TP1 - Sockets Windows

Sockets et portabilité

Koalab koala@epitech.eu

Abstract: *Ce TP est une introduction aux sockets sous Windows (Au travers de la WSAPI), ainsi qu'à l'abstraction des mécanismes des sockets dans le but de réaliser du code portable.*

Table des matières

Chapitre I

Préambule

I.1 Cadre et esprit de ce TP

Ce TP a deux buts complémentaires :

- Vous faire découvrir la facette Windows du développement réseau au travers de l'utilisation de la Windows Socket API
- Vous faire développer une abstraction basique aux sockets, afin de pouvoir réaliser des projets réseau portables

Ce TP peut vous paraître long si vous n'êtes plus très habitués à l'utilisation des sockets. Cependant, il n'est pas acceptable de le faire à moitié, car *TOUTES* les notions appliquées dans ce TP seront par la suite considérées comme *ACQUISES*, et leur application sera exigée en **soutenance** !

Ne cédez pas à la flemme !

I.2 Remarques diverses

Nous attachons une grande importance à la qualité de nos documents. Si vous constatez dans ce sujet des erreurs, des fautes d'orthographe ou de typographie, ou si vous avez des remarques quelconques à faire, n'hesitez pas a nous contacter par mail (koala@epitech.eu).

I.3 Pré-requis

Afin de faire ce TP, vous allez avoir besoin des choses suivantes :

- Un Visual Studio en état de fonctionnement
- Un Netcat pour Windows, qui vous servira pour vos tests http://www.thaoh.net/Tools/Netcat/

- Un éditeur UML (Violet UML Editor, graphviz, PowerAMC, boUML, Dia, ...)
- Un cerveau en bon état !

Chapitre II

Un peu de lecture

II.1 Rappels sur les sockets

Vous devriez déjà tous, grâce à vos projets de deuxième année, savoir ce qu'est un socket.

Cependant, si vous ressentez le besoin d'un petit rafraîchissement mémoriel, voici un peu de lecture :

- *man 2 socket*
- http://www.linuxhowtos.org/C_C++/socket.htm
- http://beej.us/guide/bgnet/

Ca va mieux ? Parfait. Passons au vif du sujet.

II.2 Les sockets Windows et la WSAPI

Le but de ce TP est, comme son nom ne l'indique pas, de vous faire réaliser des sockets sous Windows.

Vous allez à cet effet utiliser la Windows Socket API (WSAPI de son petit nom). Microsoft maintient une documentation accessible sur Internet et Visual Studio : La MSDN. C'est à partir de cette documentation (et de cette documentation seule !) que vous devrez réaliser les exercices de ce TP.

Lisez la section relative a l'API Winsock sur la MSDN (http://msdn.microsoft.com/en-us/library/ms741416%28v=VS.85%29.aspx), en vous concentrant surtout sur les différentes fonctions à votre disposition.

Vous constaterez que la plupart des fonctions de gestion de socket auxquelles vous avez été habitués sous UNIX se retrouvent également sous Windows. Cependant, vous *DEVEZ* utiliser l'équivalent WSAPI (Fonction préfixée par WSA) quand il existe. Cela vaut pour ce TP ainsi que pour *TOUS* vos projets du module C++ avancé !

Une fois que vous avez lu la documentation et identifié avec certitude les fonctions WSA* à utiliser, relisez les avant de passer à l'étape suivante.

Chapitre III

Sockets Windows

III.1 Préambule

Pendant ces premiers exercices, vous allez utiliser tous les aspects des sockets Windows : Client et serveur, mais aussi TCP et UDP.

Ces exercices peuvent sembler rébarbatifs, mais vous devez absolument les faire sérieusement, ce afin de vous habituer à leur utilisation !

III.2 Exercice 1

- Vous allez commencer par réaliser un petit serveur TCP (socket STREAM), qui se contentera d'accepter une connection, lire un message dessus, et y répondre "Hello, world !".
- Utilisez netcat ou PuTTY (en mode RAW) pour envoyer des données et tester votre serveur.

III.3 Exercice 2

- Réalisez un client TCP qui envoie un message à un serveur, reçoit une réponse, puis se déconnecte.
- Testez dans un premier temps en vous connectant à un netcat, puis à votre serveur de l'exercice 1.

III.4 Exercice 3

- Refaites la même chose que pour l'exercice 1, mais cette fois-ci en utilisant le protocole UDP (socket DGRAM).

III.5 Exercice 4

- Refaites la meme chose que pour l'exercice 2, mais cette fois-ci en utilisant le protocole UDP (socket DGRAM).

Chapitre IV

Abstraction aux sockets

IV.1 Préambule

Vous savez maintenant utilisez les sockets sous UNIX et sous Windows.

Vous allez donc réaliser une abstraction aux sockets, afin de faciliter la réalisation de projets portables.

J'attire votre attention sur le coté *CRITIQUE* de cette abstraction. Son utilisation lors de vos projets n'est *PAS* optionnelle, et sa qualité sera evaluée.

Pour les besoins de ce TP, vous vous contenterez de réaliser une abstraction aux sockets TCP. A vous ensuite de trouver un moyen intelligent de représenter des "sockets" UDP.

IV.2 Exercice 5

- Vous allez commencer par réaliser l'abstraction la plus basique possible, c'est à dire une simple encapsulation de chaque appel système disponible.
- Identifiez les appels systèmes nécessaires à la manipulation de sockets.
- Créez sur un diagramme de classes une interface possédant une méthode pour chacun de ces appels, avec des protoypes comportant les informations suffisantes à l'appel système, que ce soit sous UNIX ou sous Windows.

Vous constaterez certainement qu'en général, les fonctions WSA* prennent plus de paramètres que leurs équivalents POSIX. Deux choix s'offrent a vous pour les regrouper : Prendre tous les paramètres possibles et en ignorer certains, ou prendre le moins possible de paramètres et donner des valeurs par défaut aux autres.

A vous de choisir la façon de faire qui vous paraît la plus pratique et la plus intelligente.

- Une fois votre interface conçue, complétez votre diagramme de classes avec deux implémentations de votre interface, une pour Windows et une pour UNIX.
- Faites valider votre avancement par un assistant.
- Cela fait, réalisez cette abstraction en C++.

IV.3 Exercice 6

- Refaites les exercices 1 et 2 à l'aide de l'abstraction que vous avez réalisee.
- Votre code *DOIT* compiler et fonctionner de façon identique sous Windows et UNIX.

IV.4 Note sur l'abstraction

L'abstraction que vous venez de réaliser est la forme la plus simple d'abstraction aux sockets, à savoir une encapsulation directe des appels systèmes au travers d'une interface.

Ce type d'abstraction fonctionne pour presque toutes les problématiques de portabilité. Cependant, ce n'est pas la plus élégante d'une point de vue objet ni d'un point de vue pratique. Ainsi, si vous vous en contentez, vous serez pénalisés dans vos projets, pendant la réalisation comme à la notation.

IV.5 Exercice 7

- Vous allez maintenant réaliser une abstraction aux sockets plus élégante que l'abstraction primitive que vous venez de réaliser.

L'abstraction que nous vous proposons ici n'en est qu'une possible parmi d'autres. N'hésitez surtout pas à expérimenter et à nous proposer vos propres idées sur cette problématique !

- Reprenons la base ...
- Pour les besoins de cet exercice, on séparera les sockets en deux catégories : Un socket "serveur" (Qui ne sert qu'à accepter une connection) et un socket "client" (Socket qui se connecte à un serveur, ou socket renvoyé par un accept(), qui ont le même comportement).
- Vous allez commencer par le socket "client". Implémentez l'interface ISocket suivante (Sous Windows et UNIX, bien entendu) :

```
1       class ISocket
2       {
3           virtual ~ISocket() {}
4           bool connectToServer(std::string const & host, short port);
                //Initialise le socket et connecte a l'hote passe en
                parametre
5           bool connectFromAcceptedFd([...] fd); //Initialise l'objet
                autour du socket passe en parametre
6           int recv(std::string& buffer, int blocksize);
7           int send(std::string const & data);
8       };
```

- Comme vous l'aurez probablement compris, vous devez évidemment conserver le fd du socket dans un attribut de votre implémentation, et résoudre toutes les problématiques de bind, connect, etc ... dans *connectToServer*.
- Créez ensuite votre socket "serveur" à travers l'interface suivante :

```
1       class IServerSocket
2       {
3           virtual ~IServerSocket() {}
4           bool init(std::string const & listenHost, short listenPort);
5           ISocket* accept();
6       };
```

- Vous avez maintenant un moyen entièrement abstrait et orienté objet d'utiliser des sockets TCP.

IV.6 Exercice 8

- Refaites les exercices 1 et 2 a l'aide de l'abstraction que vous avez realisée.
- Votre code *DOIT* compiler et fonctionner de façon identique sous Windows et UNIX.

Chapitre V

Pour aller plus loin

- Réfléchissez à une abstraction du réseau UDP, peut-être en représentant également des "sockets" manipulables de la même manière que les sockets TCP ... ?
- Améliorez vos abstractions réseau, et faites en sorte qu'elles soient réutilisables. Vous gagnerez un temps précieux dans la réalisation de vos projets.