Projet Réseaux

Mamadou Camara
M2 CSMI

December 13, 2015

1 Utilisation

Ce projet contient deux principales classes le serveur de canal que j'ai appelé **Channels** et **Client**.
Pour démarrer le serveur de canal il suffit de se mettre sur répertoire **build** et de taper la commande:

```
.\ChaToYou server numport
```

Où **numport** est le numéro de port du serveur. De la même manière pour démarrer un client on se place sur **build** et taper la commande:

```
.\ChaToYou clientx
```

Où **x** varie entre 1 et 2 pour le moment. Un fichier de configuration contient la liste d'adresse IPv4/IPv6 à partir desquelles on crée un client ou un serveur.

2 Protocole

Le client envoie un message au serveur Channels qui à son tour, est chargé de diffuser le message aux autres clients connectés sur ce même canal. Un client est dit connecté s'il a envoyé un message au moins une fois au serveur Channels.

2.1 Structure du CANAL

Un canal est composé de l'adresse du serveur de Channels, du nom de canal et d'un tableau d'adresse des clients ayant déjà envoyé un message au serveur.

2.2 Motivation

Ce protocole peu proche du protocole IRC (RFC https://tools.ietf.org/html/rfc1459) qui met en oeuvre l'esprit de communication en groupe.

3 Envoie, diffusion

Le client initie la communication par la méthode **sendTo()** qui envoie le message au serveur de canal. Ensuite le serveur Channels récupère le message via sa méthode **lire_message()**. Le message récupéré, doit être renvoyé à tous les autres clients du même canal via la méthode **diffuser_message()**.

4 Architecture

Le projet contient ces principales classes :

- **AddrStorage**: encapsulation des adresses IP, gestion communes des adresses IPv4 et IPv6.
- **Client** : client envoie et réception de message.
- **Channels** : serveur de canal : réception diffusion de message
- **File** lecture et écriture dans un fichier, permet de prendre les adresses à partir de fichier de configuration
- **Converter**: conversion de type et découpage des chaînes de caractères
- **Exception**: gestion des exceptions
- **main**: Entrée principale
- **socket** : inclusion des librairies nécessaires et définition de macros

5 Principale couche: couche communication

Notre application est principalement constituée de la couche communication. Cette couche est entièrement gérée par la classe **AddrStorage**. Celle ci permet la gestion uniforme des adresses IPv4 et IPv6. Elle contient également les méthodes **sendTo()**,**recvFrom()**, ,**lire_message()** et **diffuser_message()**.

6 Conclusion

Ce projet m'a aidé à mieux comprendre la programmation réseaux. Toutefois, toutes les fonctionnalités demandées ne sont pas encore implémentées par manque de temps.

Remerciement

Je tiens à remercier M. Alexandre Kornmann [1] qui m'a aidé à beaucoup mieux comprendre ces travaux à partir desquels je me suis fortement inspiré.

References

[1] Alexandre Kornmann. Projet réseaux.