Projet réseau

Ce projet se divise en deux parties. Vous ne commencerez la deuxième partie qu'après avoir remis le document qui vous est demandé dans la première partie.

Première partie : rédaction de la RFC décrivant le protocole DNC (*Dog is Not a Chat*)

Dans cette partie, vous devez rédiger un document sur le modèle des RFC existantes (celle de POP3, par exemple). Il sera remis au format texte, sur le modèle des RFC et de préférence en anglais.

Le format complet des RFC est décrit par la RFC 2223, notamment sa section 3a. Voir aussi la RFC 2119 pour la signification des termes **doit**, **peut**, etc. utilisés dans la deuxième partie de l'énoncé.

Pour vous détendre, vous pouvez consulter la RFC 1149

Ce RFC doit décrire le protocole DNC permettant à des clients de se connecter à un serveur DNC et à un serveur DNC de dialoguer avec ses clients via une connexion TCP.

Un serveur DNC attend les connexions des clients, qui doivent proposer un pseudo. Si ce pseudo n'est pas déjà pris, le serveur connecte le client sous le pseudo demandé et prévient tous les clients déjà connectés que ce pseudo vient de se connecter. Lorsqu'un client se déconnecte, le serveur prévient tous les clients que ce pseudo vient de se déconnecter (avec éventuellement un horodatage, mais ce n'est pas imposé par le protocole — voir plus bas).

Les clients connectés peuvent envoyer des messages au serveur, qui se charge de les diffuser à tous les clients connectés. Cette diffusion devra contenir le texte du message et le pseudo qui l'a écrit pour que les clients puissent, s'ils le désirent, exploiter cette information (en affichant le pseudo de l'émetteur avant le message, par exemple).

Les clients connectés peuvent également envoyer des « messages de contrôle ». La RFC doit imposer les messages de contrôle suivants (dans cette liste, les termes `cmdX` seront remplacés par les mots-clés que vous aurez choisis : `cmd1` pourrait donc être `fin`, `exit` ou `quit`, par exemple). Ces termes ne sont pas sensibles à la casse (mais leurs paramètres le sont) :

- `cmd1` : le client met fin à la connexion
- `cmd2` : le client reste connecté, il reçoit toujours les messages mais ne peut plus en envoyer
- `cmd3` : le client qui était « cmd2 » redevient actif
- `cmd4` : liste des pseudos connectés.
- `cmd5` *pseudo* : le client change de pseudo

- `cmd6` *pseudo* : le client met en place un échange privé avec le client ayant le pseudo indiqué. Le destinataire (*pseudo*) peut autoriser ou non cet échange. S'il l'accepte et jusqu'au message `/cmd7` émis par l'une des deux parties, les messages entre ces deux clients ne seront plus diffusés aux autres. Un client peut mettre en place plusieurs échanges privés simultanément.
- `cmd8` *pseudo fichier* : le client indique à *pseudo* qu'il désire lui envoyer un *fichier*, le destinataire peut alors autoriser ou non ce transfert. Si le transfert est accepté, il démarre automatiquement. Lorsque les deux extrémités se sont mises d'accord, le transfert s'effectue en peer to peer, sans passer par le serveur : il faut donc que les deux parties se mettent d'accord, notamment sur le choix du port et du protocole de transport (TCP ou UDP). Le fichier doit être considéré comme du binaire (on peut transférer des images, par exemple).

Généralement, *pour la saisie*, les clients IRC préfixent les messages de contrôle par une barre de fraction (par exemple, `/away`) pour les différencier des messages « normaux ». Mais ce n'est pas à la RFC de le préciser : chaque implémentation peut choisir sa propre technique et ce qui est saisi n'est pas nécessairement envoyé sous cette forme au serveur.

Dans tous les cas, les réponses du serveur doivent être accompagnées de codes (qu'il vous appartient de définir) afin de prévenir les clients que leur requête a été acceptée, qu'il y a eu une erreur (et, éventuellement, le type de cette erreur). Ces codes doivent être classés en familles, en vous inspirant du modèle des codes HTTP, par exemple.

En outre, votre RFC doit bien préciser les formats des réponses du serveur à chaque requête des clients.

Bien entendu, il s'agit ici d'une définition minimale… Vous pouvez, par exemple, imposer un horodatage des messages, des connexions, des avatars, etc. ou rajouter des messages de contrôle qui vous sembleraient utiles.

- On insiste bien sur le fait qu'il s'agit ici d'une RFC et qu'*il n'est pas question de problème d'implémentation*.
- Les spécifications mentionnées ci-dessus sont le « minimum syndical » : vous pouvez laisser libre-cours à votre imagination pour définir bien d'autres fonctionnalités — mais attention, vous devrez ensuite les implémenter.

Deuxième partie : Implémentation

Écrivez une implémentation d'un serveur DNC **en Python** et d'un client DNC avec le toolkit graphique et le langage de votre choix, conformes à la RFC que vous avez définie.

Règles de travail :

- Vous rédigerez la RFC par groupes de 4 que vous enregistrerez auprès de l'intervenant de TP.
- Ce groupe de 4 se séparera ensuite en deux groupes de 2 **totalement autonomes** qui écriront chacun de leur côté une implémentation de cette RFC (vous aurez compris que les programmes de ces deux groupes devront donc être entièrement compatibles puisqu'ils sont censés implémenter la même RFC).
- **Deux groupes autonomes ne signifie pas deux groupes qui se partagent le même code, mais deux groupes qui développent des produits différents mais compatibles.**
- Vous n'avez pas le droit d'utiliser une API de plus haut niveau que `socket` ou son équivalent dans le langage retenu.
- Vous avez le choix pour le client : il peut être graphique (TK, GTK ou QT, ou Swing/JavaFx avec Java), comme HexChat ou mIRC, ou en mode texte, comme Irsii. Il **peut** utiliser un fichier de configuration `dnc.conf` stocké dans le répertoire personnel de l'utilisateur.
- Le serveur **doit** enregistrer dans un fichier journal (log) toutes les commandes qu'il reçoit des clients, avec horodatage de ces messages.
- Le serveur **doit** être configurable via la lecture d'un fichier `dncserver.conf`, stocké dans le même répertoire que l'exécutable, qui définira son port d'écoute, le chemin d'accès et le nom de son fichier log, etc.
- Le serveur **doit** pouvoir s'exécuter aussi bien sous Windows que sous Linux ou MacOS.
- Rien ne doit être codé « en dur » dans le client et le serveur !

Conseils :

- Commencez par écrire le serveur et testez-le (tests automatiques et/ou connexions « manuelles » avec telnet).
- Commencez par implémenter la RFC. N'ajoutez les éventuelles extensions que plus tard et documentez-les.
- Ne rendez pas une « coquille vide » en consacrant l'essentiel du développement à l'interface graphique…

Travail à rendre

Cette liste n'est pas une liste d'éléments facultatifs. Toute partie manquante provoquera un retrait de points dans la notation finale.

- La RFC au format `.txt` (**dernier délai : mardi 15 février à 12h50**)
- Le code source du client et du serveur et deux programmes exécutables, **prêts à être lancés en ligne de commande par une personne ne connaissant rien à la programmation**. Si le client est écrit dans un langage nécessitant une compilation (Java, C, C++, etc.), une documentation devra indiquer comment produire l'exécutable à partir des sources. Cette compilation ne devra pas nécessiter d'IDE…
- Une documentation permettant d'utiliser vos deux programmes
- Évidemment, les codes sources doivent être correctement documentés (commentaires, docstrings, etc.).
- La note tiendra compte de l'aspect « professionnel » (ou non…) du logiciel.