UNIVERSITÉ LIBRE DE BRUXELLES
Faculté des sciences
DÉPARTEMENT D'INFORMATIQUE

Titulaire : Joël Gossens
Cours : INFO-F-201 – Systèmes d'exploitation

PROJET 2 - PROGRAMMATION SYSTÈME

Hakim Boulahya – 000391737

Année académique 2014 – 2015

Table des matières

Introduction

Dans le cadre du cours de systèmes d'exploitation il nous été demandé de réaliser un projet implanté en C. Ce projet est un chat entre différents utilisateurs le permettant de communiquer via le réseau. Ce projet est divisé en deux programmes : le programme serveur, et le client. Le problème est le suivant : le serveur, tournant sur une machine, doit réceptionner les differentes connexions des clients. Ceux-ci, se connectant au serveur via l'IP ou le nom de la machine éxecutant le serveur, doivent fournir leur surnom au serveur pour ensuite envoyer différents messages. Le serveur devra ensuite faire en sorte que tous les clients qui lui sont connectés réceptionnent le message qu'un client connecté veut envoyer à tous les autres.

Les sources sont divisées en trois grande parties : L'application serveur nommée `chappServer` qui contient un main, et un module - nommé `chat_server`, l'application client nommée `chapp` qui suit le même principe que l'application serveur - dont le module se nomme `chat_client`, et un module commun aux deux dernières qui contient les appels systèmes nécessaires au bon fonctionnement des deux programmes nommé `chat`.

Dans ce rapport, il est expliqué les solutions aux problèmes rencontrés lors de l'implantation du serveur et du client.

1 Serveur

1.1 Principe

Le serveur est le programme qui récupère les connexions des clients, et permet a de réaliser une communications entre ceux-ci. Pour permettre cette réalisation le programme réalise les instructions suivantes : après la création d'un socket principal et d'une liste de socket, qui contiendra les autres sockets nécessaires à la communication entre le serveur et chacun des clients, on va écouter sur le socket principal les connexions des clients de façon non limité. Dés qu'un client se connecte, repérable via l'appel système `accept`, nous allons lancer un *thread* pour la gestion de ce client.

1.2 Thread vs fork

La raison pour laquelle nous lançons un *thread*, est qu'il ne faut pas que le serveur ne soit pas capable d'accepter les connexions d'autres clients. Si nous ne lançons pas de *thread* pour chaque client, le serveur ne sera plus capable d'accepter une nouvelle connexion tant que le client, dont aucun thread de gestion n'a été lancé, ne s'est pas déconnecté du serveur. La raison pour laquelle aucun *fork* n'est effectué est que, contrairement à un *thread*, il ne partage pas la même zone mémoire *i.e.* le *heap* du processus qui le créée, le processus fils possède sa propre zone mémoire.

Ce partage de mémoire est nécessaire lorsqu'un client se déconnecte. En effet, le *thread* de gestion du client se doit de supprimer le socket du client associé après sa déconnexion de la liste pour que le serveur puisse dans tous les cas garder à jour sa liste des sockets de clients connectés et éviter d'envoyer des informations à un socket fermé.

1.3 Thread de gestion d'un client

Le *thread* de gestion d'un client, ou `sharing_thread` dans le programme, s'occupe de demander a son client son *nickname*. Dans le cas ou un problème de connexion survient, déconnexion

ou tout autres choses, le *thread* affiche que le client s'est déconneté avant d'avoir donné son surnom. Dans l'autre cas, l'ajout du socket client dans la liste et un appel a la fonction *handle_client* sont effectués.

La fonction `handle_client` réceptionne les messages que le client envoie, tant qu'il est connecté. Après réception de chaque message, nous envoyons celui-ci à chaque client, *i.e.* que nous faisons un appel système `send` pour chaque socket présent dans la liste des sockets de client. Dés qu'un client se déconnecte, son socket se ferme, et une suppression du socket du client déconnecté est effectuée dans la liste.

2 Client

2.1 Principe

Le programme client est l'exécutable qu'un utilisateur du chat va utiliser pour communiquer avec les autres utilisateurs. La première chose que le programme va effectué, est la construction du socket. Ensuite, via l'appel système `connect`, une connexion au serveur est demandée. Pour savoir vers quel serveur se connecter, l'utilisateur doit fournir, en paramètre du programme, le nom de l'hôte du serveur. Si la connexion est réussie, un message d'acceuil est affiché, ainsi que la demande de saisie du *nickname* qui sera ensuite envoyé, via l'appel système *send*, au serveur. Lorsque le surnom est envoyé, deux choses distinctes s'effectuent : le démarrage d'un *thread* de réception, et la boucle de saisie des messages.

2.2 Thread de réception

Le `thread` de réception s'occupe d'écouter sur le socket toutes les informations que le serveur envoie au client. Il affiche chaque message réceptionné ainsi que l'heure à laquelle celui-ci a été reçu. Dans le cas où il n'est plus possible de receptionner des messages venant du serveur suite à un problème de connexion, le `thread` arrête le programme.

2.3 Boucle de saisie

C'est une boucle infinie, qui a chaque tour demande à l'utilisateur d'entrée un message, de longueur maximum 1000, et ensuite l'envoie au serveur.

2.4 Déconnexion du client

Le seul moyen qu'à l'utilisateur d'arreter le programme est via la commande `CTRL+C`. Dans le cas où le client se déconnecte ou que le serveur rencontre un problème de connexion, il est possible pour le programme appelant, c'est à dire le client ou le serveur, la fonction `receiving` qui utilise l'appel système `recv`, d'identifier si un problème de connexion est survenu. Cette fonction se charge de receptionner un message sur un socket. Dans la cas où un probème de déconnexion se fait resentir, cette fonction indique que le message receptionné est de longueur 0.

Donc lors de la réception, le programme appellant peut dans le cas échéant réaliser une action si un problème de connexion survient `i.e.` la longueur du message est nulle. Dans le cas du client, si le serveur se plante, le programme client s'arrête. Dans le cas du serveur si une connexion se revèle être coupée avec client, le serveur ferme le socket qui le lie au client, et considère que celui-ci n'est plus connecté.

Conclusion

Ce rapport met en avant les parties essentielles du projet. Ainsi les programmes serveur et client ont été expliqués de telle sorte à mettre en avant les principes de bases de la compréhension de l'implantation du chat. Les différents appels sytèmes vus en cours et utilisés dans ce projet ont été simplifiés de telle sorte que le code soit plus lisible, et que l'implémentation de chaque partie de projet, soit des plus simple et intuitive.