Compte rendu projet FAR

Ayoub MOUJANE - Pierre PERRIN - Julien WIEGANDT

SPRINT 4 :

VERSION 1 : Les clients peuvent rejoindre des salons à deux places

Protocole de communication :

Comme pour le Sprint 1 et 2 nous avons décidé de mettre en place une communication en mode connecté (TCP) pour assurer que les données échangées ne se perdent pas sur le réseau. Le fonctionnement des messages n'a pas changé par rapport à la v3, la nouveauté dans cette version est qu'un client doit désormais choisir un salon à rejoindre avant de pouvoir entrer un pseudo et de communiquer avec les clients dans le salon.

Chaque salon possède deux places, au moment du choix de salon un client peut entrer « /list-channels » pour obtenir les informations des salons. Il peut ainsi voir le numéro du salon, le titre du salon, la description du salon ainsi que le nombre de place libres.

Pour rejoindre un salon, un client entre le numéro du salon qu'il souhaite rejoindre. S'il n'y a plus de place disponible le serveur l'informe que le salon est plein et il est déconnecté. S'il entre un numéro invalide ou que le numéro ne correspond à aucun salon, il est invité à réitérer son entrée.

Une fois entré dans un salon un thread dédié au client est initié. Il sert à recevoir les messages et les transmettre aux membres du salon dans lequel le client se trouve. Lors de la création du thread on fait passer les informations nécessaires pour que le thread connaisse le salon dans lequel est entré le client. Nous avons choisi cette manière car c'est la plus simple et rapide à mettre en place, il n'a fallu que reprendre le thread réalisé dans les versions précédentes et l'adapter un peu. Ainsi les messages ne sont transmis qu'aux clients de son salon, ce qui implique qu'un client ne reçoit que les messages des clients de son salon.

Afin de rendre le code plus propre et plus compréhensible séparer, nous avons pris l'initiative de créer une librairie « salon.h » comme nous avons appris en début de semestre qui est

ensuite importée par le serveur. Cette librairie contient la structure de donnée utilisée pour représenter les salons ainsi que les méthodes qui nous permettent d'utiliser aisément les salons.
(voir diagramme de séquence en annexes page 6).

Difficultés rencontrées :

Durant ce sprint nous avons rencontré quelques difficultés mais beaucoup moins que lors du sprint précédent.

Le client veut envoyer un fichier avant d'avoir rejoint un salon :

Un client n'est pas censé pouvoir envoyer un fichier tant qu'il n'a pas rejoint de salon car à qui serait-il destiné ? Le problème était que à partir du moment où le client envoyait « file » alors il se mettait en mode envoi. Pour régler ce problème sans avoir à changer notre protocole de communication nous avons décidé de déconnecter directement le client s'il envoi « file » à ce moment-là (idem s'il envoi « fin »).

Le parcours d'un tableau dont la longueur varie :

Pour chercher les places disponibles dans le salon il fallait parcourir le tableau contenant les sockets des clients présents. Pour cette version ce n'était pas nécessaire puisque le nombre de place par salon est le même mais nous avons décidé de nous préparer au fait que le nombre de places pouvait changer en vu de la V2, il fallait donc récupérer la longueur du tableau. Au début nous utilisions sizeof(*le tableau*) mais cela nous donnait la quantité mémoire allouée pour ce tableau qui n'était pas sa longueur. Nous avons donc compris qu'il fallait juste diviser par la taille des données stockées dans le tableau afin d'obtenir la vraie longueur. Donc la longueur de notre tableau est calculée de la manière suivante :

```
Int SizeArray = sizeof(clients)/sizeof(clients[0])
```

Si un client ne choisit pas de salon il bloque les autres clients :

Cela venait du fait que le serveur traitait la connexion aux salons dans un seul et même processus. La résolution de ce problème fut rapide puisque nous étions de plus en plus familiers avec la technique du multi-threading, la solution nous est donc venue rapidement. Le fonctionnement actuel est le suivant : Lorsqu'un client se connecte au serveur, un thread de connexion aux salons est créé pour ce client.

Répartition du travail :

	Création de la structure de donnée salon	Création du thread de connexion à un salon	Gestion des input lors de la connexion au salon	Création des salons disponibles	Diagramme séquence
Ayoub Moujane					
Pierre Perrin					
Julien Wiegandt					

VERSION 2 : Les clients peuvent rejoindre des salons à m places

Grâce au fait que nous avons préparé notre code à s'adapter à la variation de taille des tableaux, il n'y a rien eu à changer à part la longueur des tableaux. Ainsi une constante TAILLE_SALON est définie dans le programme serveur et permet de changer la taille des salons.
(Voir diagramme de séquence en annexes page 6).

VERSION 3 : Les clients peuvent créer, modifier et supprimer des salons

Protocole de communication :

Dans cette version nous sommes restés en TCP et l'envoi des messages et fichiers étaient gérés exactement de la même manière que les versions précédentes. L'option que l'on a rajoutée est de donner la possibilité à un client de pouvoir créer, supprimer ou bien modifier un salon. Pour cela nous avons décidé d'implémenter une liste doublement chainée pour pouvoir y stocker plus facilement les salons sans limites maximum au niveau du nombre. Cette structure de donnée nous paraissait la plus simple pour pouvoir gérer l'ajout et la suppression des salons de la façon la plus propre possible. Ainsi, les salons qu'un client peut rejoindre sont stockés dans une liste doublement chaînée, chaque nœud contient un salon. Nous avons donc essayé d'être le plus propre possible et de mettre en pratique les enseignements du S5 en algorithme et structure de données.

Avant de se connecter à un salon, le client va pouvoir créer un salon avec la commande '/create-channel' , supprimer un salon avec 'delete-channel' et enfin modifier un salon avec 'modify-channel'.
Pour créer un salon il devra entrer un numéro qui n'existe pas déjà et qui soit valide, un nom, une description, et un nombre maximum de client. Le serveur rajoutera ainsi le salon créer à la liste des salons.
Pour supprimer un salon il n'aura qu'à rentrer le numéro de ce dernier, et si personne n'y est connecté alors le serveur le supprime.
Enfin pour modifier un salon le serveur demandera d'abord au client d'entrer un numéro de salon valide et il lui demandera ensuite de donner un nouveau nom et une nouvelle description. Si aucuns autres clients se trouvent connectés à ce salon, le client pourra en plus rentrer un nouveau nombre maximum de place avant que les modifications soient bien enregistrées. Dans le cas contraire il ne pourra pas modifier le nombre de places.
Le client pourra ensuite se connecter au salon qu'il souhaite de la même façon que les versions précédentes.
(Voir diagramme de séquence en annexes page 7).

Difficultés rencontrées :

L'implémentation de la nouvelle structure de données :

Quelques difficultés ont été rencontrées mais rien de spécialement intéressant, il a simplement fallu remplacer les endroits où on utilisait l'ancienne gestion des salons par la nouvelle.

Suppression d'un salon où des clients y sont connectés :

Quand on supprimait un salon dans lequel il y avait des clients, lorsque ces derniers souhaitaient se déconnecter le serveur allait chercher dans la liste le salon pour y actualiser le tableau des places disponibles. Or, si le salon était supprimé le serveur ne trouvait pas le salon dans la liste et provoquait une erreur. Pour régler ce problème au lieu de déconnecter les clients avant de la suppression nous avons jugé plus pratique de rendre impossible la suppression d'un salon tant qu'il n'était pas vide donc tant que le nombre de places libres était différent du nombre de places max.

Répartition du travail :

	Création de la structure de donnée ListeSalon	Implémentation de la nouvelle structure de données	Ajout des fonctionnalité de création/modification/suppression	Gestion des input et des cas d'erreur	Diagramme séquence
Ayoub Moujane					
Pierre Perrin					
Julien Wiegandt					

Compiler et exécuter le code :
Attention il est important de placer les exécutables dans les bons dossiers.

Pour compiler :

Serveur : « *gcc -o serveur serveur.c* »
Client : « *gcc -o client client.c* »

Pour placer au bon endroit :

« cp salon.h serv »
« cp ListeSalon.h serv » (pour la v3 uniquement)
« cp serveur.c serv »
« cp client.c client1 »
« cp client.c client2 »
« cp client.c client3 »
« cp client.c client4 »
« cp client.c client5 »

Créer les ressources nécessaires :

Chaque dossier client doit posséder un dossier filetosend et un dossier filereceived donc dans chaque dossier client taper les commandes suivantes :
« Mkdir filetosend »
« Mkdir filereceived »
(Les dossiers sont déjà placés pour éviter les erreurs de nom de dossier)

Pour exécuter :

Il faut d'abord lancer le serveur, ensuite les clients.
Serveur : *« ./serveur »*

Pour chaque client se déplacer dans le dossier clientN
Client : *« ./client »*

Annexes :

Diagramme de séquence uml pour la version 1 et 2 :

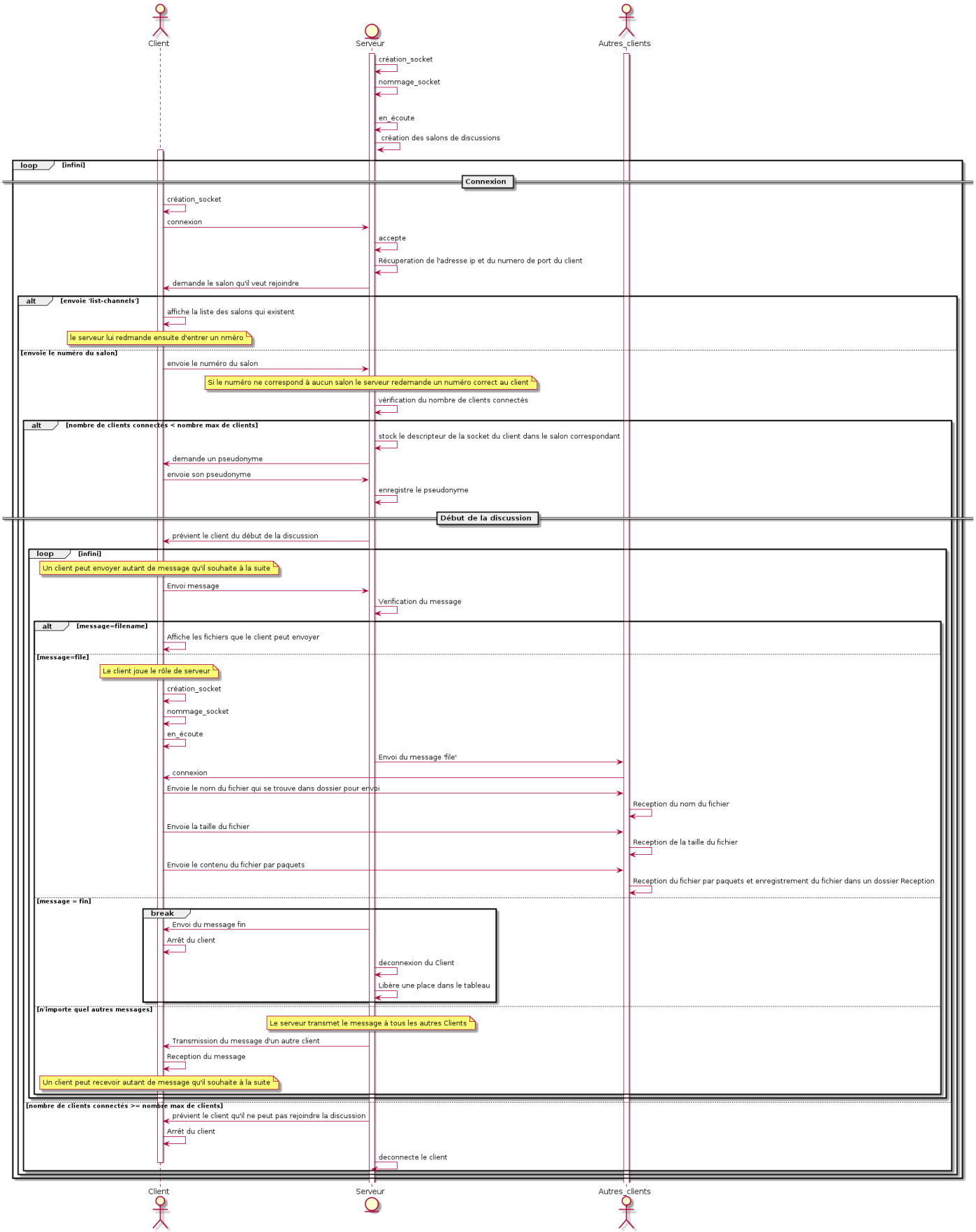

Diagramme de séquence uml pour la version 3 :