Rapport du Projet MiniChat : Rémi BOSSUET

I. Version connectée via verseur

Cette version du projet via un serveur est basée sur la manipulation de tubes nommés entre différents processus et sur l'utilisation de la fonction *select* qui permet d'écouter et de lire un ensemble de descripteur de fichier.

Le fichier serveur.c gère la partie serveur :

- Initialisation (création et ouverture) du tube écoute qui permet aux clients de se connecter au serveur. Le serveur attend donc qu'un client écrive son pseudo dedans pour rejoindre le salon.
- Une fois un client connecté, grâce à une boucle while, il regarde avec *select* si des messages sont arrivés :
 - S'il y a des données dans les tubes nommés, il s'agit un message "texte" d'un client. Le serveur transmet ces données en écrivant ces dernières dans les tubes de tous les clients actifs
 - Si des données arrivent du tube écoute, il s'agit alors d'une demande de connexion. Il ajoute alors ce client dans la liste des clients/participant actifs
- Le serveur sort de la boucle *while* et s'arrête quand un client nommé "fin" tente de se connecter. Ce provoque la fermeture du serveur et la déconnexion des autres clients (i.e. suppression des tubes et des descripteurs utilisés)

<u>Notes</u>: Au départ, lorsque j'affichais sur la console les messages de chaque utilisateur, j'avais 1 ligne et demie d'espaces blancs après chaque message: cela était dû au fait que je ne tronquais pas les messages à leur longueur réelle, ce que j'ai fait par la suite en ajoutant le caractère de fin de chaine '\0' à la fin de chaque chaine de caractère.

J'ai décidé de factoriser le code de la fonction *main* en créant des fonctions qui gèrent la création des messages pour regrouper toutes les fonctions de manipulation de chaines de caractères dans une même fonction.

Le fichier console.c gère, lui, la partie client :

- Il commence par ouvrir le tube d'écoute et écrit son nom dedans pour se connecter au client. Il créé ensuite les 2 tubes de services
- Grâce à une boucle while, le client passe son temps à sonder ces 2 tubes de service :
 - Si un autre client a posté un message, il reçoit ce message du serveur via tubeS2C. Il affiche alors ce message sur son terminal avec la fonction afficher
 - S'il y a de la donnée à lire sur le clavier, alors le client l'affiche sur son terminal et l'écrit dans le tubeC2S pour l'envoyer au serveur

- Si l'utilisateur du client écrit "au revoir", le client sort de cette boucle while et arrête d'écouter les tubes. Ce processus client meure donc en supprimant les descripteurs et les tubes utilisés grâce à la procédure *clean*.
- Enfin, si le client s'appelle "fin", on ne rentre pas dans cette boucle while. On écrit "fin" dans le tube écoute et on laisse le serveur gérer sa fermeture dans serveur.c

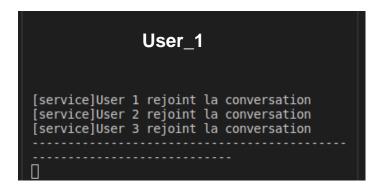
<u>Notes</u>: J'ai factorisé le « nettoyage » (i.e. la suppression des descripteurs de fichiers et des tubes nommés) avec une fonction *clean*. J'aurais peut-être pu encore plus factoriser le code de la fonction *main* en créant et ouvrant les tubes nommés dans une fonction auxiliaire, mais cela m'a paru secondaire.

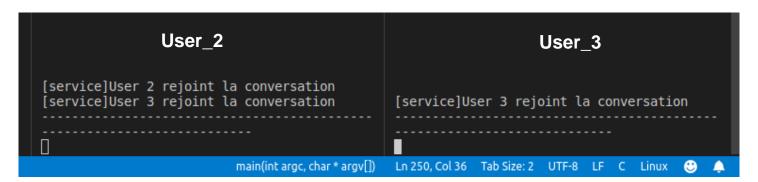
Méthodologie des tests effectués - Version Serveur

Afin de tester mon programme, j'ai commencé à me baser sur les affichages donnés dans le sujet. Le programme a été testé séparément avec 1, 2, 3, 4, 5 et 6 utilisateurs. Les résultats pour 1 à 5 utilisateurs sont identiques. Par soucis de lisibilité, les exemples ci-dessous ont été réalisés avec 3 utilisateurs

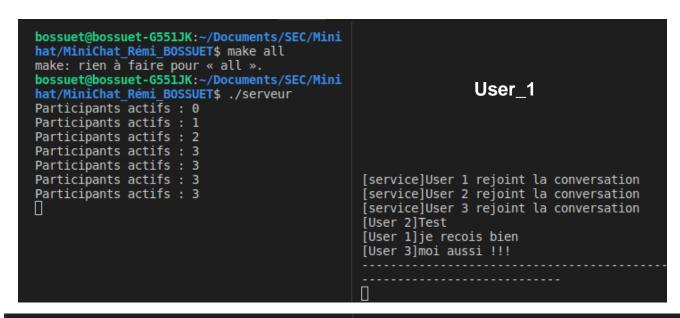
Connexion de trois utilisateurs dans l'ordre : User_1 -> User_2 -> User_3

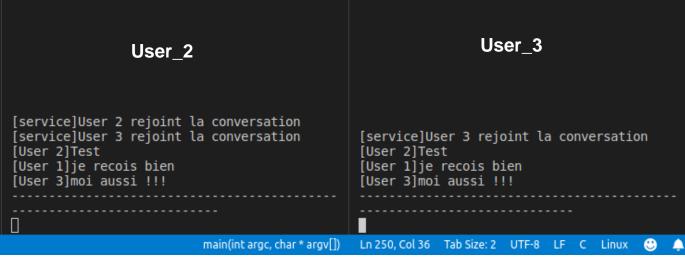
```
bossuet@bossuet-G551JK:~/Documents/SEC/Minihat/MiniChat_Rémi_BOSSUET$ make all make: rien à faire pour « all ».
bossuet@bossuet-G551JK:~/Documents/SEC/Minihat/MiniChat_Rémi_BOSSUET$ ./serveur
Participants actifs : 0
Participants actifs : 1
Participants actifs : 2
Participants actifs : 3
```





Envoie de messages





Déconnexion du client 2 : en envoyant le message « Au Revoir »

```
bossuet@bossuet-G551JK:~/Documents/SEC/Mini
hat/MiniChat Rémi BOSSUET$ make all
make: rien à faire pour « all ».
                                                                             User_1
bossuet@bossuet-G551JK:~/Documents/SEC/Mini
hat/MiniChat Rémi BOSSUET$ ./serveur
Participants actifs : 0
Participants actifs : 1
Participants actifs : 2
                                                        [service]User 1 rejoint la conversation
[service]User 2 rejoint la conversation
[service]User 3 rejoint la conversation
Participants actifs : 3
Participants actifs : 3
Participants actifs : 3
                                                        [User 2]Test
[User 1]je recois bien
[User 3]moi aussi !!!
Participants actifs : 3
Participants actifs : 3
Participants actifs : 2
                                                        [User 2]J'y vais a+ !
                                                        [User 2]au revoir
```

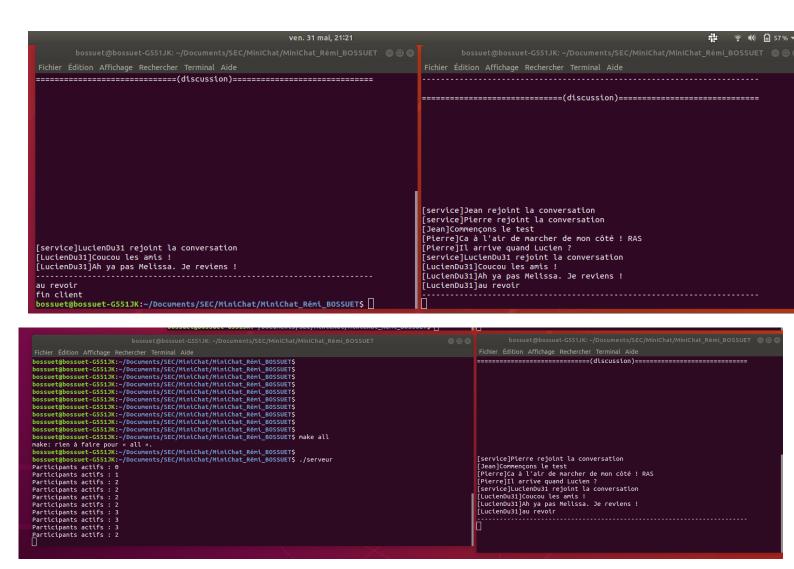
L'utilisateur 2 tente de se connecter avec le pseudo « fin »

```
bossuet@bossuet-G551JK:~/Documents/SEC/Mini
hat/MiniChat Rémi BOSSUET$ make all
make: rien à faire pour « all ».
                                                                     User 1
bossuet@bossuet-G551JK:~/Documents/SEC/Mini
hat/MiniChat Rémi BOSSUET$ ./serveur
Participants actifs : 0
Participants actifs : 1
                                                  [service]User 1 rejoint la conversation
                                                   [service]User 2 rejoint la conversation
Participants actifs : 2
                                                   [service]User 3 rejoint la conversation
Participants actifs : 3
                                                  [User 2]Test
[User 1]je recois bien
[User 3]moi aussi !!!
[User 2]J'y vais a+!
[User 2]au revoir
Participants actifs : 3
                                 Serveur
Participants actifs : 3
Participants actifs : 3
Participants actifs : 3
Participants actifs : 2
bossuet@bossuet-G551JK:~/Documents/SEC/Mini
                                                  Fermeture du serveur.
hat/MiniChat_Rémi_BOSSUET$
                                                  bossuet@bossuet-G551JK:~/Documents/SEC/MiniC
                                                  at/MiniChat Rémi BOSSUET$
```

```
[service]User 2 rejoint la conversation
                                                                     User_3
service]User 3 rejoint la conversation
[User 2]Test
[User 1]je recois bien
[User 3]moi aussi !!!
                                                  [service]User 3 rejoint la conversation
                                User_2
[User 2]J'y vais a+!
                                                  [User 2]Test
                                                  [User 1]je recois bien
                                                  [User 3]moi aussi !!!
                                Alia « fin »
                                                  [User 2]J'y vais a+ !
au revoir
                                                  [User 2]au revoir
Fin client
bossuet@bossuet-G551JK:~/Documents/SEC/MiniC
                                                 Fermeture du serveur.
at/MiniChat Rémi BOSSUET$ ./console fin
Fin client
bossuet@bossuet-G551JK:~/Documents/SEC/MiniC
                                                 bossuet@bossuet-G551JK:~/Documents/SEC/Mini
at/MiniChat Rémi BOSSUET$
                                                 hat/MiniChat Rémi BOSSUET$
                                                 Ln 250, Col 36 Tab Size: 2 UTF-8 LF C Linux
                           main(int argc, char * argv[])
```

BOSSUET Rémi Rapport Minichat SEC

Test affichage « normal » dans un vrai terminal



Note : Je n'ai pas réussi à débugger l'affichage du message de déconnexion quand une personne part en tapant « Au revoir ». J'ai donc mis en commentaire les parties correspondantes

II. Version Tableau Blanc

Cette version du projet consiste à coupler une zone de la mémoire avec un fichier, et chaque utilisateur vient écrire de façon autonome son message dans le fichier.

Le fichier chatmmap.c gère l'ensemble de cette version. Dans l'ordre :

- Il ouvre un fichier de discussion (ou le créer s'il n'existe pas)
- Il couple ce fichier avec son processus en utilisation la fonction suivant : discussion = mmap(NULL, taille, PROT_READ | PROT_WRITE, MAP_SHARED, fdisc, 0). Cela donne un accès partagé en lecture/écriture au ficher
- Enfin, une fois l'initialisation effectuée, grâce à la boucle while, le programme lit (de façon non bloquante) les données en sortie du clavier et vient les écrire dans le fichier à chaque retour chariot.
- De plus, chaque seconde, le processus vérifie la présence de nouveaux messages dans le fichier avec alarm(1). Ce signal SIGALRM est géré par un handler (nommé traintant) et affiche les nouveaux messages. Après chaque message posté par l'utilisateur, une autre alarme est lancée automatiquement après pour accélérer le rafraîchissement du salon de discussion

Note : J'ai écrit une fonction auxiliaire *ecrire_saisie* qui permet e factoriser les opérations sur les chaines de caractères

Méthodologie des tests - Version Tableau blanc

La méthodologie de tests suivie dans cette partie est la même que précédemment :

Ajout d'un message par les deux utilisateurs :



Remplacement et effacement du tableau blanc :

```
======(discussion)=
[Celine] : Le remplacement se passe bien
[Celine] : Le replacement se passe toujours bien
[Remi] : Le premier message
[Celine] : Et le deuxième
[Remi] : test
[Remi]
[Remi]
[Remi]
       : d
[Remi]
[Remi]
[Remi]
        : d
[Remi]
[Remi]
[Remi]
[Remi]
       : d
[Remi]
[Remi]
       : dd
[Remi]
[Remi] : d
```

```
======(discussion)=====
[Celine] : Le remplacement se passe bien
[Celine] : Le replacement se passe toujours bien
[Remi] : Le premier message
[Celine] : Et le deuxième
[Remi] : test
[Remi] : d
[Remi]
[Remi]
        : d
[Remi]
        : d
[Remi]
[Remi]
        : d
[Remi]
        : d
[Remi]
[Remi]
        : d
[Remi]
        : d
[Remi]
       : d
[Remi] : dd
[Remi] : d
[Remi] : d
```

Les deux premiers messages sont bien remplacés par les nouveaux messages.

J'ai aussi effectué un test complémentaire : j'ai vérifié qu'on pouvait bien quitter et revenir sur le même salon de discussion et avoir le même contenu. Pour aller plus loin, j'ai ouvert le fichier binaire de la discussion et j'ai retrouvé les différents messages du tableau

III. Comparaison des deux versions

La version serveur / client est plus proche de celle utilisée réellement. En effet, les utilisateurs ont généralement envie de dialoguer avec des personnes partout dans le monde, et pas seulement avec des utilisateurs à côté : en réalité, il n'y a pas de zone de mémoire partagée entre tous les utilisateurs. Mais elle présente quelques inconvénients : ce patron de conception oblige la redirection constante des messages jusqu'au serveur. De plus, l'état et la capacité des serveur influe grandement sur la discussion : ces éléments limitent le nombre d'utilisateurs et rendent possible tout échange si les serveurs sont *down*.

Ceci dit, la version tableau blanc présente elle aussi des avantages. Etant basée sur le couplage mémoire, elle permet de ne plus avoir à ouvrir / lire / écrire dans des fichiers puisqu'il suffit d'accéder à une adresse de la mémoire pour commencer à utiliser la messagerie. Les inconvénients de cette version seraient qu'il est nécessaire pour le client de vérifier « manuellement » s'il doit mettre à jour la discussion, et de plus, l'utilisation d'une telle version du chat en réseau me semble compliquée, car il faudrait gérer l'utilisation de ressources sur différentes machines.