Chatty
Simple Application de Chat Serveur-Client utilisant les Sockets TCP

Nathanael Bayard — L3 Info — Réseaux 2

9 décembre 2018

La classe `U` (pour *Utils*) contient, pour les deux versions du programme, un certain nombre de fonctions statiques d'utilité générale.

1 Version Simple : Transmission des Messages en Clair

La classe `Server` contient le code et la fonction `main` responsables du lancement du serveur. Cette fonction main attend un unique argument correspondant au port à utiliser pour les communications entre le serveur et le client. Une fois lancé, le serveur affiche quelques informations d'ordre général, puis se met en attente d'une connexion externe.

La classe `Client` est de même responsable de la partie client de la connexion. Les arguments de sa fonction `main` sont, dans cet ordre, l'adresse du serveur *Chatty* auquel se connecter, et le numéro de port correspondant.

Une fois le client lancé avec l'adresse et le port du serveur, la communication est établie. Le serveur prend la parole en premier, envoie au client un message que l'utilisateur lui donne dans le terminal, puis se met en attente d'une réponse de la part du client. Le client, lui, commence par attendre un message en provenance du serveur. Après réception de celui-ci, le client sollicite son utilisateur pour envoyer un message de réponse au serveur.

Les deux entités et utilisateurs peuvent ainsi s'échanger des messages en alternance, jusqu'à ce que l'un d'eux décide de mettre fin à la communication. Le message à envoyer pour terminer le dialogue est `‘‘bye’’` (sans guillemets). Pour le client, cela correspond à la terminaison du processus lui-même. Pour le serveur, par contre, cela correspond uniquement à la fin d'une conversation, et le programme se met alors en attente d'une nouvelle connexion par un autre client. Le serveur ne se termine que si l'utilisateur tue le programme manuellement, par exemple à l'aide du signal système envoyé depuis le terminal à l'aide de la combinaison `Ctrl-C`.

Une quatrième et dernière classe, `Main`, optionnellement remplace les autres classes comme point de départ du programme Serveur ou Client : en fonction du nom de l'exécutable (`args[0]`), la fonction `Main.main` appelle le code permettant de lancer le serveur (`chattyServer`) ou le client (`chattyClient`).

Les classes `ServerDemo` et `ClientDemo` peuvent être utilisés pour lancer une simulation basique d'utilisation de cette application : le port du serveur est pré-programmé pour être 4444, et l'adresse correspond au `localhost`, c'est-à-dire, le client et le serveur sont supposés être lancés depuis la même machine. Il faut bien sûr lancer le serveur en premier pour ne pas obtenir un échec de connexion lors du lancement prématuré du client.

2 Version Sécurisée : Transmission Chiffrée à l'aide de l'Algorithme AES

La structure de la partie “chat” ne diffère pas significativement par rapport à la version précédente.

La différence est limitée à la gestion d'une clé AES utilisée pour chiffrer les messages : `Server.main` et `Client.main` attendent un paramètre supplémentaire à la fin, donnant le nom du fichier dans lequel la clé AES est censée être stockée, encodée en base 64.

Pour créer une clé AES et l'enregistrer dans un fichier, il y a une nouvelle classe prévue pour cela : `KeyMaker`. La fonction `KeyMaker.main` attend un unique paramètre correspondant au chemin de fichier à utiliser pour sauvegarder la nouvelle clé à créer.

Les messages sont chiffrés de chaque côté à l'aide de la clé récupérée dans le fichier fourni en paramètre. Le programme affiche la version chiffrée et déchiffrée de chaque message des deux côtés. Afin d'éviter des problèmes d'encodage et de transmission des messages chiffrés, ceux-ci sont d'abord encodés en base 64 (armorés) avant d'être transmis de l'autre côté de la socket. Ils sont ensuite décodés sous la forme d'une liste de bytes avant d'être déchiffrés avec la clé AES. Le programme affiche donc cette version “lisible” en base 64 de chaque message chiffré.

Les classes `ServerDemo` et `ClientDemo` jouent le même rôle que dans la version non chiffrée. La classe `KeyMakerDemo` permet de créer une clé et de l'enregistrer dans le fichier `./aeskey.txt`. C'est ce fichier qui sera lu par les deux autres classes de la démo pour récupérér la clé de cryptage. Afin de lancer la simulation, il faut donc d'abord exécuter `KeyMakerDemo.main` pour créer la clé et son fichier, avant de finalement démarrer le serveur et puis le client, qui pourront à partir de là s'échanger des messages chiffrés à l'aide de l'algorithme AES en utilisant la clé contenue dans `./aeskey.txt`.