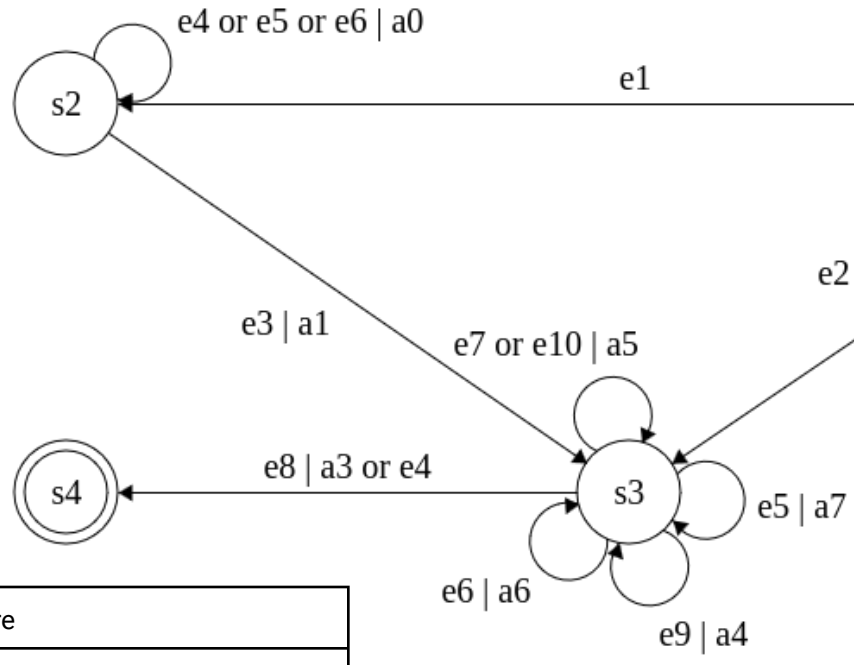
Rapport

Projet Algorithmes des réseaux - L3 Informatique

1. Introduction

Les fonctionnalités que j'ai réussi à ajouter sont le protocole en mode binaire et le transfert de fichiers.
Le protocole en mode binaire permet de réduire la taille des informations de contrôle : *0* - pour */HELO, 1* - pour */QUIT*, *2* - pour */FILE* (commande pour le transfert de fichiers). L'utilisateur peut toujours taper /QUIT ou /FILE. Les commandes seront traduites lors de l'envoi du message.Pour les messages, le format ne change pas. Si pas de commande en-tête, alors c'est un message à afficher de type DATA.
Pour le transfert de fichiers, la commande que l'utilisateur doit taper, a la syntaxe suivante: `/FILE <chemin/vers/le/fichier>`

2. Machine à états finale



a0	ignore
a1	Print remote address and port.
a2	Send /HELO or 0
a3	Send /QUIT or 1
a4	SEND /FILE or 2
a5	Send DATA
a6	Print received DATA
a7	Save received file

s1	Client START
s2	Client WAIT
s3	Client CONNECTED
s4	Client DISCONNECTED

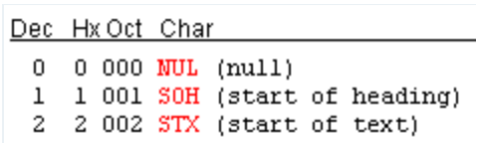
e1	No existing client on port_x
e2	Existing client on port_x
e3	Receive /HELO or 0
e4	Receive /QUIT or 1
e5	Receive /FILE or 2
e6	Receive DATA
e7	Input /HELO
e8	Input /QUIT
e9	Input /FILE
e10	Input DATA

3. Mode binaire

Pour les commandes, le programme envoie des caractères sur 1 octets. Les caractères correspondent aux valeurs ASCII : 0, 1 et 2.

A la place d'envoyer /HELO au début, le programme va envoyer 0, pour /QUIT - 1 et pour /FILE - 2. Comme indiqué dans l'introduction, si le message reçu ne contient aucune de ces commandes alors le message est de type DATA.

Cela n'est pas forcément excellent, car si un client envoie les caractères correspondant aux valeurs entières 0, 1, 2, ils pourraient être interprétés comme étant des commandes. En ASCII, ce sont des caractères de contrôle.



Pour résoudre ce problème, il faudrait utiliser un type de message pour les messages de type DATA et le vérifier lui aussi.

4. Transfert de fichiers

Si un utilisateur souhaite envoyer un fichier à un autre utilisateur, il doit taper la commande : `/FILE <chemin/vers/le/fichier>.` Le programme va vérifier que ce fichier existe et qu'il peut y accéder. Après il va envoyer des messages qui contient les champs suivants : `messageType | fileName | payloadSize | content`.

Champs		**Mode texte**	**Mode binaire**
`HEADER`	`messageType`	6 octets	1 octet
	`fileName`	Taille non-limitée	Taille non-limitée
	`payloadSize`	3 octets	3 octets
`content`		BUFSIZ - taille du `HEADER`	BUFSIZ - taille du `HEADER`

Le champ `fileName` doit permettre au programme d'identifier de quel fichier il s'agit et de le créer et d'y ajouter le contenu qui arrive par la suite. Un autre problème serait l'existence d'un fichier du même nom dans le répertoire de l'utilisateur récepteur. Le programme va modifier ce fichier en y ajoutant les informations reçues. Il y a un problème potentiel avec la taille du nom de fichier qui n'est pas limitée.

Le champ `payloadSize` permet de travailler avec les tableaux de caractères contenant des octets nuls. Quand c'est une chaîne à lire, c'est utile, mais dans le cas d'un fichier, on veut tout lire et écrire, même si ce sont des octets nuls.

Comme le programme utilise UDP, l'implémentation de transfert de fichier est peu fiable. L'utilisateur qui envoie reçoit un message de confirmation d'envoi avec succès, mais il ne sait pas si le fichier a été vraiment bien reçu par l'autre utilisateur.

5. Gestion avancée des utilisateurs

Ce programme ne comporte pas de gestion avancée des utilisateurs.

6. Conclusion

Mon implémentation a beaucoup de limites. Avec UDP, il est possible que les messages se perdent, qu'ils n'arrivent pas dans le bon ordre. Il faudrait ajouter un numéro de séquence pour tous les messages envoyés (soit texte, soit fichier).

J'ai déjà expliqué quelques limitations de la fonctionnalité de transfert de fichiers dans la section 4. Cela concerne l'interaction avec l'utilisateur et la prise de décision, certaines cas qui devraient être pris en compte : le programme écrit dans un fichier déjà existant, ou il crée un nouveau fichier avec un nom de type `<filename>xxxx` et sauvegarde quelque part la séquence `xxxx`, ou il arrête d'écrire et envoie un message de feedback à l'autre utilisateur l'informant que l'autre client a reçu mais a refusé le fichier.

Il manque l'implémentation de la fonctionnalité de gestion avancée des utilisateurs. Cela n'est pas si difficile à mettre en place (mais mauvaise gestion de temps) : il faudrait qu'un des clients devienne un serveur au moment où plus d'un client se connecte au même port. Il doit sauvegarder certaines informations identifiant chaque client et envoyer un message d'un client à tous les autres à partir de ces informations.