LE BESOIN

Les techniciens réseau sont souvent appelés pour une défaillance sur un poste informatique. Avant de se déplacer, il leurs est intéressant de connaître certaines caractéristiques de la machine. Pour ce faire, un petit programme

« Serveur » implanté sur chaque machine leurs permet d'interroger la machine distante pour connaître son système d'exploitation, sa version, le type de microprocesseur l'utilisateur connecté... Le travail proposé ici permet de définir le protocole applicatif utilisé et de coder le serveur.

5.1. Analyse et conception

Pour Windows, le serveur est capable de fournir les informations de la machine à l'administrateur client distant. Le serveur utilise pour cela les variables d'environnement disponibles sur le poste à interroger.

Cette liste est donnée de manière non exhaustive ci-après :

Variables disponibles sous Windows :

- **COMPUTERNAME** : nom de l'ordinateur.
- **HOMEDRIVE** : disque local primaire (partition système).
- **HOMEPATH** : dossier par défaut pour les utilisateurs.
- **LOGONSERVER** : nom du serveur de domaine.
- **NUMBER_OF_PROCESSORS** : nombre de processeurs installés.
- **OS** : nom du système d'exploitation.
- **PATHEXT** : liste des extensions de fichier reconnus comme des exécutables.
- **PROCESSOR_ARCHITECTURE** : type de processeur installé.
- **PROCESSOR_IDENTIFIER** : identification du processeur (type, modèle, etc.).
- **PROCESSOR_LEVEL** : niveau du processeur.
- **PROCESSOR_REVISION** : révision du processeur.
- **SystemDrive** : disque local sur lequel le système réside.
- **SystemRoot** : chemin du système (égal à **Windir**).
- **USERDOMAIN** : nom du domaine sous lequel l'utilisateur s'est connecté.
- **USERNAME** : nom de l'utilisateur courant.
- **USERPROFILE** : chemin du profil de l'utilisateur courant.

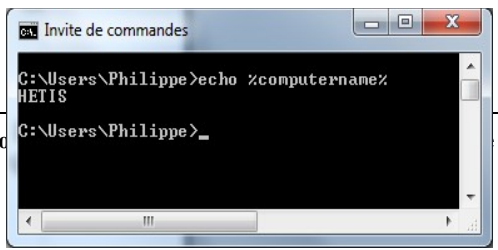
En associant une lettre à chaque requête, le serveur dispose d'une liste de commandes auxquelles il peut donner une réponse.

Demande	*Requête*	*Réponse attendue*
Nom de l'utilisateur	'u'	Le nom de l'utilisateur connecté
Nom de la machine	'c'	Le nom de la machine
Système d'exploitation	'o'	Le type de système d'exploitation
L'architecture du processeur	'a'	Le type de processeur x86 ou amd64 par exemple

Pour une lettre qui ne figure pas dans la liste, la réponse sera « **requête inconnue** »

Avec la ligne de commande, il est facile de vérifier leur valeur en tapant par exemple : ***echo %USERNAME%*** suivit de la touche Entrée. C'est sur ce principe que le serveur va fonctionner.

Vous pouvez essayer… ça marche !!!



Pour Linux, il est possible d'utiliser également les variables d'environnement.

Exemple d'utilisation : ***echo $USERNAME***

Pour avoir la liste des variables possibles il suffit de faire :

echo $<TAB><TAB>

La fonction ***QByteArray qgetenv(const char *varName)*** retourne sous la forme d'un tableau de caractère le résultat de la commande ***echo*** quelque soit le système d'exploitation.

Exemple :

```
QByteArray utilisateur = qgetenv("USERNAME");
ui->lineEditUtilisateur->setText(utilisateur.data());
```

Malheureusement, cette solution ne fonctionne pas pour toutes les variables

d'environnement !!

Malheureusement, cette solution ne fonctionne pas pour toutes les variables d'environnement. Il est parfois nécessaire de passer par un ***QProcess*** et d'exécuter un commande SHELL suivant l'environnement.

Exemple : d'utilisation de ***QProcess***, le processus est créé le signal indiquant que des données sont présentes sur la sortie standard est connecté à un SLOT chargé du traitement. Puis le processus est lancé avec la commande SHELL uname.

```
process = new QProcess;
if(!connect(process,SIGNAL(readyReadStandardOutput()),this,SLOT(On_readyReadStandardOutput())))
    qDebug() << "Erreur connect SIGNAL readyReadStandardOutput au SLOT On_readyReadStandardOutput";

process->start("uname");
```

Le résultat est obtenu est obtenu dans le slot comme le montre l'extrait de code suivant :

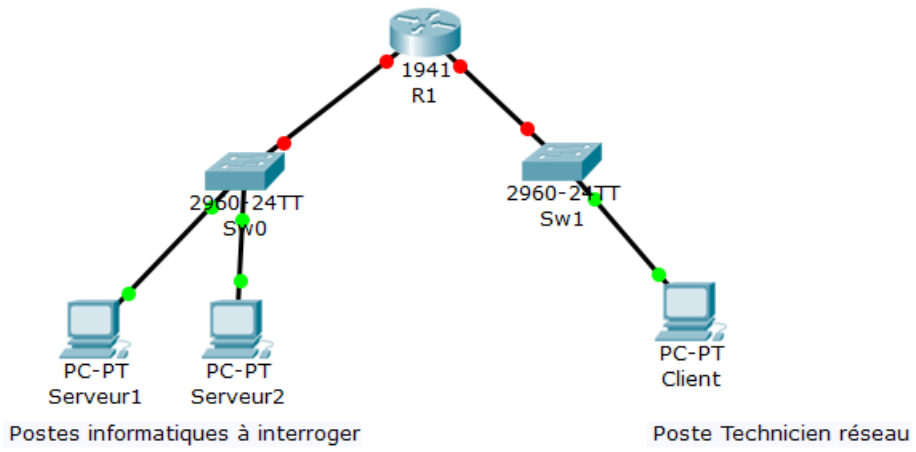
```
void ServeurInfoOrdi::On_readyReadStandardOutput()
{
    QString resultat;
    if(process->bytesAvailable())
    {
        resultat = process->readAll();
        ui->lineEditOs->setText(resultat);
    }
}
```

L'exemple ci-dessus doit être adapté si des paramètres doivent être fournis à la commande.

Recherchez les différentes surcharges de la méthode ***Qprocess::start()***.

Une autre solution est d'utiliser la classe **QsysInfo**

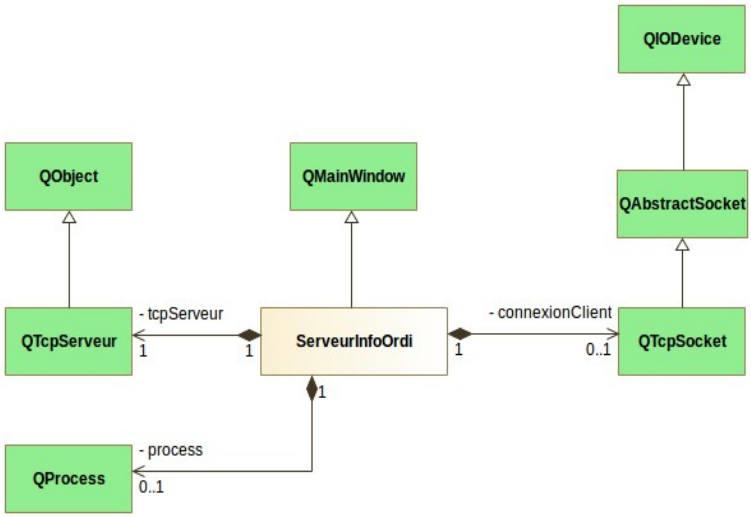
5.2. Mise en situation



6. RÉALISATION DU SERVEUR

6.1. Création du projet

Créez un projet de type Application graphique en C++ sous Qt5 avec QtCreator. La classe principale ce nomme ***ServeurInfoOrdi,*** elle hérite de ***QMainWindow*** comme le montre le diagramme de classes partiel ci-dessous. C'est la seule classe ou vous aurez à intervenir.

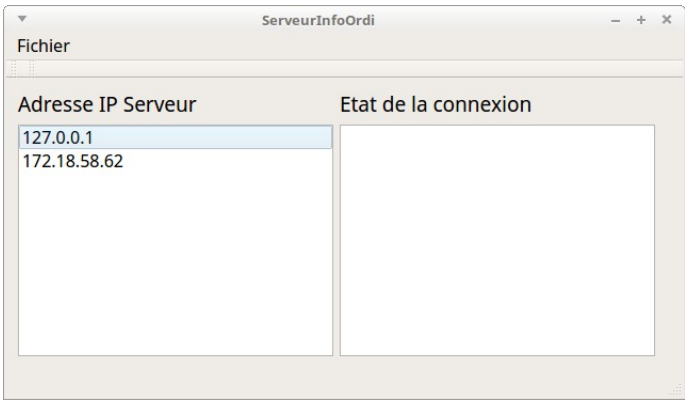


De même, ajoutez les attributs réalisant les liaisons entre la classe principale et les autres.

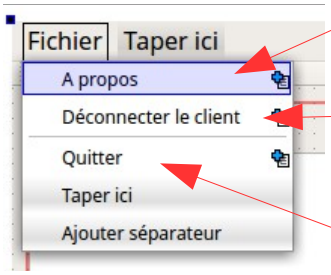
L'instance de la classe ***QTcpServeur*** sera chargée de la gestion de la socket d'écoute du serveur. L'instance de la classe ***QTcpSocket*** sera chargée de la communication avec le client connecté.

6.2. Création de l'IHM

L'interface du serveur aura l'aspect suivant :



Objet	Classe
ServeurInfoOrdi	QMainWindow
centralWidget	QWidget
horizontalLayout	QHBoxLayout
verticalLayout	QVBoxLayout
label_2	QLabel
listWidgetIp	QListWidget
verticalLayout_2	QVBoxLayout
label	QLabel
listWidgetEtat	QListWidget
labelPort	QLabel
menuBar	QMenuBar
menuFichier	QMenu
actionQuitter	QAction
actionA_propos	QAction
actionD_c...le_client	QAction
mainToolBar	QToolBar
statusBar	QStatusBar
toolBar	QToolBar



Affiche le numéro du port dans la barre d'état

Déconnecte le client connecté au serveur

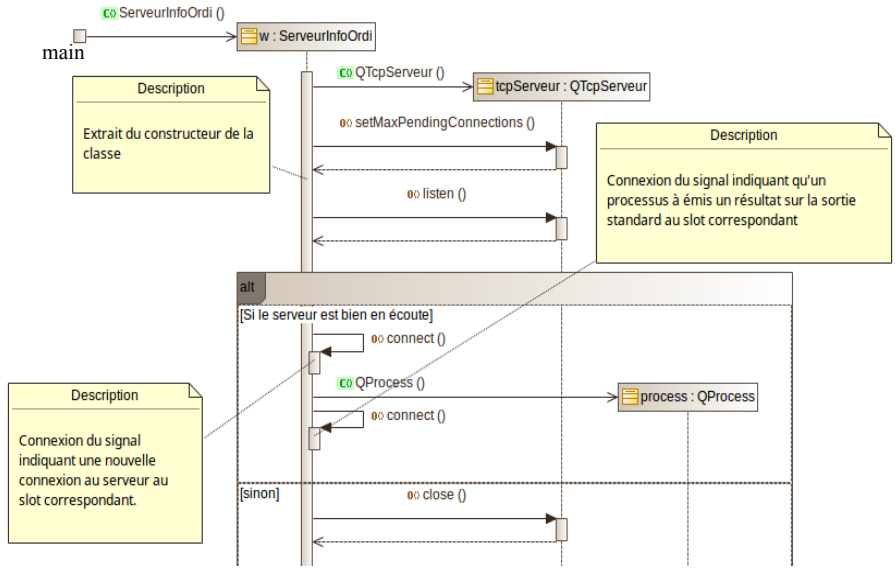
Quitte l'application

Nommez chaque ***Widget*** de manière à ce qu'il conserve son type de Widget et le nom auquel il se rapporte, comme le montre l'exemple.

Un menu et une barre d'état complète cette interface, comme l'indique la figure ci-dessus.

6.3. Utilisation de la classe QTcpServeur

Le diagramme de séquence ci-après représente le constructeur de la classe ***ServeurInfoOrdi***. Complétez le codage de ce constructeur.



A l'aide de la commande ***netstat*** déterminez un port disponible dans la plage destinée à un usage libre.

Par défaut, le constructeur de la classe QTcpServer permet l'écoute sur toutes les interfaces réseau utilisables sur le système et le numéro de port est choisi arbitrairement par le système. Déterminez les paramètres de la méthode ***QtcpServeur::listen()*** pour le port que vous avez choisi. Il permet également l'affichage des adresse IP du serveur et de son port d'écoute dans la barre d'état.

L'ensemble des adresses IP disponible sur le système est mis dans un conteneur type QList avec la ligne suivante.

```
QList <QHostAddress> ipAddressesList = QNetworkInterface::allAdresses() ;
```

L'adresse IP affichée dans le QListWidget sera la première adresse de type Ipv4 différente de *QHostAdress::LocalHost* dans le conteneur ***ipAddressesList***. Si aucune adresse n'est disponible, l'adresse affichée sera celle de *QhostAdress::LocalHost*.

```
QList<QHostAddress> listeAdresse = QNetworkInterface::allAddresses();
 for (int i = 0; i < listeAdresse.size();i++)
 {
        if(listeAdresse.at(i).toIPv4Address())
                ui->listWidgetIp->addItem((listeAdresse.at(i).toString()));
 }
```

Une fois le codage constructeur réalisé lancez l'exécution et vérifiez l'état du serveur avec la commande ***netstat -a***, que donne la commande ***netstat -l***.

7. ÉCRITURE DU SLOT DE CONNEXION D'UN CLIENT :

Indiquez le nom de la méthode de la classe ***QTcpServeur*** permettant d'initialiser le pointeur ***connexionClient***.

Réalisez le code nécessaire pour cette initialisation et la connexion des signaux indiquant la présence de donnée sur le flux d'entrée de la socket client d'une part et la déconnexion du client d'autre part. (voir dans le code du client si vous avez déjà oublié la connexion des signaux).

Dans le ***listWidget*** contenant l'état des connexions, vous indiquerez l'adresse IP du client connecté et le numéro de port utilisé.

Avec ***netstat,*** en ligne de commande, montrer l'ensemble des connexions actives et leur état pour votre programme serveur lorsqu'un client est connecté, commentez le résultat obtenu.

8. DÉCONNEXION DU CLIENT

Écrire le code du slot invoqué lors de la déconnexion d'un client, affichage de la déconnexion et destruction de la socket de communication du client. Attention cette destruction doit être différée à la fin du traitement de cette socket. (voir dans ***QObject*** que ***QTcpSocket*** à également hérité)

9. ÉCRITURE DE LA RÉCEPTION DE DONNÉES EN PROVENANCE DU CLIENT

En vous inspirant de la méthode écrite dans la classe ***ClientInfoOrdi*** pour la réception des données écrire la méthode équivalente pour le serveur.

Rappel la trame de réponse est de type QString

10. SÉCURISATION DU SERVEUR

Dans notre application nous souhaitons qu'un seul client à la fois se connecte au serveur. Vérifiez le fonctionnement du serveur lorsque plusieurs clients se connectent.

Réalisez le code nécessaire pour limiter la connexion à un seul client. On peut par exemple tester si le pointeur ***connexionClient*** est déjà affecté… si c'est le cas, il ne faut pas accepter le nouveau client, sinon c'est effectivement un nouveau client.