TPC#9 : Client & Serveur !

Table des matières

1 Structure du rendu

La structure du rendu pour ce TP ne change pas de celles que vous avez l'habitude de voir. Encore une fois, `login_x` est à remplacer par votre propre login.

```
rendu-tpcs9-login_x.zip
    - login_x/
        - AUTHORS
        - Client
           - Client.sln
           - Client/
        - Serveur
           - Serveur.sln
           - Serveur/
```

2 Introduction

Nous allons dans ce TP vous introduire les notions de sockets, threads et d'autres notions génériques en réseau. Vous allez aussi réutiliser les notions vues précédemment (Stream...). Pour cela, vous allez réaliser un petit chat ainsi qu'un serveur pour ce chat.

> "L'environnement client-serveur désigne un mode de communication à travers un réseau entre plusieurs programmes ou logiciels : l'un, qualifié de client, envoie des requêtes ; l'autre ou les autres, qualifiés de serveurs, attendent les requêtes des clients et y répondent. Par extension, le client désigne également l'ordinateur sur lequel est exécuté le logiciel client, et le serveur, l'ordinateur sur lequel est exécuté le logiciel serveur." - Wikipédia

3 Sockets et Threads

3.1 Les sockets

3.1.1 Cours et exemples

Avant de commencer notre chat, vous avez besoin de nouvelles notions. Vous allez voir quelques exemples pour apprendre à utiliser ces outils, puis vous pourrez commencer le chat.
Un socket est une interface logicielle de communication entre les services du système d'exploitation. Mais qui dit socket, ne dit pas obligatoirement réseau : on peut utiliser les sockets pour communiquer entre les applications d'un ordinateur. Pour établir une connexion en C#, on utilise la classe Socket. On va pouvoir récupérer des flux à partir de cet objet pour dialoguer. Pour établir cette connexion, il faudra connaître l'adresse distante (donc le serveur) ainsi que le port. Le port est un simple numéro représentant un service. Souvent on associe un port à un protocole.

```
using System.Net.Sockets;
```

Le constructeur d'un socket prend 3 paramètres :

- Le type d'adresse, ici nous utilisons de l'IPv4 : AddressFamily.InterNetwork
- Le type de socket : SocketType.Stream
- Le protocole de connexion utilisé entre les deux machines. Nous utiliserons le TCP, c'est le protocole le plus courant pour ce genre d'application, il permet de garder une connexion entre deux machines, de vérifier que le destinataire a bien reçu le message, et bien d'autres choses encore. Utilisez donc ProtocolType.Tcp.

Une fois votre socket initialisée, vous pouvez établir la connexion grâce à la méthode Connect de votre socket.
Voici un exemple de code utilisant les sockets pour envoyer "Hello" à Google (notez l'utilisation de Stream, que vous devrez améliorer pour la suite).
N'oubliez pas de flush les streams et de fermer les sockets.

```
Socket s = new Socket(AddressFamily.InterNetwork, SocketType.Stream, ProtocolType.Tcp);
s.Connect("www.google.fr", 80);
Stream stream = new NetworkStream(s);
stream.WriteByte((byte)'H');
stream.WriteByte((byte)'e');
stream.WriteByte((byte)'l');
stream.WriteByte((byte)'l');
stream.WriteByte((byte)'o');
stream.WriteByte((byte)'\n');
stream.Flush();
s.Close();
```

3.1.2 Exercice

Faites un nouveau console nommé MyBrowser. Dans ce projet, vous allez vous connecter à un serveur HTTP et afficher la réponse du serveur. Le but est de récupérer une page web.

- Le nom du site web devra être récupéré en argument sur la ligne de commande (utilisez donc args).
- Le nom du site testé sera de la forme suivante : "www.exemple.fr".
- Utiliser le port 80, c'est à dire le port http. C'est sur ce port que votre navigateur se connecte aux sites internet pour récupérer des pages web.
- Pour récupérer une page web, il faudra envoyer le message suivant :
  "GET /[pathtoressources] HTTP/1.1\nHost : [host]\nConnection : close\n\n" Vous pouvez vous contenter de copier coller cette ligne, mais il faudra néanmoins remplacer [host] par le nom du site sans le "www."
  Par exemple, pour http ://www.google.fr, il faudra mettre :
  "GET / HTTP/1.1\nHost : google.fr\nConnection : close\n\n"
  Vous pouvez consulter la page du Wikipédia sur le protocole HTTP pour en apprendre plus, car ce n'est pas le sujet d'aujourd'hui.
- Utilisez un Console.Read() à la fin de votre main pour laisser le temps à l'utilisateur de voir ce qui s'est affiché.
- Utilisez les StreamReader et StreamWriter.
- Comme exemple pour cette exercice, vous pouvez si vous le voulez récuperer le fichier browser_test à l'adresse suivante : perso.epita.fr/ deabre_p/browser_test.

3.2 Les Threads

Vous allez avoir besoin d'utiliser des threads pour votre chat.

3.2.1 Cours et exemples

Qu'est ce qu'un thread ? Un thread est une tâche qui peut s'effectuer en "parallèle" à d'autres tâches. L'éxécution de votre programme va se découper en plusieurs sous tâches. Pour l'instant, nous n'avions qu'un seul thread (une seule tâche). Avec un autre thread, nous pourrions en plus faire autre chose. Nous allons donc nous occuper de la réception des messages du chat sur un autre thread. Les threads sont particulièrement utiles pour les interfaces graphiques, cela permet de faire des calculs lourds sur un (ou plusieurs) thread et de continuer à faire fonctionner l'interface sur un autre thread. Le constructeur prend en paramètre un objet ThreadStart, qui prend lui même un delegate (cette notion sera abordée plus tard, passez lui simplement le nom de votre méthode, donc sans mettre de parenthèses). Il ne reste plus qu'à faire appel à la méthode Start de notre objet, pour lancer le thread.
Exemple de code :

```
static void Main(string[] args)
{
Thread t = new Thread(new ThreadStart(MyThread)); // Création de l'objet Thread.
t.Start(); // L'appel de cette méthode appelle la méthode "MyThread".
Thread.Sleep(500); // Met le thread principal en pause pour 500 ms.
t.Abort();\\
}
\\
static void MyThread()
{
// Fonction qui sera lancée pour chaque thread.
}
}
```

3.2.2 Exercice

Faites un nouveau projet Console nommé MyFirstThread. Faites une méthode statique qui affiche 100 fois "Thread 1" et une autre affichant 100 fois "Thread 2".
Dans le Main, faites 2 threads éxécutant chacun l'une des deux méthodes.
Cet exercice simple vous fera comprendre comment les threads s'éxécutent.

4 Le client

Nous allons maintenant nous attaquer à la partie client de ce TP. La partie client de votre application est le logiciel qui va se connecter au serveur du chat. Pour tester cette partie, demandez à vos ACDCs qu'ils lancent le serveur et qu'ils vous donnent l'IP de leur machine.

Commencez tout d'abord par faire une nouveau projet console que vous nommerez Client. Faites une nouvelle classe nommée Client. Cette classe doit contenir les attributs suivants :
- Le nom de l'utilisateur essayant de se connecter
- Une socket
- Un StreamWriter
- Un StreamReader

Vous pouvez ajouter d'autres attributs si cela vous semble utile.

Vous devrez dans un premier temps fournir un constructeur sans paramètres qui initialise la socket ainsi que les méthodes suivantes :

```
public void Connect(string host, int port, string name);
public bool IsConnected();
public void Run();
public void Read();
public void Write();
public void Close();
```

- Connect : se connecte au serveur demandé grâce à l'hôte et au port passés en paramètre. Vous pouvez gérer le cas où la connexion ne se serait pas effectuée grâce une SocketException et afficher un message d'erreur. A la connexion, envoyez tout d'abord votre pseudo, puis si le message reçu sur le StreamReader est "Welcome" alors on affichera un message "Confirmation : Welcome", sinon on affichera "You're blacklisted, get lost".
- IsConnected : retourne vrai si le socket est connecté à un hôte distant. Il existe une méthode de Socket permettant de vérifier cela.
- Run : Crée deux threads (vous passerez en paramètre du ThreadStart les méthodes Read à l'un et Write à l'autre, défnies ci-dessous), les lance puis fait en sorte que les threads se bloquent/attendent jusqu'à ce que le thread que l'objet représente se termine, pour cela nous utiliserons la méthode Join. Elle appellera ensuite la méthode Close (voir ci-dessous).
- Read : si il y a des données disponibles pour la lecture, affiche ces données (qui sont sur le StreamReader) sur la console. Pour cela, regarder la documentation de la méthode Poll d'un Socket.
- Write : Lis sur la console la phrase qui va être envoyée au serveur, le met sur le StreamWriter. On oublie pas de flush ce dernier à la fin.
- Close : Flush le StreamWriter, ferme ce dernier ainsi que le StreamReader et ferme le socket.

5 Le serveur

Maintenant que le client est terminé, nous allons faire le serveur. Pour le serveur, vous devez créer un nouveau projet console car le client et le serveur sont deux parties différentes (ce seront deux éxécutables différents, vous lancerez le serveur d'un côté et le client de l'autre!). Nommez ce projet Server. On décomposera cette partie en deux sous parties : la gestion des utilisateurs, et le traitement et la transmission des messages aux utilisateurs.

5.1 La classe Client

Vous allez créer une classe Client. Celle-ci aura les attributs suivants :
- Un nom, que l'on pourra récupérer mais pas modifier (utilisez les getters et setters).
- Un nom d'hôte
- Un port
- Une socket associée
- Un StreamReader

– Un StreamWriter

Elle aura un constructeur qui prendra en paramètre un nom, un nom d'hôte, un port et une socket. Ele possèdera les méthodes suivantes :

```
public void Send(string message);
public string Receive();
```

Send écrit le message passé en paramètre sur le Stream (n'ouliez pas de flush).
Receive lit un message sur le Stream et le retourne. Si une erreur survient, la méthode renvoie null.

5.2 La classe Serveur

Vous allez maintenant créer une classe Server. Notre serveur possèdera deux attributs : une socket qui lui est associée et une LinkedList pour stocker les différents clients connectés au serveur et ainsi gérer le multiclient.
Le constructeur prendra en paramètre le port. Dans ce constructeur on initialise le socket. Nous allons vouloir accepter une connexion, pour cela nous allons utiliser les méthodes Bind et Listen. La méthode Bind prend en paramètre un IPEndPoint qui lui même prendra un IPAddress.Any en paramètre. Regardez comment utiliser la méthode Listen, elle vous sera utile. On initialise également la liste des clients. Vous devez faire les méthodes suivantes :

```
public void Run();
public void AcceptClient();
public void Chat();
```

- Run : crée deux threads : un pour lancer AcceptClient (voir ci-dessous) et un autre pour lancer la méthode Chat (voir ci-dessous).
- AcceptClient : elle attend la connexion d'un client à l'aide de la méthode Accept qui retourne un objet de type Socket et qui symbolise le lien entre le client et le serveur. Nous allons créer ici notre client (on accédèra aux informations nécessaires grâce au socket récupéré), l'ajouter à notre liste et envoyer "Welcome". On écrira également sur la console "Connexion from [client name]". "Client name" à remplacer par le nom du client renseigné auparavant bien évidemment.
- Chat : Cette fonction va écrire les messages qui seront envoyés par les différents clients (ceux qui se sont connectés évidemment !) sur la console sous la forme suivante : "[client name] : [message]". Chaque client recevra ce message également.