Niveau 4

TP
S7. Réseaux, télécommunications et modes de transmission
S7.7.Programmation réseau
Concept client / serveur
Sockets POSIX

Sockets réseau

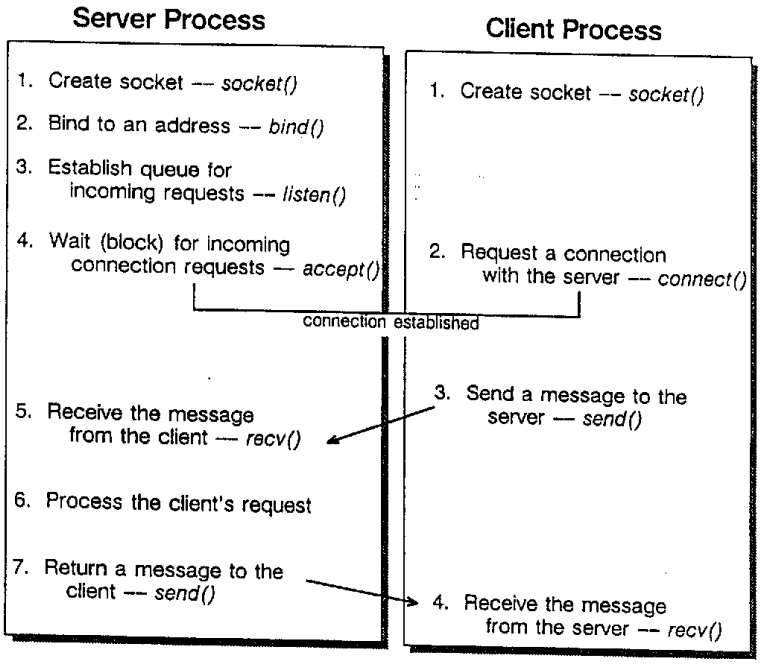
TABLE DES MATIÈRES

1 - PRÉSENTATION

On souhaite transmettre des données (dans notre cas, textuelles) entre deux ordinateurs au moyen de sockets proposés par l'API Windows.

1.1 - RAPPEL DES ÉCHANGES AU TRAVERS D'UN SOCKET

- Mode connecté : `SOCK_STREAM`
- Numéro de port : `5000`
- Famille : `AF_INET`



1.2 - PRINCIPE DE DÉVELOPPEMENT

Il ne va pas être facile de tester nos applications.

On doit travailler avec deux projets simultanés : l'un pour la gestion du Serveur, l'autre pour le Client. Chaque projet devra être créé dans une instance de Qt Creator différente afin de pouvoir les exécuter simultanément au final.

Pour la mise au point, les deux projets fonctionnent sur la même machine : on travaille en `localhost` ou `127.0.0.1` ! Par la suite, il sera possible à votre Client de se connecter au Serveur d'un de vos collègues, sur des machines distantes. Vous modifierez alors l'adresse utilisée dans le Client pour correspondre à celle du Serveur.

Certaines parties du Serveur et du Client sont identiques : il suffira de copier-coller le même code dans chacune. Parfois, des modification mineures doivent être réalisées... Soyez donc bien attentif à ces faux copier-coller !

Il est important que les parties de code qui vous sont fournies dans le cours soient étudiées avec soin pour comprendre le fonctionnement global du processus de communication par sockets.

1.3 - PROTOCOLE DE COMMUNICATION

Lorsqu'un Client fait une requête à un Serveur, un échange normalisé est mis en œuvre. Il est propre au rôle de chaque serveur.

Dans notre cas, le Serveur doit fournir la traduction d'une phrase culte : « Greetings Programs » dans diverses langues.

Le Client fournit le code international de la langue désirée au serveur.

Lorsque le Client ne souhaite plus envoyer de requête au Serveur, il lui indique la FIN... le Serveur remercie le Client pour lui spécifier qu'il va mettre un terme à la communication.

Code de requête	Numéro de requête (interne au Serveur)	Réponse du Serveur
FR	1	Salutations Programmes
EN	2	Greetings Programs
EO	3	Saluton Programoj
ES	4	Saludos Programas
IT	5	Saludis Programmi
CR	6	Pozdravni programi
DE	7	Begrüßungsprogramme
LT	8	Salvete Programs
PO	9	Programy z pozdrowieniami
Autres codes	***0***	***Requete inconnue***
FIN	-1	**Merci Client ! À bientot !**

2 - TRAVAIL À RÉALISER

2.1 - PRÉPARATION DES PROJETS

1) Créez un nouveau dossier nommé `Sockets`.

2) Ouvrez **deux instances** de QT Creator, et créez deux projets de type QT console, l'un nommé `Client`, l'autre `Serveur`. Ces projets sont à enregistrer dans le dossier `Sockets`.

3) Ajoutez la librairie de l'API Windows permettant de gérer les sockets : `libws2_32.a` : Dans le `.pro` de chaque projet, insérez la ligne `LIBS += -lws2_32` !

4) Dans chacun des `main()`, ajoutez en tout début le code nécessaire pour charger la DLL de Windows.

5) En toute fin de chaque `main()`, mais avant le `return()`, insérez le code de libération de la DLL.

Pour la suite, nous allons travailler en premier sur le Serveur uniquement.
Puis une fois son code fonctionnel et qu'il est en attente de connexion, nous passerons au codage du Client, dont de grandes parties seront des copier-coller du Serveur.
Mais gare aux légères modifications qu'il faudra y apporter dans le Client !
L'échange sera codé en premier par l'envoi par le Client d'une requête et de sa réception par le Serveur.

2.2 - LE SERVEUR

Vous compilerez et testerez chaque question codée dans le `main()` avant de passer à la suivante !

6) Déclarez la variable `ParamVide` de type structuré `sockaddr_in` et initialisez-la comme indiqué dans les prérequis du cours.

7) Codez la création du socket de connexion, `nSocketConnexion`.

8) Profitez de cette création pour penser à coder la fermeture du socket de connexion, en fin du `main()`, juste avant la libération de la DLL.

9) À la suite de la création du socket, adressons maintenant le `nSocketConnexion` en lui attribuant une adresse.

10) Créez maintenant, à la suite, la liste d'attente des demandes de connexion.

11) À partir de là, notre Serveur est en mesure d'attendre et accepter les éventuelles connexions. Codez cette partie.

ATTENTION : La fonction utilisée ici est bloquante... On ne peut la passer que si un Client demande une connexion ! La validation de cette dernière question devra attendre qu'une première partie du Client soit codée ! **Laissez votre Serveur s'exécuter pendant la création du Client...**

2.3 - LE CLIENT

12) Dans le `main()` du Client, reprenez les questions 6) à 8), et codez-les.

13) Écrivez le code permettant la demande de connexion du Client.

Lorsque vous allez tester le fonctionnement de cette dernière partie, le Serveur doit être en cours d'exécution et en attente de connexion.
Si c'est le cas, le Serveur se débloque et accepte la connexion.
Si ce n'est pas le cas, le Client retournera une erreur de connexion du socket.

Si tout va bien, la connexion est fonctionnelle à la suite de cette question !

14) Ajoutez le code permettant d'envoyer la requête vers le Serveur.
Vous demanderez à l'utilisateur de saisir au clavier l'un des codes prévus dans le protocole, par exemple `FR`.
Il n'est pas besoin ici de faire une validation du code saisi. C'est le Serveur qui s'en chargera par la suite.
Affichez la requête envoyée ainsi que le nombre d'octets effectivement envoyés.
Par exemple ici : `Requete envoyee : FR (2 caracteres)`

15) Préparez le code de réception de la réponse à cette requête et affichez-la ainsi que le nombre d'octets effectivement reçus.
Par exemple : `Reponse recue : Salutations Programmes (22 caracteres)`

2.4 - LE SERVEUR

16) Codez la réception de la requête et affichez-la ainsi que le nombre d'octets effectivement reçus.
Par exemple : `Requete recue : FR (2 caracteres)`

17) Déclarez et initialisez une variable entière `nNumRequete` devant contenir le numéro de la requête associé au texte de la requête reçue, tel que défini dans le protocole. Dans l'exemple précédent, `nNumRequete` devra prendre la valeur `1` !

18) Rédigez le code permettant au Serveur d'envoyer sa réponse en correspondance avec le `nNumRequete` et le protocole. Affichez la requête envoyée ainsi que le nombre d'octets effectivement envoyés.
Par exemple ici : `Reponse envoyee : Salutations Programmes (22 caracteres)`

Voila ! Notre Serveur et notre Client sont finalisés !
Bien Joué !

Vous pouvez tester des échanges entre vous, un Client et un Serveur sur des machines distantes !

2.5 - MULTI-REQUÊTES

Pour le moment, le Client et le Serveur ne peuvent traiter qu'une seule requête.

Nous allons modifier les deux programmes afin de pouvoir gérer autant de requêtes que l'on souhaite.

19) Modifiez le Serveur de façon à ce qu'il lise une requête et y réponde en boucle jusqu'à ce que le Client lui signale la `FIN` de l'échange. On utilisera ici le `nNumRequete` !

20) Modifiez de même le Client pour qu'il envoie des requêtes et lise les réponses en boucle jusqu'à ce que l'utilisateur décide de la `FIN` de l'échange.

Vous avez maintenant de beaux Client et Serveur capables de gérer plusieurs requêtes !
Bravo !

Vous pouvez tester des échanges entre vous, un Client et un Serveur sur des machines distantes !

2.6 - SERVEUR MULTI-CLIENTS

Le problème de notre Serveur est qu'il n'accepte qu'un seul Client ! Imaginez les serveurs Web travaillant ainsi ! Ce serait la galère pour accéder à un site internet !

Nous allons transformer notre Serveur en Serveur multi-tâche.

Son fonctionnement global ne changera que très peu :

- Il doit attendre une connexion ;
- une fois un Client accepté, il crée et lance un thread dédié à ce Client ;
- puis il se remet en attente d'une autre connexion... Et ceci en boucle infinie !

Le thread ainsi créé et lancé s'occupera, lui, de la réception des requêtes du Client et des réponses, en boucle, comme notre ancien Serveur le faisait.

Ne fermez pas le Qt Creator gérant le projet du Client : il va nous être nécessaire ici aussi !

2.6.1 - CRÉATION D'UN NOUVEAU PROJET

21) Copiez le directory du projet du Serveur et renommez-le en `ServeurThread`.

22) Renommez le `.pro` en y ajoutant `Thread` à la fin.

23) Lancez une nouvelle instance de Qt Creator et ouvrez ce nouveau projet.

24) Dans le `main()`, modifiez la variable `nNbreClientMaxi` en l'initialisant à `10`. C'est le nombre maximal de demandes de connexion que le Serveur va pouvoir gérer grâce à sa liste d'attente.

2.6.2 - PRÉPARATION DU THREAD

25) Définissez la fonction qu'exécutera le thread. Vous pourrez ici vous servir de la soluce du précédent TP sur le multi-tâche.
Cette fonction reprend intégralement le code de la boucle de réception et traitement de la requête du Client, ainsi que la fermeture du socket de communication une fois la boucle terminée.

26) Ajoutez, dans le `main()`, le code utile à la déclaration, la création et l'initialisation de la priorité du thread.
Cette partie est à mettre à la place de la boucle de traitement des requêtes, juste après la phase d'acceptation d'un Client !
Remarque : La fonction du thread a besoin de connaître le `nSocketDeCommunication` pour bien traiter le bon Client ! Il nous faut forcément lui passer ce socket en paramètre, lors de la création du thread !

27) Dans la fonction du thread, déclarez une nouvelle variable `nSocketDeCommunication` que vous initialiserez avec le paramètre reçu correctement transtypé.

28) Gare ! Comme le Serveur a déporté son code de traitement de la requête dans le thread, il ne sera plus bloqué par la lecture de la requête ni par l'envoi de sa réponse... Il va se terminer rapidement après avoir lancé le thread !
Ajoutez les quelques lignes qui demande l'appui sur une touche juste avant la fermeture du socket de connexion !

29) Vous êtes prêt maintenant à lancer votre thread ! Ajoutez le code adéquat à la suite de l'initialisation de la priorité.

À ce stade, votre Serveur doit avoir le même comportement que précédemment :
il ne peut toujours gérer qu'un seul Client... mais il le fait grâce à un thread !

2.6.3 - PASSAGE EN MULTI-CLIENTS

Le Serveur va maintenant pouvoir accepter, en boucle infinie, toutes les demandes de connexion des Clients et en déléguer la gestion à des threads individuellement affectés à chacun.

30) Encadrez le code d'attente et d'acceptation de connexion ainsi que la gestion du thread dans une boucle infinie. C'est tout !

31) Lancez une nouvelle instance de Qt Creator et ouvrez le projet du Client. Vous avez ainsi deux Clients sur votre poste que vous pouvez lancer pour tester !

Ayé !
Voila votre Serveur capable de gérer un nombre incalculable de Clients !
Génial !
Félicitation !

Vous pouvez tester des échanges entre vous, des Clients et un Serveur sur des machines distantes !

3 - BONUS

Notre serveur multi-Clients n'est pas vraiment bien finalisé : Si le thread s'occupe de fermer le socket de communication, personne ne s'embarrasse à bien clôturer les threads, qui encombrent la mémoire sous forme fantôme. N'ayant pas mémorisé les handles à chaque création, il est impossible de les supprimer correctement.

32) Le thread peut aussi clôturer son propre handle à la fin de son exécution.
Reste à lui passer en paramètre : Créez une structure de données permettant de mémoriser le handle du thread et le socket de communication.
Initialisez une variable de type de cette structure avec les bonnes valeurs et passez-la en paramètre à la fonction du thread.
Dans la fonction du thread retrouvez le handle du thread ainsi que le socket de communication
En fin du thread, clôturez son handle.

33) Au moyen d'un conteneur de type vecteur, mémorisez chaque handle de thread créé. Ce qui permettra aussi de savoir combien de Clients sont actuellement connectés !