Université Libre de Bruxelles

INFO-F201

Systèmes d'exploitation

Projet programmation système : Chatroom

Bas Alessandra
Nieto Navarrete Matias
B-INFO

Décembre 2021

1 Description

Ce projet a été réalisé dans le cadre du cours INFO-F201 (Sytèmes d'exploitation).
Il s'agit d'un salon de discussion auquel plusieurs utilisateurs peuvent se connecter et discuter les uns avec les autres. Les messages envoyés sur le salon de discussion sont globaux, un message envoyé par un utilisateur est visible pour tous les utilisateurs connectés au même moment.

Notre programme est écrit en C et est composé de 5 fichiers :

1. makefile
2. server.c (./server [port])
3. client.c (./client [Username] [ip] [port])
4. client_interface.c (./c_interface [Username] [ip] [port])
5. common.h

Le makefile compile server.c, client.c et client_interface.c sans erreurs avec le flag -Wall.
Une fois qu'un serveur est lancé, les clients peuvent se connecter en utilisant l'ip et le port correspondants. Les clients peuvent ensuite échanger des messages.
Si un client se connecte/déconnecte, un message de connexion/déconnexion est affiché chez tous les clients.
Pour finir, si le serveur est fermé avec CTRL+C ou se ferme pour une autre raison, tous les clients sont automatiquement déconnectés avec un message de fermeture.

2 Implémentation

2.1 Client

- `void setup_socket(int port, const char *ip_address)`

  Prépare le socket pour la connexion en prenant le port et l'adresse ip donnés pas l'utilisateur.

- `void send_msg()`

  Prend le message écrit par l'utilisateur, vérifie si le message contient au moins un caractère et envoie le message au serveur.

- `void recv_msg()`

  Reçoit les messages envoyés par le serveur et vérifie qu'ils ont la bonne taille (en coupant la fin du message si ce n'est pas le cas). Vérifie également si le message reçu indique que le serveur s'est fermé. Si c'est le cas, le client se ferme.

- `int main(int argc, char const *argv[])`

  Envoie le username au server. Ensuite, créé une thread pour l'envoie de messages et un thread pour la réception de messages.

2.2 Interface avec ncurses

- `void interface()`

  Créé une interface simple avec champ d'écriture et les messages qui s'affichent.

2.3 Server

Pour le server, nous avons commencé avec le code donné à l'exercice 1 de la scéance 9 des travaux pratiques. Nous avons ensuite adapté ce code pour les besoins de ce projet.

- `struct User`

  Structure qui créé un tableau de char, qui permet de récupérer le pseudo à la connexion des clients.

- `void setup_socket(int port)`

  Prépare le socket pour la connexion en prenant le port.

- `int main(int argc, char *argv[])`

  Démarre le serveur, gère les différents clients grâce à l'appel système select.
  Le main se charge d'avertir tous les clients si le serveur se ferme, d'afficher les messages chez tous les clients avec les bons username et indique les connexions et déconnexions.

2.4 Common

Le code du fichier common.h provient essentiellement de l'exercice 1 de la scéance 9 des travaux pratiques avec quelques changements.

- `void catch_ctrl_c(int sig)`

  Permet d'arrêter proprement le programme et affiche un message à l'utilisateur après avoir fait un CTRL+C.

- `int _checked(int ret, char* calling_function)`

  Vérifie si les fonctions du socket sont possible sinon ferme le programme avec message d'erreur.

- `int ssend(int sock, char* timestamp, void* data, size_t len)`

  Permet d'écrire dans un socket et renvoi le nombre d'octets envoyés.

- `size_t receive(int sock, char* timestamp, void** dest)`

  Permet de lire dans un socket et renvoi le nombre d'octets lus.

- `char* get_time()`

  Renvoi la date et l'heure.

3 Difficultés

3.1 Problèmes de transmission des messages, caractères en trop.

Le problème était que quand le client recevais un message, celui-ci avait des caractères en trop.
Pour détecter d'où provenait l'erreur, nous avons affiché sur le terminal le nombre d'octet du message grâce aux fonctions ssend, receive, strlen(const char *str). Ce qui nous a permit de constater la différence de taille entre les fonctions receive et strlen().
Pour résoudre le problème, nous avons couper notre chaîne de caractère au bonne endroit grâce a cette ligne de code : " `message[recv] = '\0';`" .

3.2 Récupération du pseudo du client lors de sa connexion au serveur.

Nous avons dû créé une structure qui initialise un tableau de caractères.
Ensuite nous avons crée une variable "`struct User nom_user[100]`", qui est un tableau de tableaux de caractères. Ainsi, à chaque connexion d'un client sur le serveur son pseudo est enregistré dans le tableau.
Ceci permet au serveur d'associer un pseudo à chaque message envoyé.

4 Limitations

1. Plusieurs clients peuvents être connectés simultanément avec le même username sans générer d'erreur car aucune vérification n'est faite à ce niveau.

2. Les messages ne sont pas sauvegardés, si un client quitte le programme et se reconnecte par la suite, les messages précédents auront disparus.

3. Dans l'interface ncurses, lorsque les messages atteignent le bas de la fenêtre, ils sont supprimés pour que l'utilisateur puisse continuer à écrire.

4. Egalement dans l'interface ncurses, si un message s'affiche sur deux lignes, le prochain message sera superposé sur la deuxième ligne du message précédent.

5 Bibliographie et références

1. Livre du cours "Introduction aux systèmes d'exploitation"

2. TP9 : " Programmation réseau multi-client et programmation multi-threads"

3. `https://github.com/nikhilroxtomar/Chatroom-in-C`

5.1 Exemple d'exécution

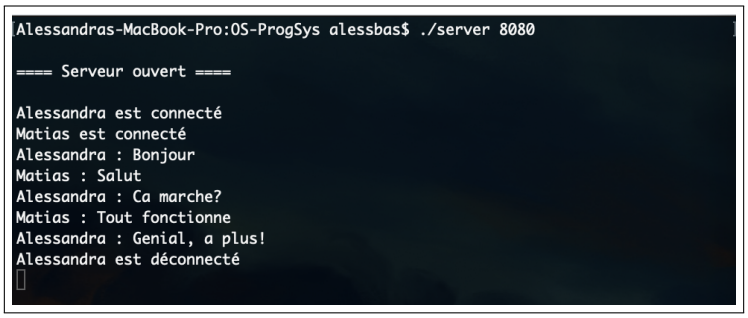

Figure 1: Serveur avec deux clients connectés

Figure 2: Interface ncurses avec deux clients connectés

Client Alessandra quitte le programme avec CTRL+C.

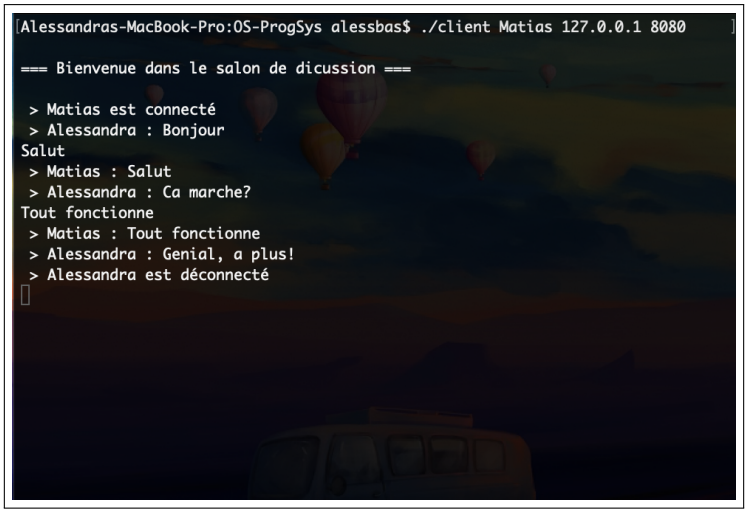

Figure 3: Interface terminal après déconnexion d'un client

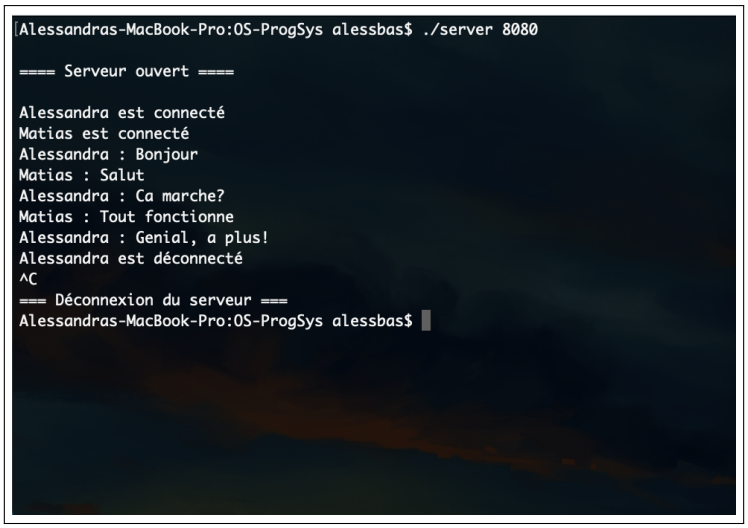

Figure 4: Serveur fermé avec CTRL+C

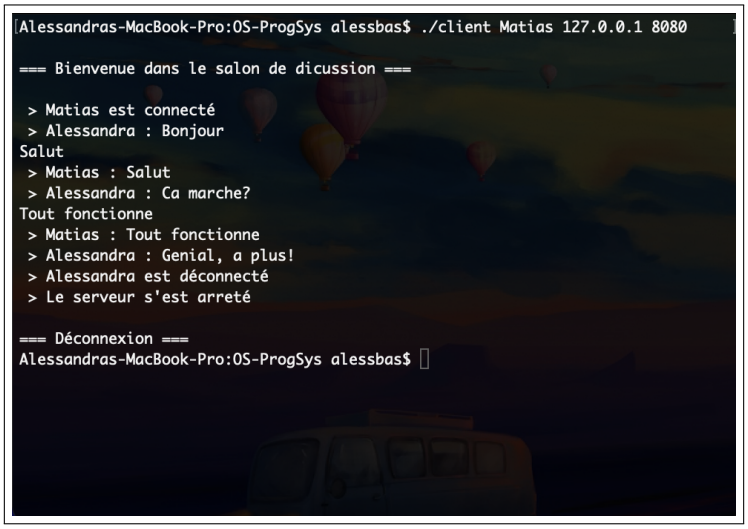

Figure 5: Client quitte automatiquement quand le serveur est fermé