Rapport du projet FREESCORD

Réalisé par : Khireddine BENMEZIANE
Groupe : DL2
N°étudiant : 12308874

Encadré par : Pierre ROUSSELIN

Université Sorbonne Paris Nord
99 Av. Jean Baptiste Clément, 93430 Villetaneuse
Année universitaire : 2024 - 2025

Table des matières

Lien vers le dépôt du projet : https://github.com/khirobenn/Freescord

1. Les ajouts effectués au structures fournies

- **La structure `user` :**

```
#define NAME_MAX 16

struct user {
   struct sockaddr *address;
   socklen_t addr_len;
   int sock;
   char nickname[NAME_MAX];
   char pseudonyme[NAME_MAX];
};
```

- address : On l'utilisera lors de l'appel `accept`.
- addr_len : Pour stocker la longueur de la variable `address`.
- sock : Pour sauvegarder la socket retourné par le client.
- nickname : Pour sauvegarder le nickname du client.
- pseudonyme : Pour sauvegarder le pseudonyme du client.

- **La structure `buffer` :**

```
typedef struct buffer {
   int fd;
   int size_read; // Taille lu par read
   int index; // index courant, si c'est égal à size_read on
devrait appeler read sur fd
   int size; // taille de mon buffer
   char *buff; // mon buffer
}
buffer;
```

- fd : C'est le fichier descripteur dont on lis par `read`.
- size_read : La taille lus par `read`.
- index : L'index courant du buffer, c'est-à-dire le caractère à lire prochainement.
- size : La taille maximale de mon buffer.
- buff : Le tableau de caractères dont on stocke tous les caractères lus.

2. Les fonctionnalités faites

- J'ai pu faire jusqu'à l'exercice 7 avec quelques options de l'exercice 8. Le reste je n'ai pas pu le faire par manque de temps.
- Pour le protocole, je n'ai pas changé celui fourni dans le sujet.

3. Fonctionnement

- **Coté serveur :**
  J'ai 4 variables globales : un mutex, un tube, liste des utilisateurs et la socket du serveur.

```
pthread_mutex_t mutex;
int tube[2];
struct list * users;
int serveur_socket_global; // On l'utilisera pour fermer la
socket du serveur lors de CTRL + C
```

- mutex : On utilisera le mutex à chaque fois qu'on veut effectuer quelque chose sur la liste des utilisateurs, comme ça y'aura pas d'erreur.
- tube : Le tube on l'utilisera de façon que chaque utilisateur peut écrire dessus (chaque utilisateur sera traité par un thread), et y'aura un thread qui pourra lire ce qui écrit dessus, comme ça il pourra transmettre le message aux autres utilisateurs.
- users : On ajoutera nos clients içi à chaque fois qu'il réussit à se connecter au serveur, c'est-à-dire ils ont donné un nickname et pseudonyme valides.
- serveur_socket_global : J'ai décidé de mettre la socket du serveur en global pour que je puisse l'utiliser dans une fonction appelée `` `ctrl_c_handler` `` qui est passée comme argument pour la fonction `` `signal` ``. Cette fonction permettra de fermer la socket d'écoute et libérer la mémoire allouée pour la liste des utilisateurs puis de mettre fin au serveur (source : Write a C program that does not terminate when Ctrl+C is pressed | GeeksforGeeks).

Donc, pour faire simple, le serveur créera une socket d'écoute et un thread qui sera utilisé pour la transmission du message reçu à tous les utilisateurs puis dans la fonction main, on va rentrer dans une boucle infinie qui attendra la connexion d'un client.

À chaque fois qu'un client se connecte, le main du serveur créera un thread et lui donne la structure du client comme argument, comme ça le thread va s'occuper de l'envoi du ascii art, de vérifier le nickname et le pseudonyme du client reçu et aussi de recevoir les messages de ce client. Pendant ce temps, le main pourra attendre un autre client.

- **Coté client :**
  Pour le client, y'a pas grand chose à dire, on crée une socket, on se connecte au serveur. Après cela, on utilisera poll pour lire soit dans la socket créé (c'est-à-dire un message du serveur) ou dans le terminal. Si on a lu dans le terminal alors il faut envoyer les octets lus au serveur. Si on a lu dans le serveur alors il faut afficher les octets lus sur le terminal.
- CTRL+D fermera le client.

4. Compilation et exécution

- **Compilation :** On exécute juste la commande `make`.
- **Exécution :** On ouvre deux terminal et on commence par le serveur, on exécute dans le premier terminal le fichier `srv` avec la commande `./srv`. Maintenant que le serveur est lancé, on peut exécuter le fichier exécutable du client dans le deuxième terminal comme ceci : `./clt ip_address`. Et voilà le client est connecté. Désormais, vous pouvez connecter plusieurs clients ensemble et chattez ;)

5. Sources utilisées

- https://www.geeksforgeeks.org/write-a-c-program-that-doesnt-terminate-when-ctrlc-is-pressed/
- Les cours de Monsieur ROUSSELIN.