

Une application de messagerie instantanée



Un projet FAR de programmation en C en cycle itératif par

CHEKOU Nesrine, GUYON Valentin & VEBER Mathieu

ANNEE 2018-2019

Responsable : M. TIBERMACINE

Sommaire

[Itération 1 2](#_Toc6167251)

[1. Les difficultés rencontrées 2](#_Toc6167252)

[2. La répartition du travail 3](#_Toc6167253)

[3. Compilation et exécution 4](#_Toc6167254)

[4. Le schéma du protocole de communication 5](#_Toc6167255)

# Itération 1

## Les difficultés rencontrées

Nous avons fait face à un certain problème avant même de commencer le projet, lors de la création des groupes il nous était demandé de constituer des binômes ou trinômes hétérogènes, c’est-à-dire d’origines différentes. Or dans le groupe 2, très peu nombreux sont les élèves issus d’IUT ou de BTS informatique. Notre groupe est donc composé de 3 PeiP et aucun d’entre nous ne possède une quelconque expérience en C ou en réseaux. C’est donc sans aucun référent technique que nous nous lançons corps et âmes dans le projet FAR. Concernant ce problème nous n’avons pas trouvé de solution.

De plus le langage C en lui-même est difficile d’accès, malgré les très bons cours d’introduction de M. Fioro et M. Mirka, maitriser un tel langage en si peu de temps n’est pas chose aisée. Il nous a fallu un peu plus de temps pour nous accoutumer au C, inévitablement ce temps perdu a entrainé un retard sur notre projet d’application de messagerie instantanée.

La première vraie difficulté rencontrée concerne la distinction entre les clients 1 et 2, car pour la première itération il nous est demandé de coder un serveur relayant des messages textuels entre deux clients de manière alternée, c’est-à-dire lorsque le client 1 a envoyé un message il doit attendre une réponse de client 2 avant de pouvoir en renvoyer un. Il fallait donc dissocier client 1 de client 2, pour ce faire nous avons déclaré deux variables globales dans le serveur. C’est d’ailleurs ce dernier qui dissociera les 2 clients. Les variables globales id1 et id2 définies sur les valeurs 0 et 1 symbolisent respectivement les clients 1 et 2. Grâce à la variable idclient qui prend la valeur d’id1 ou d’id2, le serveur arrive à déterminer de quel client est le message qu’il reçoit de la manière suivante :

* Le premier message est toujours du client 1, c’est-à-dire le premier client exécuté, idclient est donc initialisé à id1 donc 0.
* Une fois qu’il a recu le message du client 1 idclient vaut id2, ainsi il sait que le prochain message qu’il recevra sera du client 2.
* Une fois qu’il a recu le message du client 2 idclient vaut id1, ainsi il sait que le prochain message qu’il recevra sera du client 1.
* Et cela continue tant que le serveur ne reçoit pas de message « fin ».

Avec cette méthode on identifie qui envoie les messages au serveur il ne reste plus qu’à « redistribuer » le message reçu à l’autre client.

Toutefois renvoyer le message du client 1 au client 2 n’était pas si simple. On ne peut pas simplement recréer une variable pour stocker chaque message car au bout d’un moment la mémoire sature. Nous avons donc créé une unique variable avec une taille fixée à 120 (la taille est une variable globale, on peut donc à tout moment changer la taille maximale des messages que peuvent s’échanger le client 1 et 2. C’est donc sur cette variable nommée message que l’on écrira les messages à renvoyer, c’est le fonctionnement d’un buffer. Mais quand on réécrit une chaine de caractère par-dessus une autre dans le buffer grâce à la fonction sprintf(), si la nouvelle chaine est plus courte que l’ancienne les derniers caractères du buffer seront ceux de l’ancien message. Pour éviter ce problème d’écriture il faut reset le buffer à l’aide de la fonction bzero() qui remplace tout les caractère du buffer par des ‘\0’, une fois cette manipulation effectuée on peut réécrire sur le buffer sans risque avec sprintf().

En ce qui concerne cette itération il ne reste pas de bug majeur ou des difficultés à surmonter. Cependant il reste des détails à améliorer par exemple si un client quitte le chat sans dire fin, il faut redémarrer le serveur et l’autre client reste coincé dans l’attente d’une réponse.

## La répartition du travail

Pour cette première itération, nous avons majoritairement travaillé ensemble pour compenser nos lacunes en C. Toutefois Nesrine et Mathieu ont passé plus de temps sur le code alors que pour ma part j’ai rédigé ce rapport.

## Compilation et exécution

Pour compiler les programmes, place vous à l’aide du terminal dans le répertoire où se trouve les fichiers server.c et client.c, puis taper dans le terminal les 3 commandes suivantes :

* gcc -o server server.c
* gcc -o client1 client.c
* gcc -o client2 client.c

Pour exécuter les programmes taper les commandes suivantes dans l’ordre, pour obtenir l’<ip> faites un ifconfig et récupérer votre adresse ipv4 car c’est cette dernière qui correspond à l’<ip> :

* ./server
* ./client1 <ip>
* ./client2 <ip>

Voilà, vous pouvez maintenant chatter avec vous-même !

## Le schéma du protocole de communication