Lycée TOUCHARD WASHINGTON

Section des Techniciens Supérieurs
Systèmes Numériques option
Informatique et Réseaux

TD3 - Socket sous Qt

Audit - Côté serveur

-
- *Date : Novembre 2022*
- *Version : 2*
- *Référence : TD1 – Socket sous Qt (Audit_Client).odt*

1. Objectif

- *Utilisation des sockets avec la bibliothèque **Qt***
- *Comprendre le fonctionnement des sockets en mode événementielè*
- *Codage de l'information*
- *Communication réseau*

2. Conditions de réalisation

- *Ce fichier contient des liens hypertextes.*
- *Ressources utilisées :*
  - *Un PC sous Linux*
  - *Un client est disponible*
  - *Qt-creator*

3. Ressources

*Les classes socket dans la technologie QT, consulter le site https://doc.qt.io/qt-6/qtnetwork-programming.html et plus particulièrement la classe, **QTcpServer** https://doc.qt.io/qt-6/qtcpserver.html.*

4. Le besoin

Les techniciens réseau sont souvent appelés pour une défaillance sur un poste informatique. Avant de se déplacer, il serait intéressant de connaître certaines caractéristiques de la machine. Pour cela, un petit programme «**AuditServeur** » implanté sur chaque machine que le technicien par l'intermédiaire du programme «**AuditClient** » pourra interroger est une aide précieuse. Le travail proposé ici permet de définir et de coder ce client.

4.1. Analyse et conception

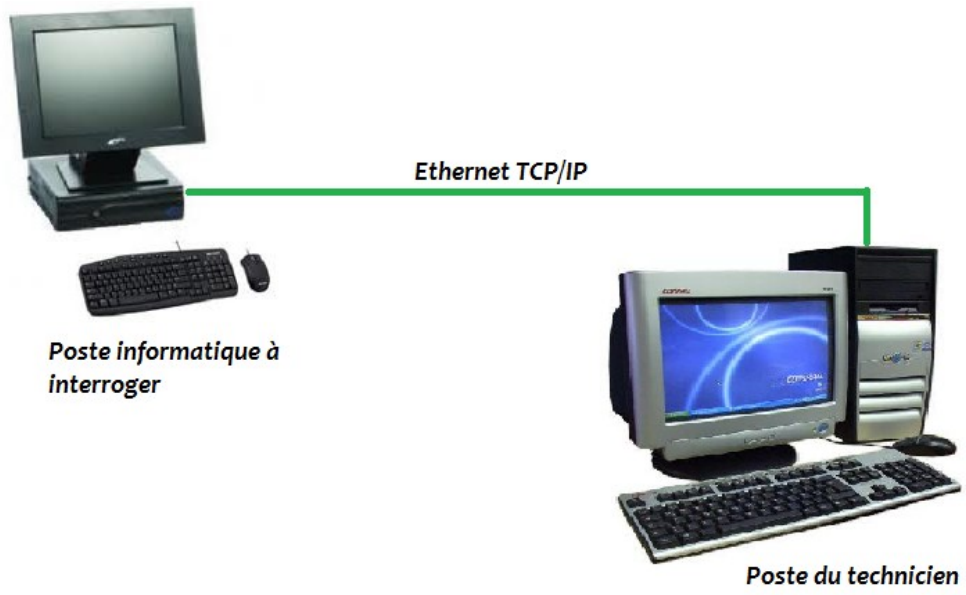
Le serveur est capable de fournir les informations de la machine à l'administrateur client distant.

Selon la demande de celui-ci, le serveur envoie les informations correspondantes :

Demande	*Commande*	*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

D'autres commandes pourront être ajoutées par la suite.

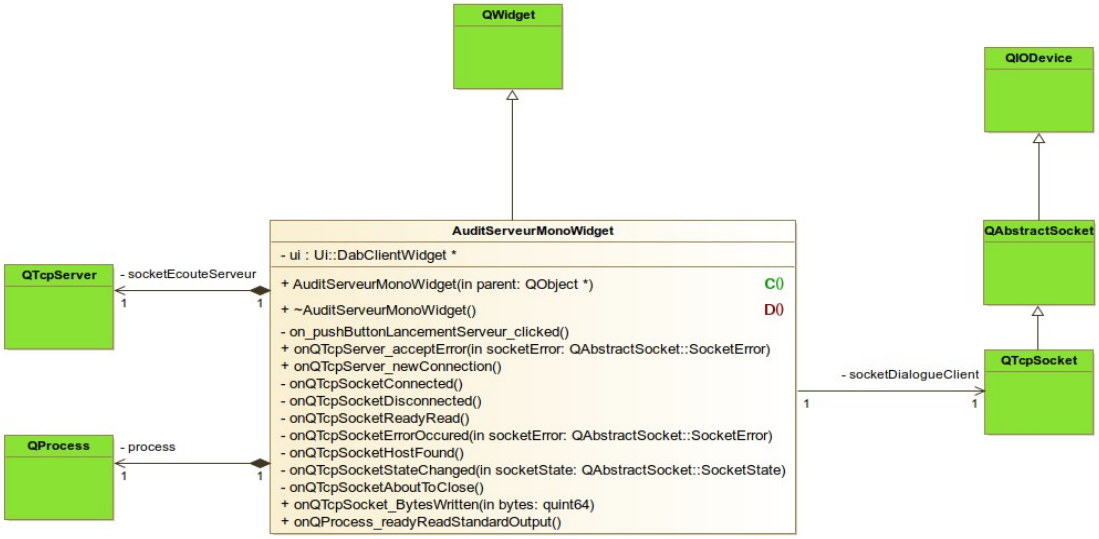
4.2. Mise en situation



5. Réalisation d'un serveur mono-client

5.1. Création du projet

Créez un projet ***AuditServeurMono*** de type Application graphique en C++ sous QT6 avec ***Qt Creator***. La classe principale ce nomme ***AuditServeurMonoWidget***, elle hérite de ***QWidget*** comme le montre le diagramme de classes ci-dessous.

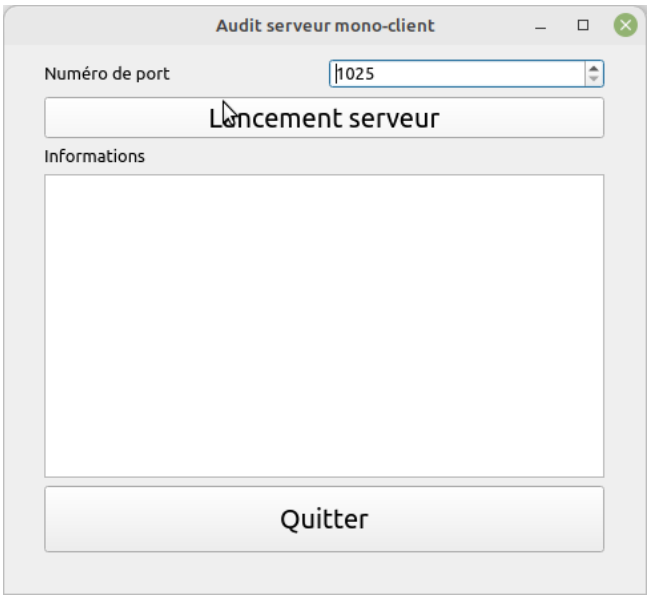


Les relations entre la classe ***AuditServeurMonoWidget*** et les classes ***QTcpServer*** et ***QProcess*** seront implémenté de manière dynamique. Le moment venu, vous ajouterez, le code nécessaire dans le constructeur et dans le destructeur de la classe.

Pour la relation d'association entre la classe ***AuditServeurMonoWidget*** et la classe ***QTcpSocket*** la relation sera instanciée lors de la connexion du client au serveur.

5.2. Création de l'IHM

L'interface du client aura l'aspect suivant :



1. Nommez chaque Widget suivant la convention de nommage habituelle.
2. Associez le bouton de lancement du serveur au slot ***clicked()***. Pour le bouton Quitter, il sera associé graphiquement au slot ***close()*** de ***QWidget***.
3. Complétez la déclaration de la classe ***AuditServeurMonoWidget*** de manière à correspondre au diagramme de classe.

5.3. Utilisation de la classe QTcpServer (version mono client)

4. Dans un premier temps, instanciez l'attribut ***socketEcouteServeur*** dans la classe ***AuditServeurMonoWidget***, comme indiqué à la suite du diagramme de classes. Ensuite, liez les signaux de la classe ***QTcpServer*** aux slots qui seront implémentés dans la classe ***AuditServeurMonoWidget.***
5. Dans quelle méthode de la classe ServeurSocket sera utilisée la méthode ***listen*** de la classe ***QTcpServer ?***
6. A quoi correspondant le retour de la méthode ***nextPendingConnection*** de la classe ***QTcpServer ?***

Après avoir initialiser l'attribut ***socketDialogueClient***, réalisez la connection des signaux en provenance de la socket de dialogue avec le client avec les différents slots ***onQTcpSocket_xxxx*** présent dans le diagramme de classes.

Lors de la déconnexion de la socket de dialogue avec le client, il est nécessaire de détruire la socket pour restituer la mémoire et de lui redonner la valeur ***nullptr*** pour pouvoir être utilisé ultérieurement comme le montre le code ci-dessous :

Déconnexion de la socket de dialogue avec le client **auditserveurmonowidget.cpp**

```
void AuditServeurMonoWidget::onQTcpSocket_Disconnected()
{
    disconnect(socketDialogueClient,nullptr,this,nullptr);
    socketDialogueClient->deleteLater();
    socketDialogueClient = nullptr;
    ui->textEditLogs->append("Client détconnecté");
}
```

7. Expliquer la ligne disconnect(socketDialogueClient,nullptr,this,nullptr) ;
8. Codez le slot de réception des données **onQTcpSocket_ReadyRead**.

Vous utiliserez la méthode **readAll** pour lire les données en provenance du client. Le traitement des demandes **"u"** et **"c"** peut se faire à l'aide de la fonction ***getenv*** et de la méthode statique ***localHostName*** de la classe ***QHostInfo*** retournant chacune un ***Qstring***. Vous pourrez l'utilisez pour envoyer la réponse à la socket de dialogue avec le client.

Récupération du nom d'utilsateur et du nom de l'Hôte

```
QString reponse;

reponse = getenv("USER");

reponse = QHostInfo::localHostName();
```

Pour les demandes **"o"** et **"a"**, il faut passer par l'appel d'un processus externe.

Pour cela, utilisez la classe **QProcess** (un peu l'équivalent de la fonction popen en C), qui permet de lancer un processus externe au programme et de récupérer le résultat sur la sortie standard.

Utilisation de la classe QProcess respectivement pour l'architecture et pour OS

```
process->start("uname",QStringList("-p")); // récupération de l'architecture
process->start("uname"); // récupération de l'OS
```

Principe de fonctionnement : lorsque des données sont disponibles, un signal ***readyReadStandardOutput*** en provenance de ***QProcess*** est émis.

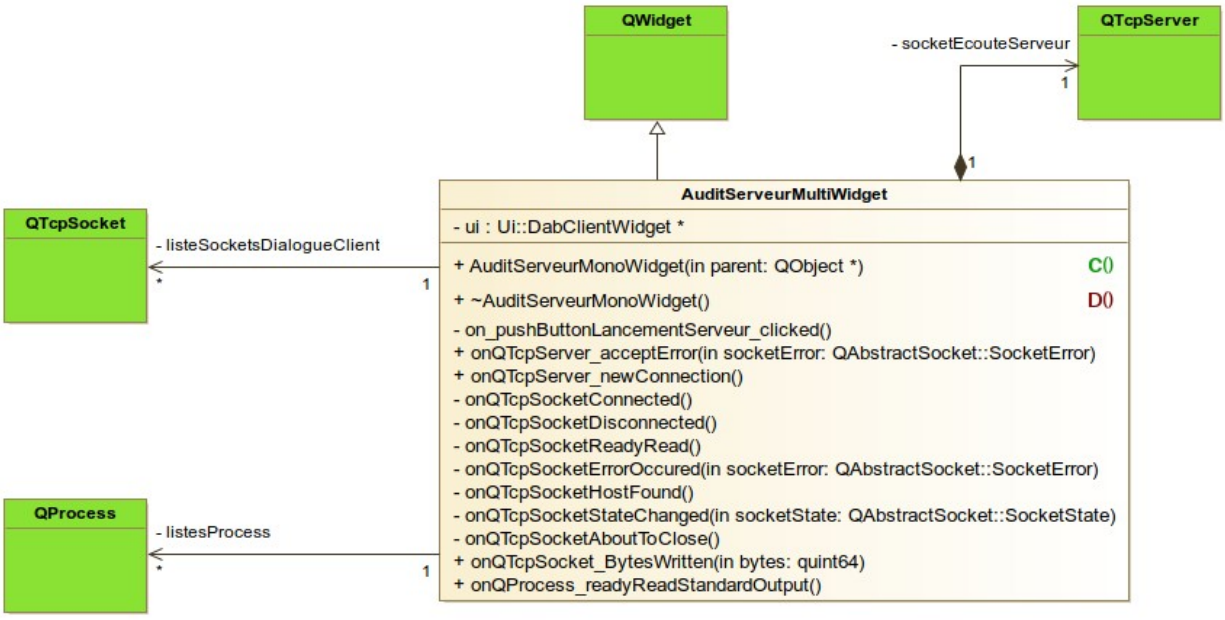
9. Complétez le code du constructeur et du destructeur afin de mettre en place la 2ème relation de composition présente dans le diagramme de classes. Réalisez également la connection du slot ***onQProcess_readyReadStandardOutput*** avec le signal ***readyReadStandardOutput***. Un exemple de code pour le slot est fourni à la suite.

Récupération des informations en provenance de la sortie standard

```
void AuditServeurMonoWidget::onQProcess_readyReadStandardOutput()
{
    QString reponse = process->readAllStandardOutput();
    if(!reponse.isEmpty())
        socketDialogueClient->write(reponse.toLatin1());
}
```

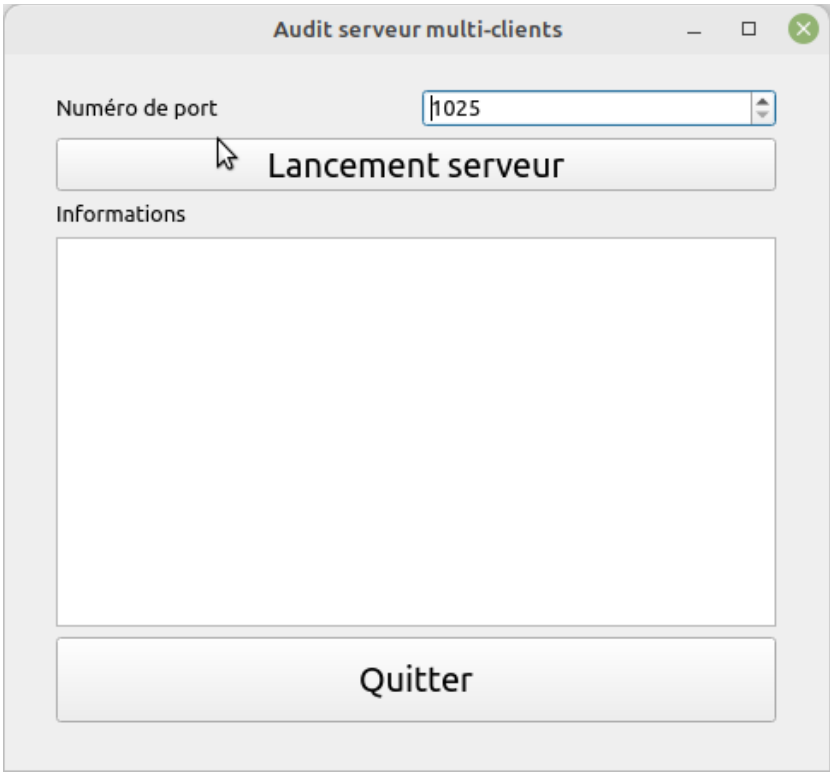
6. Version multi-client

Pour pouvoir traiter un nombre de clients simultanément de façon presque illimitée, le diagramme de classe diffère. L'implémentation de la relation avec les sockets de dialogue avec les clients doit se faire sous la forme d'une liste, **QList** avec la bibliothèque **Qt**. De même un process doit correspondre à chaque socket de dialogue avec le client. La relation devient donc une association implémenté également sous la forme d'une liste.Le contenu de chacune de ces liste s'adappte dynamiquement en fonction du nombre réel de clients connectés.



6.1. Création du projet

Créez un nouveau projet ***AuditServeurMulti*** de type Application graphique en C++ sous QT6 avec ***Qt Creator***. La classe principale ce nomme ***AuditServeurMultiWidget***, elle hérite de ***Qwidget***. Vous pouvez recopiez l'ensemble des widget du projet précédent pour gagner du temps, l'interface est indentique au titre de la fenêtre prêt.



1. Dans le constructeur reprenez uniquement les ligne de code en relation avec la socket d'écoute du serveur. Paramétrez la pour qu'elle accepte jusqu'à 30 connexions simultanées. Dans le destructeur, libérez la mémoire allouer au pointeur sur la socket d'écoute du serveur.
2. Pour l'appui sur le bouton « Lancement du serveur » rien ne change, recopiez le code.
3. Faites évoluer la méthode ***onQTcpServer_newConnection*** pour mettre les nouveaux clients dans la liste des sockets et leur associer un process dans la liste des process.

La gestion des ***QList*** est décrite en suivant ce lien https://doc.qt.io/qt-6/qlist.html. Son fonctionnement est très proche de la liste de la STL étudiée en cours.

Méthode à compléter

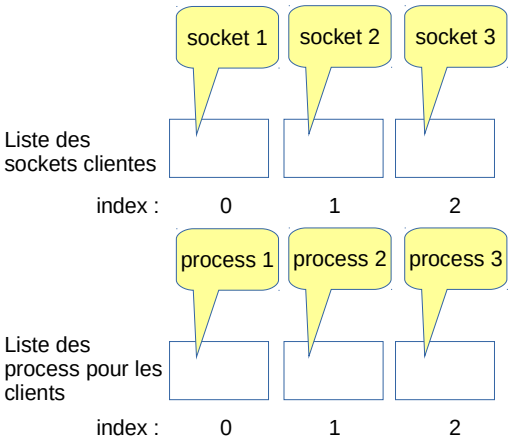
```
void AuditServeurMultiWidget::onQTcpServer_newConnection()
{
    QTcpSocket *client;
    client = socketEcouteServeur->nextPendingConnection();
    connect(client,&QTcpSocket::readyRead,this,&AuditServeurMultiWidget::onQTcpSocket_ReadyRead);
    // complétez pour les autres signaux

    listeSocketsDialogueClient.append(client);  // ajout dans la liste des sockets
}
```

Ajoutez également la création d'un process que vous déposerez dans la liste des process et associez le process au slot ***onQProcess_readyReadStandardOutput***.

Ainsi, pour chaque client se connectant, on aura un process.

Il est possible de connaitre a position d'un objet dans une QList à l'aide de la méthode **indexOf**.

Les sockets cliente et les process étant créés en même temps, les index dans chacune des listes sont les mêmes.



Lorsqu'un client envois des données, c'est toujours le slot ***OnQTcpSocket_readyRead*** qui est appelé, il faut donc déterminer, quel client est à l'origine du signal comme le montre l'exemple suivant dans la méthode ***onQTcpSocket_Disconnected***

Exemple pour obtenir l'adresse du client

```
void AuditServeurMultiWidget::onQTcpSocket_Disconnected()
{
    QTcpSocket *client=qobject_cast<QTcpSocket*>(sender());
    int indexClient = listeSocketsDialogueClient.indexOf(client);
    listeProcess.removeAt(indexClient);
    listeSocketsDialogueClient.removeOne(client);
}
```

4. Expliquez le rôle des méthode ***removeAt*** et ***removeOne*** de la classe ***QList***. Indiquez la différence entre les deux.

5. Complétez la méthode ***OnQTcpSocket_readyRead***

6. Complétez la méthode ***OnQProcess_readyReadStandardOutput***

7. Complétez les autres slot en réponse aux autres signaux de ***QTcpSoket***.