Rapport INFO-F201 Projet 2: Salon de discussion en C

Abraham Théo, Merian Emile

17 December 2021

Contents

1 Introduction

Dans le cadre du projet 2 d'Info-f-201, nous avons implémentés une chatroom, un salon de discussion via serveur. Chaque personne en connaissance de l'IP du serveur et du port peux se connecter en fournissant un pseudo, et échanger en temps réel avec les autres utilisateurs connectés.

Ce rapport présente les implémentations, limites et difficultés rencontrées lors de ce projet.

2 Implémentations

2.1 Fichier de base

Afin de commencer notre projet 2 sur de bonnes bases, nous avons travaillés sur les fichiers de l'exercice 1 du TP 9 portant sur l'utilisation de select().

Cette base couvre une partie non-négligeable des implémentations nécessaires:

1. L'implémentation d'un fichier C client.c, d'un fichier C serveur.C et d'un fichier commun aux deux common.h. Le fichier common.h possède une fonction ssend(socket, buffer, size) servant à envoyer des messages dans un pipe et une fonction recv(socket, buffer, size) servant à lire les messages envoyés dans le pipe. Le serveur initialise le pipe a son lancement et le client s'y connecte a travers le port du serveur. Le client peut écrire des messages et le serveur les lit puis les renvoies au client-autheur.

2. L'implémentation du système select dans le serveur afin de gérer la connexion et la reception de messages de plusieurs clients simultanés.

2.2 Pseudos et parsing d'arguments

Le parsing d'arguments s'implémente comme suit:

Dans le serveur, argv[1] est stocké dans un objet int port et est utilisé dans htons(port)

Dans le client, argv[1] est stocké dans un const char* pseudo, argv[2] est stocké dans un const char* ip_adress et argv[3] est stocké dans un int port et utilisé comme décrit précédement.

Le pseudo est envoyé au serveur dès la connexion du client et est stocké dans une liste de char* pseudo[1024] au même index que le client dans la liste client[1024]. Ainsi le serveur récupère le pseudo correspondant au clients[nclients] ayant envoyé un message pour formatter le message de retour aux clients. Le format du message se présente ainsi: "pseudo: message". Il est suivi de la taille du message (hors timestamp et pseudo), du timestamp de l'envoi, puis du message en question comme décrit dans l'énoncé.

2.3 Client: pthreads

La librairie pthread permet de lancer une fonction particulière d'un processus en parallèle, c'est a dire sans bloquer l'execution du processus "original". C'est de cette librairie dont nous nous sommes servis pour gérer l'envoi et la reception simultanée de messages par le client. La reception est en thread, tandis que l'envoi est géré par le thread original.

2.4 format du message

trois objets interviennent dans un message reçu puis renvoyé par le serveur:

1. la taille du message size_t nbytes,
2. le temps time_t du message obtenu par la librairie time.h,
3. le message char* recvbuffer.

2.5 Mutex

L'implémentation d'un mutex a été jugé inutile, puisqu'il n'y a pas de partage de ressources. Une discussion avec d'autres groupes nous a appris qu'un Mutex n'était nécessaire que dans le cas de l'utilisation de la librairie ncurses.

3 limites

Ce code connait malheureusement quelque limites que nous n'avons pas réussis a résoudre dans les temps :

1. Le pseudo ne peux pas depasser la taille de 7 caractères,
2. Des caractères spéciaux peuvent apparaitres aux cotés du message a de très rares occasions,
3. La gestion du signal CTRL+C peut peut être réagir aux autres signaux.

Cependant, -Wall ne soulève aucun flag, et le code se compile sans soucis.

4 Difficultés

4.1 Manipulation de char*

Ce projet nous a montré à nouveau que le C est un langage difficile. Une des principales difficultés que nous avons rencontré est la manipulation de pointeurs de char, savoir quand il faut utiliser un pointeur, une référence ou simplement un objet n'est pas tâche aisé, surtout dans le formattage des messages et dans la réception des messages chez le client.

4.2 Allocation de la mémoire

L'allocation mémoire et sa gestion sont des sujets qui nous étaient flous pour ne pas dire inconnus, et l'apprentissage fut rude, mais ce projet met bien en contexte l'utilité d'apprendre ce genre de choses.

5 solutions originales

5.1 gestion de signal()

Pour permettre au serveur de se terminer, select() a un timeout de 1s, et signal en cas de CTRL+C renvoi vers une fonction qui ferme tous les canaux de communications et éteint tous les clients.