1) Introduction

a) Mode Binaire

l'introduction du mode binaire permet l'utilisation d'une représentation plus efficace des messages échangés entre les utilisateurs. Chaque type de message (dans le cas de ce programme HELO et QUIT) est codé sur un octet, permettant une transmission plus rapide. Ce choix est motivé par la nécessité de réduire la taille du message transmis lors de l'ouverture ou de la fermeture de la communication.

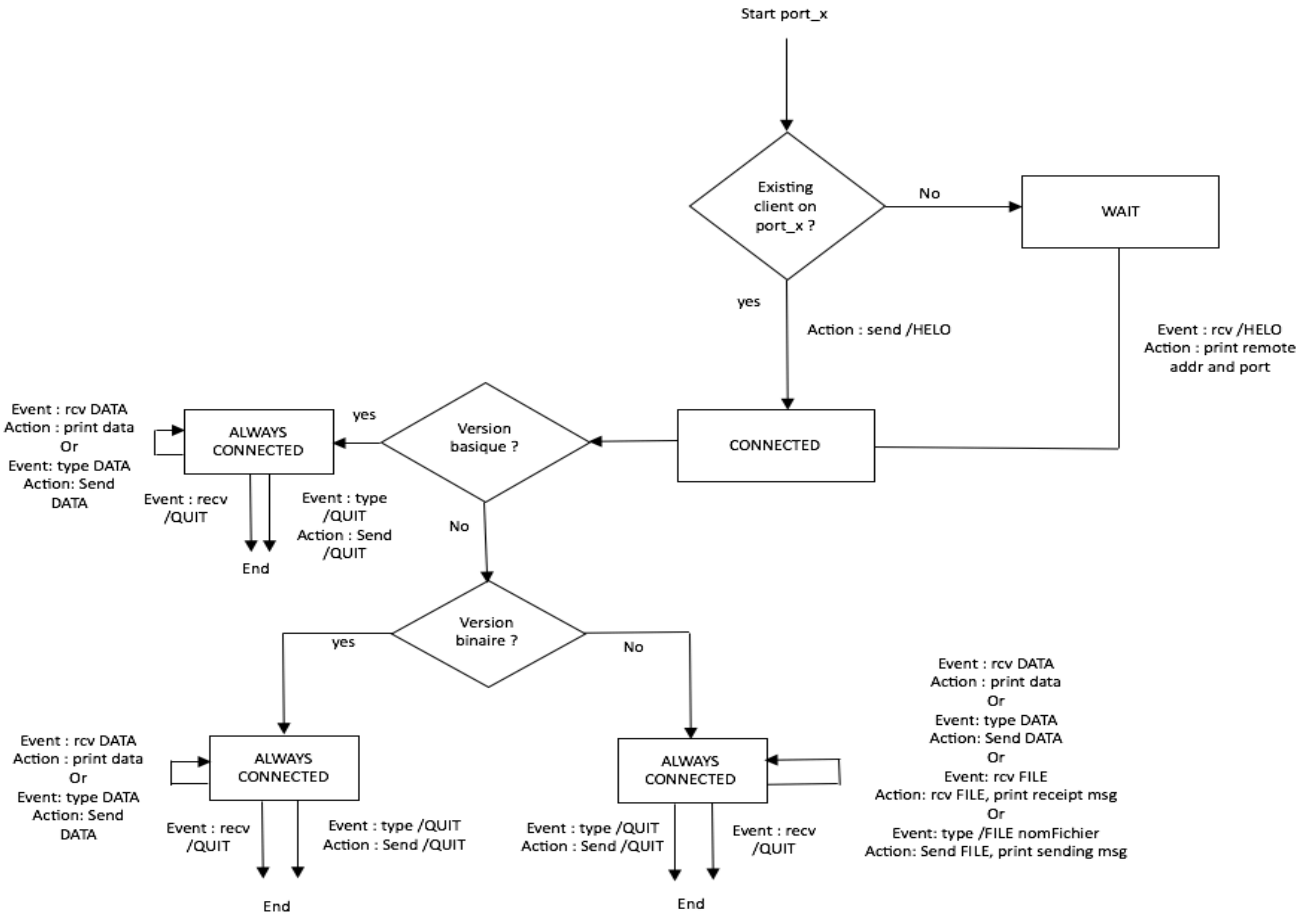
b) Transfert de fichier

Le transfert de fichier est ajouté à ce programme pour permettre aux utilisateurs de pouvoir s'envoyer des fichiers textes et binaires sans perte d'informations. Étant donné que le protocole utilisé est UDP et n'étant pas un protocole orienté connexion comme TCP, il était nécessaire dans l'implémentation de cette partie du programme de tenir compte de ce problème et de s'assurer d'un transfert de données correctes et complets.

c) Gestion avancée des utilisateurs

La gestion avancée des utilisateurs permet la connexion de plusieurs utilisateurs, la version basique ne prenant en charge que le dialogue entre 2 utilisateurs maximums, cette version offre une variante plus large sur le nombre de clients connectés.

2) Machine à états finale



3) Mode Binaire

Dans le mode binaire les messages HELO et QUIT sont représentés chacun par un seul octet(8bits). J'ai définit deux constantes HELO et QUIT et par convention initialisée respectivement à 1 et 0.

Lors de la compilation avec la macro BIN, seules les parties définies par cette macro seront prises en charge, notamment la représentation binaire des messages, le format des champs, et d'autres aspects du protocole qui diffèrent de la version texte. Plutôt que d'envoyer les messages texte /HELO et /QUIT en dur, la version binaire transfère ces messages sous une forme d'un octet chacun.

4) Transfert de fichier

Dans l'implémentation du code, je suis partie du principe qu'avant chaque envoi de fichier l'utilisateur notifie à son correspondant l'envie de lui transférer un fichier en envoyant au préalable la commande "/FILE" suivit du nom du fichier à envoyer (ex: /FILE fichier_a_envoyer). A la réception de ce message , l'interlocuteur sait qu'il doit recevoir un fichier et prépare donc un fichier de réception pour effectuer l'écriture (la copie).

Le fait est que j'ai réfléchi à plusieurs manières de recevoir le fichier, la première était de créer un nouveau répertoire et stocker le fichier reçu ainsi évitant un conflit de nommage, la deuxième option celle que j'ai choisi était d'ajouter au nom du fichier un suffixe "_recu" pour éviter ce conflit de nom et donc si le fichier à envoyer s'appelle "fichier" le fichier à recevoir prendra toujours la base du nom et rajoutera le suffixe on obtient "fichier_recu" comme nom.

Pour gérer la bonne transmission des erreurs et s'assurer de la bonne réception du fichier dans son entièreté, j'ai utilisé la retransmission en fonction des ACK recu, c'est à dire que lorsque l'émetteur du fichier envoie un paquet en fonction de la taille de mon buffer "msg" que j'ai defini a MAX_MESSAGE_SIZE 1024 , envoie en même temps le numéro de séquence du paquet. Du cote récepteur il compare le numéro de séquence recu et de celui attendue si identique on incrémente la prochaine séquence attendue et on écrit le paquet recu dans le fichier, dans le cas échéant renvoyer l'ACK attendu pour que l'autre puisse ramener le paquet perdu. Pour sortir de la boucle de transmission on compare la taille totale des données envoyés et la taille des données reçus.

5) Gestion avancée des utilisateurs

J'ai réussi à mettre en place la connexion des utilisateurs. Lorsqu'un utilisateur se connecte, l'initiateur de la connexion reçoit un message /HELO de chaque utilisateur connecté, puis affiche l'adresse IP et le port de chacun.

Cette partie n'est pas présente sur le schéma, car elle n'est pas encore implémentée dans sa version finale. Il me reste à mettre en œuvre l'envoi des messages sous forme de diffusion, de sorte que tous les utilisateurs puissent recevoir le message. De plus, je dois encore ajouter la fonctionnalité de sortie du programme avec la commande /QUIT.

6) Conclusion

En conclusion, ce projet a permis de mettre en œuvre une application de chat en utilisant des sockets UDP en C. j'ai réussi à établir la connexion entre utilisateurs avec l'envoi du message /HELO, affichant ainsi les adresses IP et les ports des utilisateurs connectés et également le transfert de fichiers entre 2 utilisateurs.

Cependant, certaines fonctionnalités restent à implémenter sur la gestion avancée des utilisateurs, notamment la diffusion des messages à tous les utilisateurs et la gestion de la sortie du programme avec la commande /QUIT.

Les défis majeurs rencontrés au cours du projet étaient liés à la gestion des sockets UDP et à la synchronisation des messages entre les utilisateurs. Bien que j'aie réussi à récupérer le tableau des adresses IP des utilisateurs connectés, j'ai rencontré des difficultés pour envoyer effectivement les messages. Malheureusement, ces obstacles ont persisté jusqu'à l'échéance du rendu.

Malgré ces défis, le projet a constitué une expérience pratique inestimable dans le développement des sockets réseau en C. Il reste des améliorations à apporter pour rendre mon programme plus robuste et fonctionnelle, mais cette expérience a joué un rôle essentiel dans l'acquisition de connaissances dans le domaine des communications réseau, un secteur dans lequel je nourris des ambitions professionnelles.