École d'ingénieurs
Télécom Physique Strasbourg

TP RIO

Auteurs : Thomas DUMOND & Thomas BOUYER

Sommaire

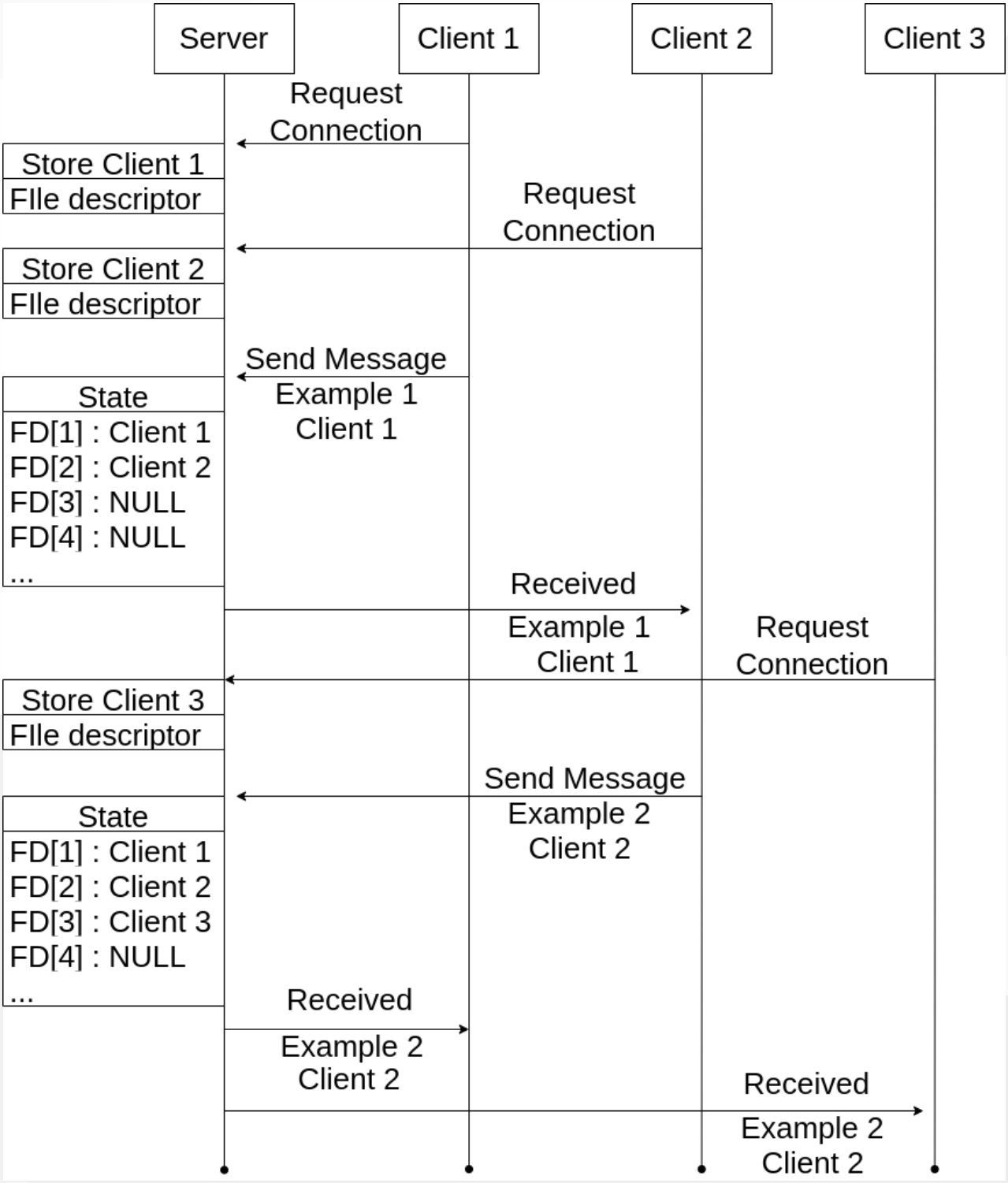
Introduction

Le sujet choisi est :

- Client - Serveur centralisé multi-utilisateurs avec communication par identifiant

Nous avons implémenté à l'aide de threads ,de sémaphores et de gestion d'entrée sortie bloquantes sans utilisation de `select`.

Principe de fonctionnement



Utilisation

Plusieurs choix sont possibles pour tester la bonne execution de l'application :

- Utilisation en local (on lance le serveur et les clients en local pour nos tests).
- Utilisation en réseau (le serveur est déjà lancé sur un Raspberry et les clients sont connectés au Raspberry distant).

Utilisation en local

- On compile et on lance le serveur avec la commande : `make run-server`. Ainsi, le programme serveur sera compilé et lancé en local avec comme port le port **36000**.

> On peut aussi lancer le server après compilation avec la commande : `./bin/server -p PORT` avec PORT le numéro de port d'écoute du serveur.

- On compile et on lance le client avec la commande : `make run-client`. Ainsi, le programme client sera compilé et lancé en local avec comme port le port **36000**.

> On peut aussi lancer le client après compilation avec la commande : `./bin/client -p PORT -t TARGET` avec PORT le numéro de port d'écoute du serveur et TARGET l'adresse IP du serveur distant.

Utilisation en réseau

- Le serveur est déjà lancé sur un Raspberry et les clients seront connectés au Raspberry distant. Il suffit donc juste de lancer et de compiler les clients avec la commande : `make run-client-rpi`.

Le Raspberry distant est actuellement en train d'écouter sur le port **32000** à l'adresse 88.170.206.241. Il est donc utilisable pour tester l'application.

Explications du code

Serveur

Premièrement, on initialise le serveur sur le port d'écoute avec la fonction :

```
srv = init_server(targ->port);
```

Ainsi sera initialisé le descripteur de fichier du serveur et renvoyé dans la structure `srv` qui contient les informations sur le serveur.

Ensuite, on crée tous les threads qui seront utilisés pour gérer les connexions entrantes :

```
// launch thread to handle client requests
pthread_t tid[NB_CLIENTS];

// Init file descriptors
for (int i = 0; i < NB_CLIENTS; i++) {
    srv->fds[i] = NULL;
}

// Init sem to get id for thread
T_CHK(sem_init(&srv->sem, 0, 1));

for (int k = 0; k < NB_CLIENTS; k++) {
    T_CHK(sem_wait(&srv->sem));
    srv->id = k;
    T_CHK(
        pthread_create(&tid[k], NULL, (void *(*)(void *))run_server, srv));
}
```

Ainsi, on crée un tableau de threads qui seront utilisés pour gérer les connexions entrantes. On crée également un tableau de descripteurs de socket qui seront utilisés pour stocker les descripteurs de socket des clients. Le sémaphore `srv->sem` est utilisé pour gérer l'accès concurrent à la structure `srv`.

Client

Pour le client, on commence par créer notre structure `frame_t` qui contiendra la trame envoyé par le client (elle contient le message ainsi que le nom d'utilisateur du client qui l'a envoyé).

On demande donc a l'utilisateur de saisir son nom d'utilisateur puis on initialise le socket avec le port et l'adresse IP du serveur distant.

```
int sockfd;
frame_t frame;

// Ask user fot username
char *username = get_line("Username: ");
trim(username);
strlcpy(frame.name_id, username, STR_LEN_MAX);

sockfd = init_client(target, port);
run_client(sockfd, &frame);
done_client(sockfd);

exit(EXIT_SUCCESS);
```

Le client se connecte au serveur distant avec la fonction `run_client` qui prend en paramètre le socket et la structure `frame_t` contenant le message a envoyer. Cette fonction utilise des threads le premier qui se charge de lire les messages du serveur et les afficher à l'écran. Le second qui se charge de lire les messages de l'utilisateur et les envoyer au serveur.
La fonction `done_client` est appelée à la fin du programme pour fermer le socket et libérer les ressources.
La fonction `trim` permet de supprimer les caractères d'espacement en début et fin de chaîne.

Futures améliorations

Le projet est actuellement fonctionnel mais est limité dans ces fonctions. Les futures améliorations sont :

- Choisir à qui envoyer le message car actuellement un message est envoyé à tous les clients
- Authentification pour les clients
- Message qui ne transitent pas par le serveur
- Cryptage des messages
- Allocation dynamique de la table des descripteurs de socket

Conclusion

Nous avons donc implémenté un server de chat multi-utilisateurs avec communication par identifiant. Ce projet est encore en cours de développement.