INF1010 - Réseaux d'ordinateur II

Projet : Application client/server de messagerie

UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

Réalisé par:

Hyacinthe Brochu
Simon Lafrenière
Patrick Duhaime

Professeur: Boucif Amar Bensaber
Session Automne 2018

Table des matières

Introduction

Le présent document vise à présenter le travail réalisé pour le cours INF1010. Les programmes développés pour ce travail consistent en une application serveur et une application client toutes deux développées sous Visual Studio 2017 en utilisant le langage C-Sharp. Les applications utilisent le protocole TCP/IP et les .NET sockets pour établir les communications.

Objectifs et buts

Il s'agit de présenter une application inter-usagés de messagerie instantanée pour permettre l'échange de messages entre clients d'un même serveur (modèle client/serveur). Chaque client qui s'enregistre auprès du serveur avec un nom d'utilisateur unique pourra envoyer des messages aux autres utilisateurs, le client pourra aussi recevoir les messages envoyés par tous les autres utilisateurs. Au minimum, 3 clients doivent être supportés en même temps par le serveur afin de démontrer le bon fonctionnement. L'objectif de ce travail est de familiariser les participants à la communication entre processus client et processus serveur.

Méthodologie de conception

Processus serveur

Le processus serveur reçoit des messages des clients et transmet ces messages vers les autres clients. Le serveur tient à jour un dictionnaire de tous les clients (nom, adresse) afin de pouvoir compléter correctement la retransmission aux autres clients.
Lorsqu'un paquet est reçu, le serveur décapsule ce paquet de façon à récupérer les informations pertinentes (nom du client, commande pour le serveur, message client…).
Quant un client se connecte pour la première fois ou quant un client quitte, le serveur envoie la liste des clients qui sont encore connectés (Push). Si un client envoie la chaîne de caractères **« list »** (Pull), le serveur interprète cette commande comme une demande d'envoi de la liste des clients enregistrés.

Processus client

Le processus Client permet d'effectuer l'envoi des chaînes de caractères entrées par l'utilisateur aux autres clients connectés (1/ à un client, 2/ à un groupe de clients, 3/ à tous les clients) par l'entremise du serveur. Lorsqu'un paquet est envoyé, le logiciel encapsule ce paquet de façon à obtenir un paquet d'envoi conforme. Lorsqu'un paquet est reçu, le client décapsule ce paquet de façon à récupérer le nom de l'utilisateur qui a fait l'envoie et le message afin d'afficher ces informations sur l'interface du client.

Analyse, description du programme

Programme serveur

Le serveur utilise les NET sockets et écoute sur l'adresse 0.0.0.0. Dans le contexte des serveurs, 0.0.0.0 signifie "toutes les adresses IPv4 sur la machine locale". Si un hôte a deux adresses IP, 192.168.1.1 et 10.1.2.1, et qu'un serveur s'exécutant sur l'hôte est configuré pour écouter sur 0.0.0.0, il sera accessible à ces deux adresses IP. Cela permet de déplacer le serveur ou de lui assigner de nouvelles adresses IP sans changer la configuration du serveur.

Au démarrage le serveur affiche le MOTD (message of the day), MOTD est un message d'accueil du programme, généralement des serveurs, ce message identifie le serveur, donne les procédures de connexion et affiche les mentions légales d'usage, nous avons récupéré le message générique de Cisco pour les mentions légales.

Le serveur boucle ensuite pour vérifier si des connexions de clients arrivent sur le port 8080, si le serveur détecte une connexion, il lance un thread pour cette connexion dans lequel il assigne un objet TCPClient qui contient un NET socket dédié pour la communication, il ajoute le client dans le dictionnaire des utilisateurs connectés sous forme de (clé,valeur) ⇒ (utilisateur,TCPClient), le thread est fermé et les ressources libérés à la déconnexion du client.

Le nom d'utilisateur doit être unique sur le serveur, si un client tente de se connecter avec le même nom d'utilisateur qu'un client déjà existant dans le dictionnaire, la connexion sera refusée. Le serveur ne prend pas de mot de passe et ne garde aucune information à propos des comptes clients. Cependant, toutes les transmissions sont enregistrées dans le fichier TP2Chat.log.

TP2Chat.log

```
Server started  2018-12-06  12:17:01
Access granted to patx from IP: 127.0.0.1  2018-12-06  12:17:20
Access granted to Simon from IP: 127.0.0.1  2018-12-06  12:17:31
Access granted to Hyacinthe from IP: 127.0.0.1  2018-12-06  12:18:04
Message sent from: "Hyacinthe" To: "patx,Simon"  2018-12-06  12:18:25
Message body: "Allo a vous 2, ca va vous autres ?"  2018-12-06  12:18:25
Access denied to patx from IP: 127.0.0.1  2018-12-06  12:19:36
Access granted to patx2 from IP: 127.0.0.1  2018-12-06  12:22:26
Closing communication with patx2 from IP: 127.0.0.1  2018-12-06  12:22:38
```

Avec l'analyse de ce fichier nous pouvons identifier les utilisateurs, leurs adresses IP ainsi que l'heure et la date de l'action enregistrée. (connexion accepté, refusé, déconnexion, messages…)

Programme client

Le client utilise aussi les NET sockets et se connecte au nom de domaine **server.ca**. Puisque ce nom de domaine ne nous appartient pas, il sera impossible de configurer la zone DNS pour pointer vers un IP public de notre choix, puisque ce travail sera présenté dans un réseau local, il suffit d'ajouter le nom de domaine et l'adresse IP du serveur aux fichiers hosts des machines qui lanceront les clients. Sur les machines Windows le fichier hosts est situé dans le dossier C:\Windows\System32\drivers\etc\, ce fichier doit être édité avec les droits d'administrateur.

C:\Windows\System32\drivers\etc\hosts

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97         rhino.acme.com          # source server
#       38.25.63.10         x.acme.com              # x client host
       192.168.12.111       server.ca

# localhost name resolution is handled within DNS itself.
#         127.0.0.1       localhost
#         ::1             localhost
```

Sur ce fichier exemple, l'adresse IP 192.168.12.111 est l'adresse du serveur. Le hostname server.ca pointe maintenant sur le IP 192.168.12.111 pour cette machine. Avant d'interroger les serveurs DNS configuré sur la carte réseau, le système vérifie toujours le fichier host pour savoir si il connaît déjà ce nom de domaine, si une entrée existe pour le domaine, les DNS ne seront pas sollicités et l'adresse IP est récupéré à partir du fichier hosts directement.

Utiliser un nom de domaine pour effectuer la connexion du côté client permet de pouvoir déplacer le serveur ou de lui assigner de nouvelles adresses IP sans changer la configuration des clients. Il suffit de faire les changements au niveau de la zone DNS du nom de domaine.

Au démarrage, le client présente une interface de connexion et demande d'entrer un nom d'utilisateur pour se connecter au serveur, si le nom d'utilisateur est unique, la connexion est établie avec le serveur. Le client encapsule les requêtes et les messages sous forme de chaînes de caractères (String).

"Connexion;|&|;Utilisateur"

Connexion représente la commande.
;|&|; est le délimiteur de la chaîne.
Utilisateur est le nom d'utilisateur entré par le client.

Une fois la connexion accepté par le serveur, l'interface de messagerie est affiché et le client reçoit du serveur (**push**) la liste des utilisateurs connectés qui est afficher dans le panneau gauche de l'application.

"List;|&|;Utilisateur2:Utilisateur1"

List représente la commande.
;|&|; est le délimiteur de la chaîne.
Utilisateur2:Utilisateur1 est la liste des utilisateurs connectés sur le serveur.

Si un nouvel utilisateur se connecte sur le serveur, le client reçoit une commande Connexion.

Connexion représente la commande.
;|&|; est le délimiteur de la chaîne.
Utilisateur4 est le nom d'utilisateur qui s'est connecté sur le serveur.

Si le client reçoit un message d'un autre utilisateur, il reçoit la chaîne:

"Message;|&|;Utilisateur1;|&|;allo"

Message représente la commande.
;|&|; est un délimiteur de chaîne.
Utilisateur1 est le nom d'utilisateur qui envois le message
;|&|; est un délimiteur de chaîne.
allo représente le corps du message

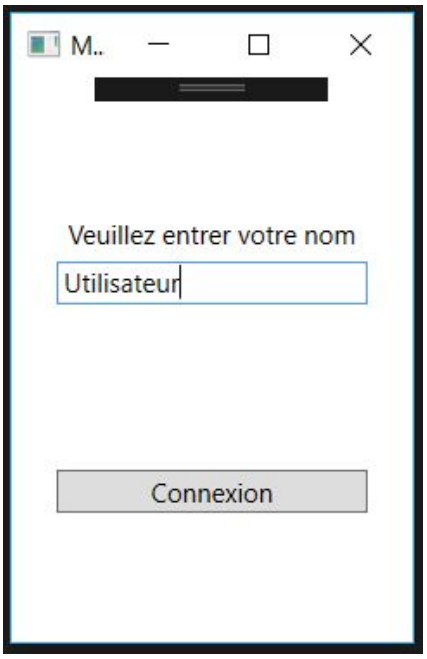
Le client encapsule les commandes envoyé au serveur avec le même format que les commandes décrites pour la réception. Le client envoi les requêtes suivantes au serveur:

Connexion demande de connexion au serveur. 1 attribut (nom utilisateur)
Message envois de message. 2 attributs (liste des destinataires, corps du message)
Liste demande la liste des utilisateurs connectés. Sans attribut
Deconnexion demande de déconnexion du serveur. Sans attribut

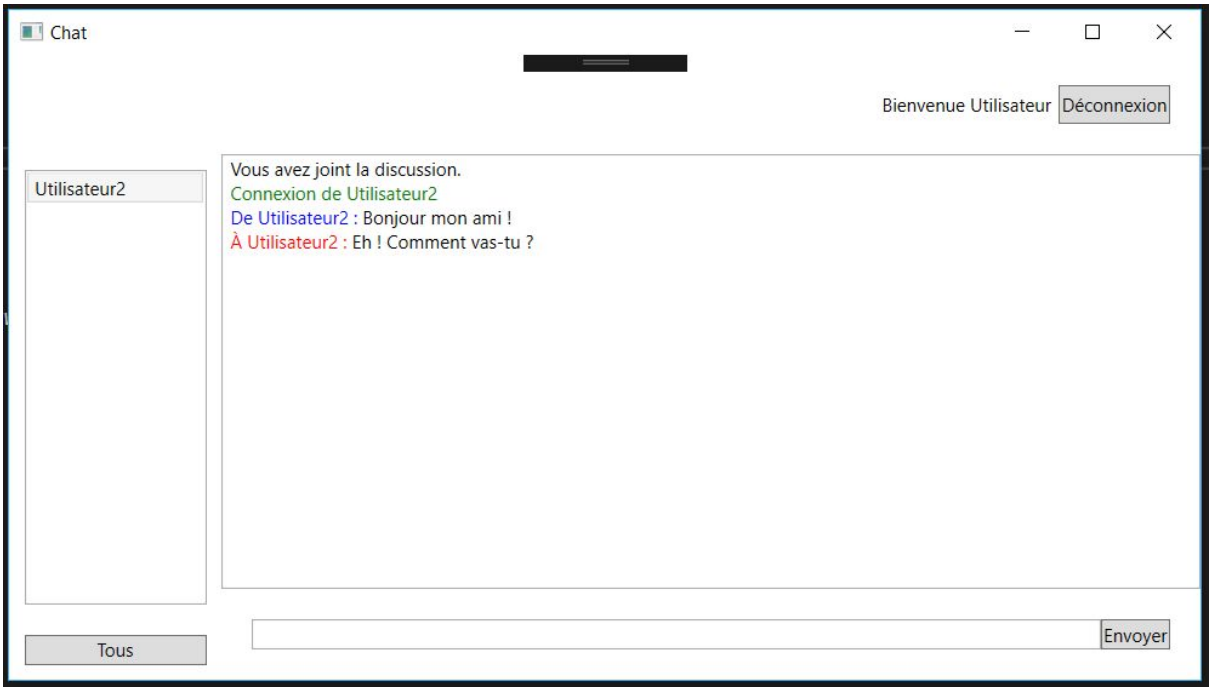
Mode d’emploi

1- Démarrez le programme du serveur puis ceux des clients.

2- Dans l’interface de connexion du client, entrez un nom. Ces derniers doivent être uniques. Un message d’erreur s'affiche donc si le nom est déjà utilisé.



3- Dans l’interface principale, sélectionnez les destinataires du prochain message dans la liste des connectés à gauche. Vous pouvez cliquer sur le bouton «Tous» pour sélectionner tout le monde.



4- Dans la barre inférieure, écrivez votre message puis cliquez sur le bouton «Envoyer».
5- Si vous rédigez le message «list», le serveur vous affichera à la place la liste des utilisateurs connectés.

Couleurs des messages affichés :
Rouge : Messages envoyés par l'utilisateur.
Bleu : Messages reçus.
Vert : Annonce de connexion et de déconnexion.
Violet : Messages système.

Discussion

Nous avons ajouté la journalisation au serveur pour démontrer que cette dernière est essentielle pour la surveillance des services, du matériel et pour pouvoir étudier les traces des activités des utilisateurs sur le système pour des fins de sécurité.

Essentiellement le fichier .log peut être ouvert et examiner de façon manuelle par un administrateur mais aussi par d'autres programmes et ce en temps réel. Cela permet de prendre des actions rapidement en cas d'attaque ou d'abus par un utilisateur.

Nous avons ajouté au fichier .log le corps des messages pour démontrer une grande faiblesse du modèle client/serveur. Cette faiblesse réside dans la confiance que l'utilisateur doit donner aux administrateurs du serveur. En effet, notre application ne garde pas de donnés des comptes clients, mais la plupart des applications client/serveur gardent ces informations, des mots de passe crypté sont associé aux utilisateurs et des comptes clients sont créés sur le serveur. Il devient alors possible de construire des profils des utilisateurs en analysant le contenu des échanges, ces profils servent ensuite de différentes façons selon le bon vouloir des administrateurs. Cette problématique combinée avec la centralisation des services internet vers quelques compagnies comme Google, Facebook, Netflix etc. ont changé le fonctionnement d'Internet et ce pas toujours en la faveur de l'utilisateur. Si on prends l'exemple de Google, selon leurs termes de services, Google recueille et utilise vos données pour vous présenter des annonces. Google prétend porter une attention particulière à la sécurité et ne pas vendre de renseignements personnels comme vos noms, adresses de courriel et coordonnées bancaires à des tiers mais cela reste impossible à vérifier. Il est possible de configurer son compte Google pour bloquer certains renseignement de la collecte automatique mais puisque le code de configuration du compte n'est pas disponible, il est impossible de vérifier si la récupération des données est réellement bloquée.

Une partie importante des activités et des revenus de Google repose sur l'affichage des annonces (publicité ciblé), tant sur les services Google que sur les sites Web et les applications mobiles qui sont associés avec la compagnie. Les annonces, selon Google,

aident à maintenir la gratuité des services pour tout le monde. L'utilisation des données est strictement pour présenter des annonces ciblé selon les choix et les goûts de l'utilisateur.

Les compagnies qui basent leurs revenus sur les données des utilisateurs finissent par considérer leurs utilisateurs comme un produit commercialisable et influe grandement sur la fonctionnement de l'Internet. Le moteur de recherche de Google qui utilise les données personnelles de ses utilisateurs ne retourneras pas les mêmes résultats de recherche pour tous les utilisateurs, les résultats sont personnalisé pour le profil du compte, ainsi, deux profils d'utilisateurs différents n'obtiendront pas les même résultats pour un même terme recherché. Le but est de présenter des résultats pour inciter l'achat des produits de commerçants qui eux auront payé pour être mieux positionné dans les résultats pour leurs publics cibles respectifs. L'appropriation et l'utilisation des données personnelles nuisent au fonctionnement et au développement de l'Internet. Le moteur de Google représente 80% des parts du marché, on peut donc affirmer que 80% des recherches sur Internet sont biaisées.

De plus en plus de gens sont sensible à l'appropriation des données et cherchent des alternatives surtout depuis la dénonciation fait par Edward Snowden ou suite aux différents scandales médiatisés comme celui de Facebook. La confiance des utilisateurs envers les grandes compagnies qui contrôlent des parts importantes des activités qui prennent place sur Internet est mise à rude épreuve. Des solutions alternatives sont de plus en plus en demande.

Conclusion

Le modèle client/serveur reste aujourd'hui le modèle le plus répandu pour le déploiement d'application en ligne. Il est un modèle fort apprécié des administrateurs systèmes car il concentre l'information en un seul point ce qui facilite grandement la maintenance des systèmes. Des solutions comme Kubernetes ont vu le jours pour palier à certains problèmes du modèle et plusieurs autres outils viendront se greffer au cours de prochaines années pour améliorer les performances générale du modèle. La centralisation et la monopolisation restent des problèmes importants, le modèle client/serveur ne semble pas offrir de solution à ce phénomène. La solution réside-t-elle dans le modèle pair à pair, ou encore dans le blockchain ?