COMPTE-RENDU DU PROJET

IMPLÉMENTATIONS RÉSEAU

IRCchat

Anis **DA SILVA CAMPOS**
Valentin **CICUTO**
Robin **COLLINET**
Pierre-Emmanuel **COCHET**

Sommaire

I- Protocole

Le protocole est basé sur UDP.
Le protocole propre à notre développement est défini par des *mécanismes*, une *architecture* et un *format de trames* mis en place, définis dans ce document.

1.1. Format des trames

Nous avons défini pour des raisons d'unité et de simplification un format de trames standardisé pour tous les échanges, entre le serveur et les clients.

Structure des trames:

4 octets (int)	4 octets (int)	4 octets (int)	4 octets (int)	500 octets (string)	4 octets (int)	4 octets (int)
ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON

- ID_OP : correspond à l'id d'une commande, commun au client et au serveur. En fonction de sa valeur, les actions effectuées seront différentes;
- ID_SEQ : identifiant unique d'une séquence de trames pour un client donné;
- NB_TRAM : nombre de trames dans la séquence;
- NUM_TRAM : numéro de la trame (ordonnée) pour la séquence;
- DATA : contiendra l'essentiel des données;
- ID_USER : id du client source
- ID_SALON : id du salon pour lequel est destiné le message.

Les différents ID_OP sont définis dans le tableau ci-dessous et demandent pour certains des arguments, placés dans le champ DATA.

ID_OP	Nom	Argument
0	connect	[username]
1	connectOk	[id attribué]
2	connectUserRefuse	
3	connectNumberRefuse	
4	join	[nom du salon]
5	joinOk	[id du salon]
6	joinRefuse	
7	disconnect	

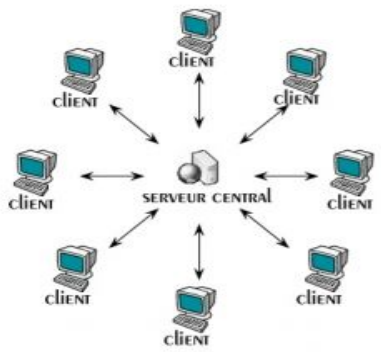
8	leave	[nom du salon]
9	liste	[ ] | [-s] | [-c] [salon]
10	say	[texte à partager]
11	sayOk	
12	sayError	
13	echo	[texte à faire afficher]
14	errorCommande	
15	heartbeat	
16	leaveOk	
17	verify	

1.2. Architecture et mécanisme

Communication :

Le principe d'un chat est la communication entre plusieurs utilisateurs.
Ici d'un point de vue utilisateur, c'est ce qu'il se passe.

En réalité, tous les utilisateurs (clients) se connectent en UDP à un seul et même serveur (serveur). Les clients ne communiquent qu'avec le serveur. C'est le serveur qui redistribue les messages aux clients concernés: les clients communiquent donc.



Persistance :

Pour assurer la persistance de la connexion, le serveur et les clients envoient un heart-beat respectivement aux clients et au serveur, en correspondance avec le principe de communication susmentionné.

Toutes les 30 secondes, ils envoient un message.

- Si au bout de 15 minutes, le serveur n'a pas eu de heart-beat de la part d'un client, le serveur déconnecte l'utilisateur correspondant.
- Si au bout de 15 minutes un client n'a pas de heart-beat de la part du serveur, il considère que le serveur est down, et affiche un message d'erreur.

Acquittement :

Bien que nous utilisions UDP, nous avons mis en place un système de connexion et d'acquittement, afin d'assurer le bon relai des informations.

Le principe est le suivant : chaque client peut envoyer des informations au serveur par le biais des trames. Mais avant de les considérer effectives, le client attend la réponse d'acquittement du serveur.

C'est donc le serveur qui acquitte au client la bonne réception de son message.

Le client lui n'acquitte pas la réception des messages du serveur. Le serveur envoie donc sans se soucier de la bonne réception de la part de ses clients. Mais bon, c'est UDP, il fallait s'y attendre...

1.3.Trames envoyées par le client

Le client a donc des fonctions qui lui permettent de communiquer et d'envoyer des demandes au serveur.
Pour envoyer une commande (texte interprété comme une commande) il faut que l'utilisateur entre dans son terminal la commande (en minuscule) précédée de "**:**".
Par exemple **:join salon**

Parmi la liste générale, du côté client, on utilisera exclusivement :

- Connect
- Disconnect
- List
- Join
- Leave
- Say
- Heartbeat

Connect :

Par exemple, l'utilisateur "Pec58" se connecte au serveur. La trame suivante sera envoyée au serveur via le socket à l'adresse IP choisie.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
0	1-65535	1-65535	1-65535	"Pec58"	-1	-1

Disconnect :

Sert à se déconnecter du serveur. :disconnect
Déconnecte l'utilisateur 10 qui a tapé la commande, de tous les salons dans lesquels il était connecté.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
7	1-65535	1-65535	1-65535	null	10	-1

Le client se ferme.
Le serveur affiche dans tous les salons où il était connecté :
"* Pec58 à quitté le salon #salon "

Liste :

Sert à lister les salons disponibles ou les utilisateurs connectés au salon :list -s -c [salon]
-s : affiche la liste des salons disponibles sur le serveur.
-c [name] : affiche les utilisateurs connectés dans le salon "name" sinon tous les utilisateurs.
REMARQUE : si list -s par défaut avant "join" et list -u "salon" après ":join salon"

Par exemple, l'utilisateur Pec58 dont l'id_user est 10 demande des informations sur les connexions au salon CPE dont l'id_salon est 26.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
9	1-65535	1-65535	1-65535	"-s"	10	-1
9	1-65535	1-65535	1-65535	"-c"	10	-1
9	1-65535	1-65535	1-65535	"-c CPE"	10	26

Join :

Le client 10, Pec58, veut rejoindre le salon CPE :

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
4	1-65535	1-65535	1-65535	"CPE"	10	-1

Le serveur annoncera à tous les clients du salon, l'arrivée du nouvel utilisateur.

Leave :

Sert à quitter seulement le salon précisé. :leave BDSM

L'utilisateur est n'est plus référencé comme appartenant au salon. Il ne recevra plus les messages de ce salon.

Client 10 veut quitter le salon 7

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
8	1-65535	1-65535	1-65535	null	10	7

Le serveur répond a tous :
" Pec58 à quitté le salon #BDSM "

Say :
Sert à communiquer un message sur un salon.

:say "Hello world."

Cette commande peut être utilisée implicitement, c'est la commande par défaut dans un salon.

Hello world.

Il se passe que le client 10 dit bonjour au salon numéro 44.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
10	1-65535	1-65535	1-65535	"Hello world.\n"	10	44

Le serveur enverra un acquittement ainsi qu'un message à tous les utilisateurs du salon 44:
#CPE <Pec58> Hello world.

Heartbeat :

Le heartbeat sert à montrer au serveur que le client est bien actif, même s'il n'envoie pas de message. Un thread dédié envoie donc un heart-beat (battement de coeur) au serveur à intervalles réguliers, définis par FREQ_HEART.

1.4. Trames envoyées par le serveur

Le serveur lui, envoie des acquittements, des messages d'erreurs et les messages des salons, soit à un seul utilisateur, soit à tous ceux d'un salon.

Ce qui correspond aux commandes suivantes du protocole :

- Connectok
- Connectuserrefuse
- Connectnumberrefuse
- Joinok

- Joinrefuse
- Sayok
- Sayerror
- Errorcommande
- Echo
- Heartbeat

ConnectOk :

Cette commande sert à acquitter un client de sa bonne connexion au serveur. Il renvoie par la même occasion l'ID_USER qu'il lui a attribué. La trame est renvoyée seulement à l'utilisateur qui avait envoyé **:connect**

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
1	1-65535	1-65535	1-65535	"* Pec58 Bienvenue. Connexion au serveur effectuée avec succès"	0-49	-1

Pour que la connexion soit ok, il faut que le nombre d'utilisateurs ne dépasse pas la limite et à fortiori, que l'IP et le port aient permis la bonne réception du précédent message.

ConnectUserRefuse :

Cette commande est une réponse négative, à la demande du client de connexion. Ici c'est parce qu'il y a un nom d'utilisateur déjà utilisé.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
2	1-65535	1-65535	1-65535	"*Désolé le pseudo Pec58 est déjà utilisé sur ce serveur…"	-1	-1

Le client recevant ce message sera invité à rentrer un nouveau pseudo, avant d'essayer à nouveau de se connecter.

ConnectNumberRefuse :

Cette commande est une réponse négative, à la demande du client de connexion. Ici c'est parce que le nombre maximum d'utilisateurs est atteint.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
3	1-65535	1-65535	1-65535	"*Désolé, vous ne pouvez pas vous connecter pour l'instant à ce serveur. Le nombre max d'utilisateurs est atteint"	-1	-1

JoinOk :

Cette commande sert à acquitter un client de sa demande d'entrée dans un salon. Cela signifie que le serveur lui enverra par la suite tous les messages qui circulent sur le salon. Le serveur lui envoie lors de cet acquittement le nom et l'id du salon.

Par exemple le serveur accepte que le client Pec58 dont l'identifiant est 10, rejoigne le salon CPE dont l'id est 26. La trame sera:

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
5	1-65535	1-65535	1-65535	"CPE"	10	26

Un message sera envoyé juste après, à tous les utilisateurs sur ce salon, du type:
"#CPE *Pec58 a rejoint le salon."

JoinRefuse :

Cette commande est une réponse négative, à la demande du client de joindre un salon. La raison peut être par exemple que le nombre max de clients est atteint sur le salon, ou que le salon n'existe pas...

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
6	1-65535	1-65535	1-65535	"* Désolé mais vous ne pouvez pas vous connecter au salon #RTFM."	10	-1

SayOk :

Cette commande sert à acquitter un client de la bonne réception de ce qu'il voulait dire et que son message va être diffusé.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
11	1-65535	1-65535	1-65535	null	0	44

Rien ne s'affiche, mais comme l'utilisateur est sur le salon, il devrait recevoir le message que le serveur envoie à tous les utilisateurs du salon. Cf **:echo**

SayError :

Cette commande est une réponse négative, à la demande du client d'afficher un message sur un salon. Il survient quand un utilisateur veut faire un **:say** en dehors de tout salon.

ID_O	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON

P						
12	1-65535	1-65535	1-65535	“*Erreur de diffusion du message”	10	-1

ErrorCommande :

Cette commande est une réponse négative, à la demande du client d’effectuer une commande. Il surgit quand le client envoie une commande inconnue.
Par exemple, Pec58 envoie **:clean CPE**

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
14	1-65535	1-65535	1-65535	“*Cette commande n’existe pas.”	10	44

Echo :

Cette commande est la commande la plus importante du serveur. Elle sert à diffuser aux différents salons, les messages reçus de la part d’un utilisateur connecté au dit salon.

Si Pec58 avait dit sur #GTFO “Fallait me les montrer avant...”.

ID_OP	ID_SEQ	NB_TRAM	NUM_TRAM	DATA	ID_USER	ID_SALON
13	1-65535	1-65535	1-65535	“#GTFO <Pec58> Fallait me les montrer avant...”	1-50	66

Tous les utilisateurs de #GTFO verront: “#GTFO <Pec58> Fallait me les montrer avant…”

Heartbeat :

Le heartbeat sert à montrer aux clients que le serveur est toujours “up”, même s’ils n’envoie pas de données. Si un client ne reçoit plus de heart-beat depuis trop longtemps, il considère avec effroi qu’il n’est plus connecté… Il se déconnecte.

II- Partie implémentée

3.1. Dans le client

Le client a les nombreuses fonctionnalités implémentées.

Tout d’abord il peut effectuer une connexion à un serveur.

Une fois le pseudo et l’adresse IP du serveur entrée, les trames envoyées au serveur permettent d’initialiser une connexion qui est suivie et persistante.

La réciproque est aussi implémentée, le heart-beat et le timeout.

Le client a un thread dédié à l'envoie d'un heart beat au serveur pour lui manifester son existence. Le client détecte quand le serveur n'envoie ni heart-beat ni acquittement, et en conclue que le serveur n'est plus connecté.

Le client peut aussi rejoindre un salon

Le client par le biais de la fonction **:join** peut rejoindre un salon afin de diffuser ses messages à tous les utilisateurs du salon et recevoir tous les messages du salons.

La dernière mais pas la moindre, l'envoie de messages

Par la fonction **:say**, le client peut envoyer ses messages au serveur pour que celui ci les diffuse.

Donc globalement du coté client, les fonctionnalités sont au rendez-vous, afin de permettre la communication via le chat.

3.2. Dans le serveur

Le serveur lui, permet d'effectuer les actions demandées par le client.

Il interprète les commandes reçues

Le serveur peut interpréter les données reçues et comprendre de quelle commande il s'agit et donc d'effectuer une action correspondante.

Envoyer des acquittements

Le serveur renvoie systématiquement des acquittements aux clients, ce qui permet au client de savoir que le message est bien arrivé au serveur.

Gérer les listes d'utilisateurs

Le serveur enregistre les clients qui joignent un salon et les ajoutent à la liste de diffusion des messages du salon.

Diffuser les messages

Le serveur peut diffuser à tous les utilisateurs connectés d'un salon les messages du salon. Les utilisateurs voient tous les messages postés sur leur salon.

3.3. Limitations

Comme tout produit qui démarre, l'IRCChat présente des limitations :

- Le taille maximale des messages: La taille d'un message est limité à une ligne, dans le sens où l'utilisateur quand il appuie sur Entrée, valide sa commande, et ne peut donc pas revenir à la ligne.
- Le nombre maximum d'utilisateurs : nous avons décidé pour des raisons de facilité en mémoire de limiter le nombre d'utilisateurs en simultané à 50. Ainsi nous utilisons un tableau[50] au lieu d'une mémoire dynamique.
- Pour l'instant il existe des salons de bases crées par le serveur (une quinzaine) mais un client ne peut pas créer de nouveaux salons ou changer le nom d'un salon. Cette fonctionnalité aurait pu être implémentée si le temps alloué avait été plus long.

3.4. Comment compiler et lancer

Pour ce qui est de la compilation, nous avons utilisé CMake, qui génère le makefile en faisant appel à tous les fichiers nécessaires.
Les paramètres sont définis dans le CMakeLists.txt.

Pour générer le makefile et compiler, il faut entrer les commandes suivantes dans le terminal UNIX:

```
$ cmake ./CMakeLists.txt
$ make
```

Il faut que les sources soient bien présentes:

```
client.c    client.h    serveur-udp.c    CMakeLists.txt    protocol.h
```

Ensuite pour lancer le chat, il faut lancer d'une part l'exécutable du serveur sur une IP joignable[1] et où le port 1500 est ouvert.

```
$ ./ServeurIRCChat
```

Et finalement les clients voulant chatter via IRCChat, lanceront l'exécutable du client:

```
$ ./ClientIRCChat
```

III- Projets d'amélioration

4.1. Vérification des trames

Comme vous avez pu le constater, les trames présentes des champs qui permettraient de vérifier que les trames sont bien arrivées et sont dans le bon ordre.

ID_SEQ	NB_TRAM	NUM_TRAM

Cette méthode n' a pas pu être implémentée. Et cela convient pour l'envergure actuelle du projet. Quand notre chat aura vocation a être utilisée massivement, cette fonction devrait être finalisée.

4.2. Buffer dédié par salon

L'aspect graphique et pratique est une valeur auquelle nous attachons de l'importance. C'est pourquoi nous espérons améliorer cette partie. Pour cela il faudrait mettre en place du côté client, un buffer dédié par salon, qui enregistre tout ce qui arrive pour ce salon, et quand l'utilisateur change de salon, le buffer correspondant au salon demandé s'affiche.

4.3. Fin

Le projet s'est bien passé. L'ambiance et la motivation du groupe était bonne, ce qui nous a permis de nous mettre d'accord sur l'architecture et le protocole assez rapidement, et par la suite de répartir les tâches.

[1] Joignable par les futurs clients.

Le projet nous a beaucoup plut, même nous aurions aimé plus de temps pour ce projet, qui représente quand même une partie applicative vraiment intéressante, alliant le C et le réseau. Nous n’avons pas vraiment eu le temps d’approfondir.