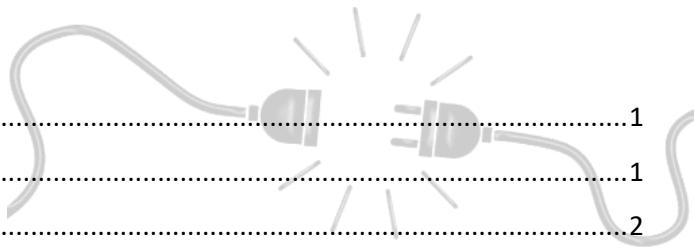


TP Programmation réseau : les Sockets

Module : Programmation 5 : système et réseau



1. Sommaire

2. Objectifs	1
3. Introduction	1
4. netcat (Linux).....	2
5. Création d'un projet C# .NET avec Git sous Visual Studio.....	2
6. Socket UDP (mode non connecté).....	4
7. Fonctionnement client/serveur (TCP)	7
Serveur TCP	8
Client TCP	8
Observation des paquets échangés avec « WhireShark »	8
8. Réception asynchrone de données à partir d'un Socket	9
9. TCPCCLIENT et TCPLISTENER	10
10. Annexe	11
La Classe Socket.....	11

2. Objectifs

- Réaliser des programmes communiquant via le réseau en UDP et en TCP (Client/Serveur)
- Utiliser la commande netcat sous linux
- Mise en œuvre de la classe socket du Framework Microsoft
- Programmer en C#
- Pratiquer la programmation avec un gestionnaire de version Git.

3. Introduction

Les sockets étaient au début un support de communication réseau qui avait été introduit en 1984 par Bill Joy, lors de la sortie de la version 4.2 BSD (*Berkeley Software Distribution*) d'Unix appelée Berkeley Unix. Ce support de communication est finalement devenu un standard et a été rapidement utilisé pour les **communications réseau et interprocessus**.

Nous utiliserons dans ce TP, la **Classe Socket du Framework** qui implémente l'interface de sockets Berkeley.

Définition :

Un Socket est un point de terminaison dans une communication bidirectionnelle entre deux programmes fonctionnant sur un réseau.

Il est associé à un numéro de port afin que la couche TCP ou UDP puisse identifier l'application vers laquelle les données doivent être transmises.

4. netcat (Linux)

⇒ Quelles sont les fonctionnalités et les utilisations de netcat ?

Netcat est un outil en ligne de commande polyvalent qui permet de lire, écrire et transférer des données à travers des connexions réseau en utilisant les protocoles TCP ou UDP, souvent utilisé pour le diagnostic, le transfert de fichiers, et le test de ports.

⇒ Lancer une machine virtuelle Debian avec VirtualBox. La VM doit avoir une GUI et Wireshark.

Etablir une connexion par socket UDP entre deux terminaux sous linux en utilisant la commande **netcat (nc en abrégé)**. Pour cela trouver les paramètres adéquates :

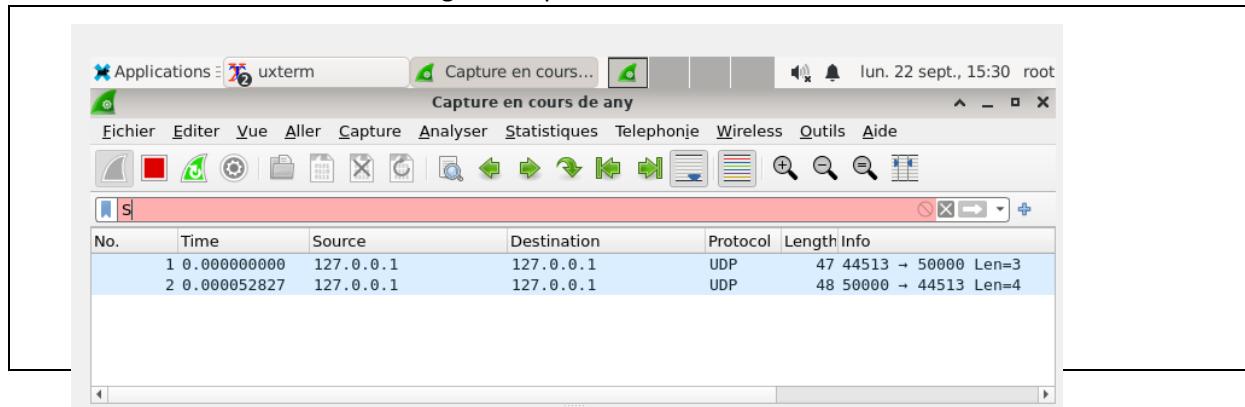
⇒ Ligne de commande pour lancer un serveur sur le port 50000 :

```
nc -l -p 50000
```

⇒ Ligne de commande pour lancer un client sur le serveur précédent :

```
nc -u 172.0.0.1 50000
```

⇒ Relever une trame d'échange en udp avec « WhireShark »



Autoévaluation

5. Création d'un projet C# .NET avec Git sous Visual Studio

Faire un commit lorsque c'est indiqué dans les questions « Commit # » suivi d'un numéro.

Le nom du commit commencera par # suivi du numéro indiqué puis de quelques mots pertinents résumant ce qui a été fait.

Note : Il est très important de bien nommer un commit.

Avant chaque commit, assurez-vous que les modifications sont correctes en effectuant un test (exécution du programme, fonctionnalité validée).

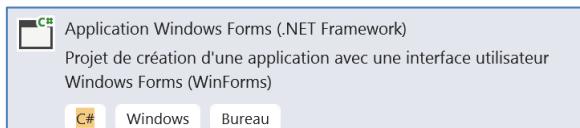
Vous pouvez faire des commits supplémentaires pour des corrections de code (indiquez l'erreur corrigée) ou faire du refactoring de code en précisez la modification (ajout de commentaires, tabulations, changement de nom de variable...).

⚠ N'utiliser pas d'opérations Git sans en connaître les conséquences.

L'évaluation du TP sera basée sur la progression des validations (commits) dans le gestionnaire de version, par la consultation du .git présent dans le BACKUP de votre répertoire de TP et aussi par la consultation de votre dépôt GitHub.

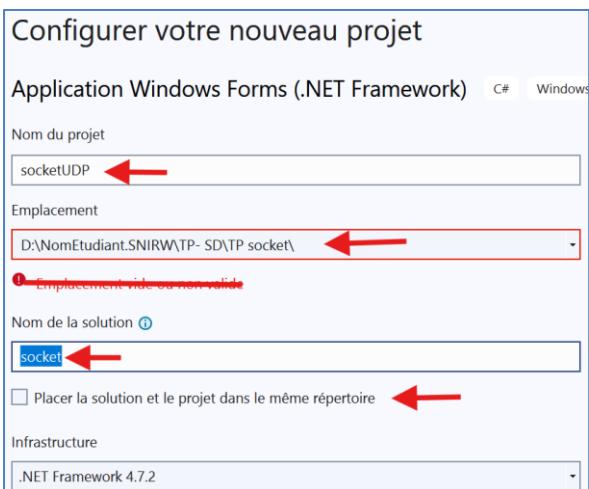
⌚ Lancer Visual Studio

⌚ Créer un nouveau projet Visual Studio C# / Windows Forms / .NET Framework

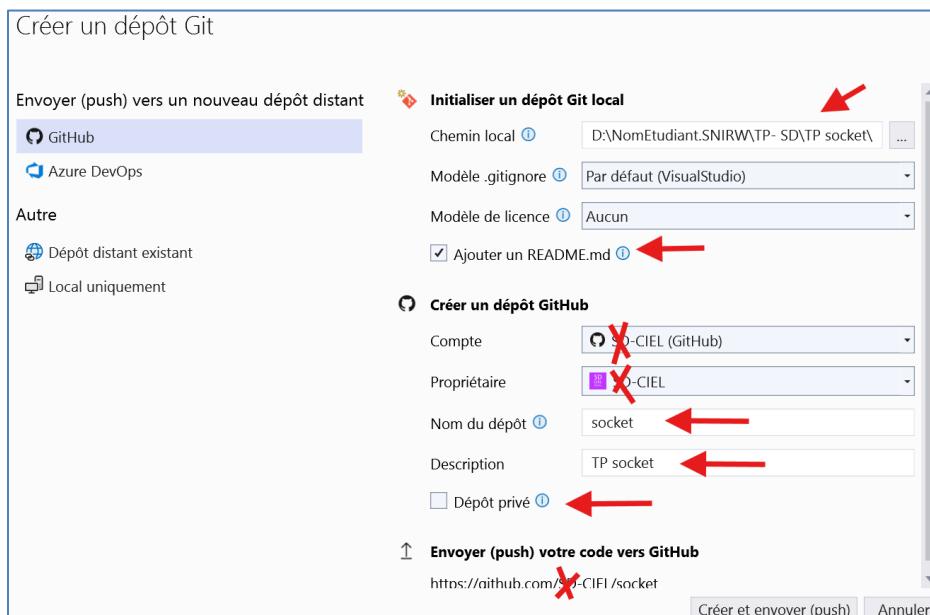


⌚ Configurer le projet nommé « socketUDP » de la solution « socket » dans le répertoire :
D:\NomEtudiant.SNIRW\TP- SD\TP socket

Dans ce TP, il aura deux projets dans la solution :



⌚ Initialiser un dépôt Git et GitHub avec :



⇒ Renseigner le README.md avec le nom du TP et la date.

👉 Commit #1

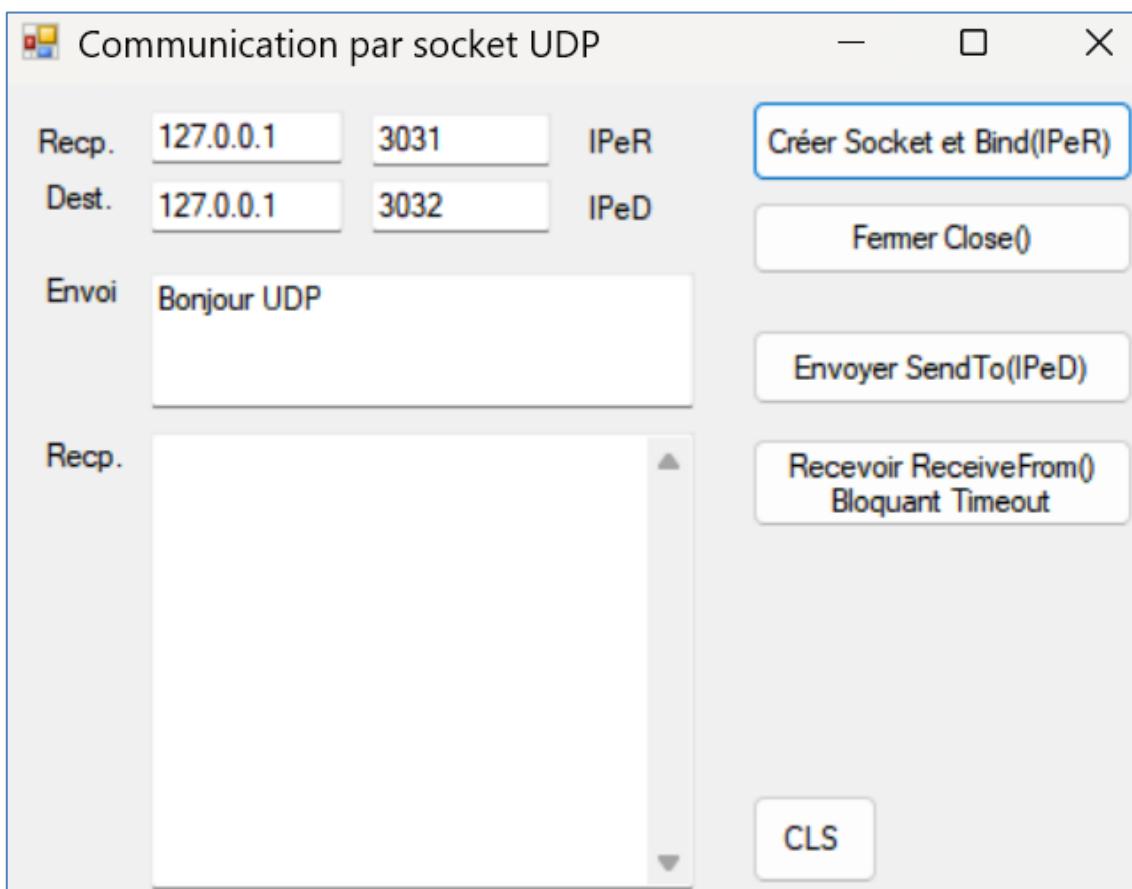


Autoévaluation

6. Socket UDP (mode non connecté)

Le but est de réaliser un programme permettant d'envoyer et de recevoir des messages (chaînes de caractères) par l'intermédiaire d'un socket UDP.

L'interface utilisateur (IHM) du programme doit ressembler à cela :



Les grandes lignes du programme sont les suivantes :

*En vert, les méthodes (**primitives**) à utiliser. L'espace de nom est System.Net.Sockets.*

Création d'une socket UDP

- Création d'un socket **Socket**
- Création du point de terminaison de l'hôte (destination) (IPeD) **IPEndPoint**
- Création du point de terminaison de local (réception) (IPeR) **IPEndPoint**
- Déclaration du point de terminaison de l'émetteur (IPeFrom) **IPEndPoint**
- Lier explicitement le Socket au point de terminaison local (réception) **Bind**

Envoi d'un datagramme

- Mise en forme du message à envoyer (msg)
`var msg = Encoding.ASCII.GetBytes ("Texte à envoyer");`
- Envoi du message à l'hôte, en indiquant le point de terminaison de la destination `SendTo`

Réception d'un datagramme

- Déclaration du buffer de réception :
`var buffer = new byte[1024];`
- Réception du message `ReceiveFrom`
- Affichage du message :
`this.textBox1.Text+=Encoding.ASCII.GetString(buffer, 0, buffer.Length);`

Fermeture du socket

- `Close`

Réalisation

- ⇒ Exécuter le projet pour vérifier que la fenêtre « Form1 » s'ouvre sans problème.
- ⇒ Changer le titre de la fenêtre « Communication par socket UDP »

Commit #2



Autoévaluation

Pour mieux comprendre le fonctionnement des sockets, chacune des actions précédentes correspondra à un bouton de l'IHM.

Pour commencer, l'application pourra envoyer et recevoir un texte. Ces textes seront présents dans deux champs (textbox).

 Note : Pour le nommage des composants, garder toujours le nom par défaut complété par son rôle. Exemple pour un bouton, changer le nom par défaut « button1 » par « buttonClose ».

Pour le codage, aidez-vous aussi de l'Annexe.

- ⇒ Déclarer les variables (handle) nécessaires en début de la classe Form1 :
`private Socket SSockUDP;`
...
- ⇒ Ajouter une textbox pour saisir le message à envoyer et une autre pour afficher le résultat de la réception.
- ⇒ Ajouter le code pour les boutons pour « Créer socket udp », « Envoyer » « Recevoir » et « Fermer socket» en suivant les étapes décrites en début de chapitre.
- ⇒ Ajouter à l'IHM le paramétrage de l'adresse IP et du port en émission et aussi en réception.

Commit #3



Autoévaluation

- ⇒ Tester le fonctionnement nominal en exécutant deux instances de votre programme. Remplir la fiche de test :

Test fonctionnel			Test N° 1	
Envoyer et recevoir un texte par une socket UDP				
Exécution du test				
N°	Démarche	Données	Comportement attendu	C/ NC
1				
2				
3				
4				
5				
6				
7				



Autoévaluation

- ⇒ Intercepter les messages d'erreurs liés au socket pour éviter le « plantage » de l'application et afficher un message compréhensible par l'utilisateur.

Exemple :

```
try
{
    SSockUDP.XXXXXX;
}
catch(System.Net.Sockets.SocketException se)
{
    This.textBoxX.Text += "Message d'erreur : "+se.ToString();
}
```

- ⇒ Tester le fonctionnement optimal et les cas particuliers suivants :
 - Créer plusieurs fois le même socket.
 - Fermer plusieurs fois le même socket.
 - Envoyer ou recevoir sans avoir créé le socket.
 - Attribuer le même point de terminaison en réception et en destination.

❖ Commit #4



Autoévaluation

Le programme utilise des **méthodes synchrones**, l'envoi est rapide et passe inaperçu dans notre programme, par contre la réception est bloquante tant qu'un message n'est pas reçu.
Solutions pour recevoir les données sans blocage :

- **Première solution : gestion d'un temps maximum d'attente en réception**

A l'aide de la méthode : `SetSocketOption` en utilisant `SocketOptionName` et `ReceiveTimeout`

- ⇒ Faire en sorte que la réception rende la main automatiquement au bout de 10 secondes.
`SSockUDP.SetSocketOption(SocketOptionLevel.Socket,
SocketOptionName.ReceiveTimeout, 5000);`

❖ Commit #5

- **Deuxième solution : scrutation de la disponibilité de données en file d'attente dans la mémoire tampon réseau, la scrutation est périodique et sera réalisée à l'aide d'un TIMER.**
- ⇒ Tester périodiquement par le « tick » d'un Timer, la présence de données reçues dans la mémoire tampon avec la méthode `Available` (du socket).

❖ Commit #6



Autoévaluation

7. Fonctionnement client/serveur (TCP)

Une des applications appelée « serveur » possède un socket associé à un port d'écoute.

Le serveur attend une demande de connexion de la part d'un « client » sur ce port.

Si le serveur accepte la connexion, il va alors créer un nouveau socket associé à un nouveau port.

Les grandes lignes du programme sont les suivantes (en vert, les méthodes (primitives) à utiliser) :

Côté client :

- Crédit d'un socket TCP/IP `Socket`
- Connexion de ce socket avec une destination (adresse IP + port TCP) `Connect`
- Communication entre le client et le serveur : envoi et réception de messages `Send/Receive`
- Fermeture du socket `Close`

Côté serveur :

- Crédit d'un socket TCP/IP `Socket`
- Lier le socket à la structure d'adressage `Bind`

- Écouter sur le socket `Listen`
 - Acceptation d'un nouveau client et création d'un nouveau socket `Accept`
 - Prise en charge du client sur le nouveau socket (réception et envoi de données) `Send/Receive`.

Serveur TCP

Vous allez créer un serveur qui renvoie aux clients leur propre message.

- ⇒ Réaliser le programme serveur, une IHM comportera un bouton de lancement du serveur et une textbox qui affichera la requête du client.
- ⇒ Utiliser la méthode `Poll` pour vérifier l'état de la socket (blocant)
- ⇒ Commenter soigneusement le code.
- ⇒ Tester votre programme avec un navigateur (comme client).
- ⇒ Afficher sur l'IHM du serveur l'adresse IP et le port du client connecté.

❖ Commit #7



Autoévaluation

Client TCP

- ⇒ Réaliser un client TCP et tester le avec le serveur précédemment réalisé.

❖ Commit #8



Autoévaluation

Observation des paquets échangés avec « WhireShark »

- ⇒ Analyser de façon détaillée les échanges lors d'une connexion entre le serveur écho et le client.

Note : Il est important de mentionner que les lignes d'une requête HTTP se terminent par les caractères "\r\n". La requête HTTP utilise une ligne vide comme indicateur de fin de requête.

⇒ Faire aussi l'analyse dans le cas des sockets UDP.

⇒ Conclure



Autoévaluation

8. Réception asynchrone de données à partir d'un Socket

⇒ Tester et commenter l'exemple de client asynchrone sur socket :

<http://msdn.microsoft.com/fr-fr/library/vstudio/bew39x2a.aspx>



Autoévaluation

9. TCPCLIENT et TCPLISTENER

Socket simplifié en TCP, hérite de Socket.

Espace de nom : System.Net.Socket :

```
TcpClient client = new TcpClient(server, port);  
  
var ipEndPoint = new IPEndPoint(IPAddress.Any, 13);  
TcpListener listener = new(ipEndPoint);
```

La communication se fait par flux :

```
NetworkStream stream = client.GetStream();
```

⇒ Tester et commenter les exemples client et serveur :

<https://learn.microsoft.com/fr-fr/dotnet/fundamentals/networking/sockets/tcp-classes>

[https://msdn.microsoft.com/fr-fr/library/system.net.sockets.tcplistener\(v=vs.110\).aspx](https://msdn.microsoft.com/fr-fr/library/system.net.sockets.tcplistener(v=vs.110).aspx)



Autoévaluation

10. Annexe

La Classe Socket

Elle prend trois arguments :

- **Le domaine** : indique le domaine de communication qui va être utilisé. Cet argument permet de sélectionner la famille de protocoles qui sera utilisée (InterNetwork).
- **Le type** : définit la sémantique des communications (Stream, Dgram).
- **Le protocole** : spécifie le protocole particulier qui sera utilisé avec le socket (traduction de socket). (Udp, Tcp).

Les sockets doivent établir une connexion avant de pouvoir être utilisées comme des flux de données. Les sockets permettent de s'assurer qu'il n'y aura ni perte ni duplication des données. Tout se passe comme si on lisait un fichier.

Déclaration :

```
private :    Socket socket ;
```

Ouverture du socket (TCP) :

```
socket = new Socket(    AddressFamily.InterNetwork,
                      SocketType.Stream,
                      ProtocolType.Tcp);

//connection sur le serveur 127.0.0.1 port 80
IPEndPoint iped = new IPEndPoint(IPAddress.Parse("127.0.0.1"), 80);

Socket.Connect(iped);
```

Fermeture du socket :

```
Socket.Shutdown(SocketShutdown.Both);
socket.Close();
```

Requête

```
// envoie la demande de page au serveur web

Socket.SetSocketOption( SocketOptionLevel.Socket,
                      SocketOptionName.ReceiveTimeout, 5000);
var messageEnvoi = Encoding.ASCII.GetBytes("GET /\r\n\r\n");

socket.Send(messageEnvoi);
```

Réception

```
var messageRecu = new byte[1024];
int nbcarrecu = socket.Receive(messageRecu);
this.textBoxReception.Text = "nbcarrecu " + nbcarrecu + "\n" +
Encoding.ASCII.GetString(messageRecu, 0, nbcarrecu);
```